CONTROL-INDUCED LEARNING FOR AUTONOMOUS ROBOTS

by

Wanxin Jin

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

School of Aeronautics and Astronautics

West Lafayette, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Shaoshuai Mou, Chair

School of Aeronautics and Astronautics

Dr. Martin J. Corless

School of Aeronautics and Astronautics

Dr. Dengfeng Sun

School of Aeronautics and Astronautics

Dr. Dana Kulić

Department of Electrical and Computer Systems Engineering, Monash University

Approved by:

Dr. Gregory A. Blaisdell

ACKNOWLEDGMENTS

I have been very fortunate and privileged to have been given safety, health, ability, knowledge, financial well-being, support, guidance, service, happiness, and love, which have led me to writing this thesis. Thank so many people in the universe and history, who have laid the foundation of science and technology that make this thesis research possible.

This thesis would not have been possible without the huge supports from my advisor Shaoshuai Mou over the past four years. I am grateful for the chances Shaoshuai has generously given to me, allowing me to join and stay Purdue and pursue this thesis research. I have been privileged to benefit from Shaoshuai's spirit of mentorship—a blend of freedom and guidance to develop my own ideas. Shaoshuai's rigorous attitude—never been settled with answers that are not crystal clear— has profoundly shaped the way I think about research, and will forever be a fortune in my future career.

I would like also take this opportunity to especially thank all my collaborators who have contributed to this thesis work, including Prof. Dana Kulić, Prof. Sandra Hirche, Prof. Todd Murphey, and Prof. Zhaoran Wang. They have brought me to the frontiers of robotics and machine learning, and provided me a lot of advice to conduct this interdisciplinary thesis research. I also would like to thank the readers in my thesis committee, Prof. Martin J. Corless and Prof. Dengfeng Sun. My lab colleagues and friends Xuan Wang, Paulo Heredia Aguilar, Zihao Liang, Zehui Lu, Jason King Lo, Jiazhi Song, Jingqiu Zhou, Jeffery Hall, Kevin Shi, Tianyu Zhou, Jiazhen Zhou, Dawei Sun also helped me a lot during the four years at Purdue. I thank them all.

Personally, I would like to thank my parents, Sheng Jin and Xuemei Wu, for raising me in a happy family. I feel indebted to their endless love, support, and encouragement. Thank my little sister, Yuxin Jin, for putting up with me all the time. I would also like to thank my girl firend, Han Zhao, who has been always there accompanying me through every happy and sad moment of my Ph.D. life. Her accompanying, patience, and encouragement make my life wonderful and full of blessings. To my grandparents, Changchun Jin and Runnv Wang, I will be ever grateful for their love and sadly they have not lived to see my graduation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

a	non-bold letters represent scalars
$\boldsymbol{x}$	bold lowercase letters represent vectors
A	non-bold capital letters represent matrices
$\boldsymbol{A}$	bold capital letters represent block matrices
col $\{\arg_1, \arg_2, ..., \arg_n\}$	column operator stacks its arguments into a column
$\boldsymbol{x}_{k_1:k_2}$	stack of $\boldsymbol{x}$ indexed from k_1 to k_2 ($k_1 \leq k_2$); $\boldsymbol{x}_{k_1:k_2}$=col $\{\boldsymbol{x}_{k_1}, ...\boldsymbol{x}_{k_2}\}$
$\boldsymbol{A}'$	transpose of matrix $\boldsymbol{A}$
$\boldsymbol{f}(\boldsymbol{x})$	vector function $\boldsymbol{f}(\boldsymbol{x})$ with variable vector $\boldsymbol{x}$
$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}^*}$	Jacobian matrix of function $\boldsymbol{f}(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ evaluated at $\boldsymbol{x}^*$
$\boldsymbol{0}$	zero vector/matrix with dimensions implied from the context
$\boldsymbol{I}$	identify matrix whose dimensions can be implied from the context
$\sigma_i(\boldsymbol{A})$	i-th smallest singular value of $\boldsymbol{A}$; $\sigma_1(\boldsymbol{A})$ is the smallest singular value
ker $\boldsymbol{A}$	kernel of matrix $\boldsymbol{A}$
$\boldsymbol{1}_N$	all-one vector with dimension N

ABSTRACT

The recent progress of machine learning, driven by pervasive data and increasing computational power, has shown its potential to achieve higher robot autonomy. Yet, with too much focus on generic models and data-driven paradigms while ignoring inherent structures of control systems and tasks, existing machine learning methods typically suffer from data and computation inefficiency, hindering their public deployment onto general real-world robots. In this thesis work, we claim that the efficiency of autonomous robot learning can be boosted by two strategies. One is to incorporate the structures of optimal control theory into control-objective learning, and this leads to a series of control-induced learning methods that enjoy the complementary benefits of machine learning for higher algorithm autonomy and control theory for higher algorithm efficiency. The other is to integrate necessary human guidance into task and control objective learning, leading to a series of paradigms for robot learning with minimal human guidance on the loop.

The first part of this thesis focuses on the control-induced learning, where we have made two contributions. One is a set of new methods for inverse optimal control, which address three existing challenges in control objective learning: learning from minimal data, learning time-varying objective functions, and learning under distributed settings. The second is a Pontryagin Differentiable Programming methodology, which bridges the concepts of optimal control theory, deep learning, and backpropagation, and provides a unified end-to-end learning framework to solve a broad range of learning and control tasks, including inverse reinforcement learning, neural ODEs, system identification, model-based reinforcement learning, and motion planning, with data- and computation- efficient performance.

The second part of this thesis focuses on the paradigms for robot learning with necessary human guidance on the loop. We have made two contributions. The first is an approach of learning from sparse demonstrations, which allows a robot to learn its control objective function only from human-specified sparse waypoints given in the observation (task) space; and the second is an approach of learning from human's directional corrections, which enables a robot to incrementally learn its control objective, with guaranteed learning convergence, from human's directional correction feedback while it is acting.

1. INTRODUCTION

The past decades see the notable progress in development of autonomous robots, from high-skilled legged robots, autonomous aerial and ground vehicles, to planetary/space platforms. In the control system perspective, the behavior of an autonomous robot can be specified by the following aspects: dynamics, which dictates the evolution rule of the robot states given control inputs, a task objective, which encapsulates the goal or utility of the robot behavior; a control policy, which defines how the control inputs are generated given the robot states, and additional constraints imposed by the environments and task. These aspects constituting an autonomous robot are schematically shown in Fig. 1.1.

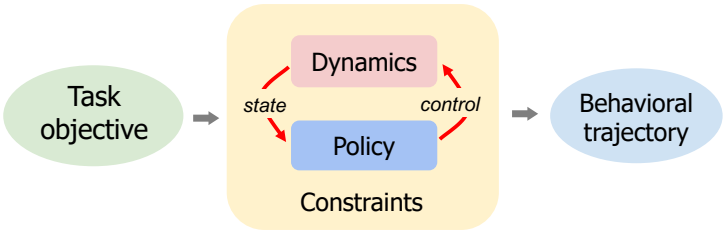


Figure 1.1. A schematical description of an autonomous robot.

Throughout this thesis, we adopt the following hierarchical perspective to understand different aspects of an autonomous robot. First, a task objective can be thought of as a high-level compact policy specification, which encodes task goal, control principle, and other contextual information for the robot behavior. Second, a task objective and robot dynamics jointly determine the robot specific policy, which finally leads to the robot specific behavioral trajectory. Towards higher robot autonomy, many research work in machine learning and control fields focuses on how to *automate* the programming of different aspects of an autonomous robot. Throughout this thesis, we call the general problem of obtaining the mathematical specifications for different aspects of an autonomous robot from experience/demonstration data as *autonomous robot learning*.

Two different frameworks have been developed in machine learning and control fields to formulate an autonomous robot. In machine learning, an autonomous robot is typically studied under reinforcement learning framework [1], which provides a generic date-driven

paradigm to enable a robot to find an optimal policy through its experience in environment and rewarding signals. In control, an autonomous robot is usually studied based on optimal control theory [2], which offers systematic tools to solve for the optimal behavioral trajectory of a robot given the model specification of a task. The pros and cons for the above two frameworks are complementary. Machine learning emphasizes more on experience data and less on detailed structures of a control system. Thus, most learning algorithms enjoy the high-level autonomy (i.e., requiring less human specifications), but suffer from huge data and computation complexity especially for high-dimensional and continuous tasks. For optimal control methods, while decades of successful applications have arguably demonstrated its capability to handle challenging problems, most control algorithms are built upon the precise specification of the system and task, such as dynamics, control architectures, optimization formulations, etc, which requires practitioners to have high expertise and knowledge about robot/task programming and thus may degrade the autonomy of the algorithm itself.

Given that the benefits of machine learning and control methods are largely complementary, this thesis is motivated to integrate the benefits of both for more efficient robot learning algorithms. The first fundamental question to address is:

Question 1: can we incorporate the fundamental structures from optimal control theory into the autonomous robot learning to maintain its autonomy while expedite its efficiency? We claim to inject optimal control theory into the robot learning paradigm. Such an injection can lead to a series of *control-induced learning* methods, which maintain high-level autonomy (i.e., data-driven and without manually programming robots) like generic machine learning algorithms, at the same time inheriting high efficiency and scalability from control methods for handling high-dimensional challenging tasks.

Another way to improve the efficiency of autonomous robot learning is to involve human guidance on the loop. Two sources of motivations can be identified. First, because humans are generally adequate for high-level decision making and contextual rationality, human guidance can be goal-and-policy informative, and thus, if properly leveraged, can inform the robot's search direction or reduce its search space during learning progress, potentially increasing its efficiency. Second, with the ultimate goal of deploying robots into people's

daily life, autonomous robots are desired to have custom autonomy; learning with human's guidance enables a robot to quickly learn and adapt to human user's preference. With these motivations in mind, the second goal of this thesis is to answer the following question:

Question 2: how can we make the most of human guidance to boost the efficiency of robot learning, while maintain the human's burden in providing guidance as low as possible? This thesis addresses the above question in two directions. First, we explore the learning paradigm that enables a robot to learn only from human's *sparse* inputs; and second, we explore a robot learning paradigm that allows a human to provide guidance in his/her most intuitive and natural way.

1.1 Summary of Research Contributions

The first portion of the thesis aim to answer *Question 1* by developing some foundational control-induced learning methodologies.

Chapter 2 presents a set of new methods for inverse optimal control, where the key problem of interest is to learn an objective function of an optimal control system given the system trajectories subject to optimality principles. These new inverse optimal control methods address the existing gaps in objective learning techniques. Specifically, Sections 2.2 and 2.3 answer the questions of what is the relationship between an incomplete trajectory data and the objective function parameters, and how to enable efficient objective learning from limited data. Section 2.4 addresses the question of how to learn time-varying objective functions underlying a long and continuous system trajectory. Section 2.5 addresses the question of how to learn an objective function in the context where both data and computational resources are distributed. Section 2.6 focuses on some novel applications of the developed inverse optimal control methods, including human motion prediction and segmentation.

Chapter 3 proposes a Pontryagin Differentiable Programming (PDP) methodology. By integrating the benefits of optimal control theory, deep learning, and backpropagation, PDP establishes a unified framework to solve a broad class of learning and control tasks. PDP distinguishes from existing methods by two novel techniques: first, we differentiate through Pontryagin's Maximum/Minimum Principle, and this allows to obtain the analytical deriva-

tive of a trajectory with respect to tunable parameters within an optimal control system, enabling end-to-end learning of dynamics, policies, or/and control objective functions; and second, we propose an auxiliary control system in backward pass of learning, and the output of this auxiliary control system is the analytical derivative of the original system's trajectory with respect to the parameters, which can be iteratively solved using standard control tools. PDP is shown flexible enough to be customized for different learning and control problems, including inverse reinforcement learning (inverse optimal control) [3, 4], model-based reinforcement learning such as model-based policy optimization [5, 6], motion planning (optimal control) [7, 8], learning neural ODEs (system identification) [9, 10], and efficient enough to solve high-dimensional and continuous-space problems.

The second potion of this thesis focuses on robot learning with necessary human guidance on the loop, attempting to answer *Question 2*.

Chapter 4 develops the method of learning from sparse demonstrations. Learning from demonstrations (LfD) empowers a non-expert human user to program a robot by only providing demonstrations. The proposed method addresses the gaps of existing LfD techniques with the following fundamental features. First, the method enables a robot to learn an objective function only from human's sparse demonstrations, which consist of a small number of desired waypoints the human wants the robot trajectory to follow at some sparse time instances. Second, the method finds an objective function within a given function set such that the robot trajectory has minimum distance to the sparse demonstrations, even though the sparse demonstrations may be sub-optimal or even are randomly given. Third, the method allows for the time-inconsistency between demonstrations and robot dynamics, by jointly learning a time-warping function to align the duration between the human's sparse demonstration and the feasible motion of the robot.

Chapter 5 develops the method of learning from directional corrections. This new learning scheme enables a non-expert human user to teach a robot by improving the robot's motion while it is acting. For instance, consider a robot that plans its motion under a (random) control objective function. While it is executing the motion, a human user who supervises the robot finds the robot's motion not satisfactory; thus, the human user applies a correction

to the robot during its motion execution. Then, the robot leverages the correction to update its control objective function. This process of planning-correction-update repeats until the robot eventually achieves a control objective function such that its resulting trajectory agrees with the human user's expectation. In addition to the incremental learning capability, this new method also has the following enabling features. First, the method only requires human's directional corrections. For instance, to teach a mobile robot, the human's directional corrections are simply as 'left' or 'right' without dictating how far the robot should move. Second, the human's directional corrections to the robot's motion can be sparse; that means that the corrections can be applied only at sparse time instances within the time horizon of the robot's motion. Finally, the theoretical results are established to show the convergence of the proposed learning algorithm.

1.2 Summary of Publications and Open Source Codes

The content of Chapter 2 appears in:

- Wanxin Jin, Dana Kulić, Shaoshuai Mou, and Sandra Hirche. "Inverse optimal control from incomplete trajectory observations". In: *The International Journal of Robotics Research*. 2021, 40(6-7), pp. 848-865.
- Wanxin Jin, Dana Kulić, Shaoshuai Mou, and Sandra Hirche. "Inverse Optimal Control for Multiphase Cost Functions". In: *IEEE Transactions on Robotics*. 2019, 35(6), pp. 1387-1398.
- Wanxin Jin and Shaoshuai Mou, "Distributed Inverse Optimal Control". In: *Automatica*. 2021.
- Wanxin Jin, Zihao Liang, and Shaoshuai Mou, "Inverse Optimal Control from Demonstration Segments". In: *arXiv preprint arXiv:2010.15034*. 2020. Access at https://arxiv.org/abs/2010.15034

The code and experiments developed for the content of Chapter 2 can be accessed at

- https://github.com/wanxinjin/IOC-from-Incomplete-Trajectory-Observations

The content of Chapter 3 appears in:

- Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. "Pontryagin Differentiable Programming: An End-to-End Learning and Control Framework". In: *Advances in Neural Information Processing Systems*. 2020.

The code and experiments developed for the content of Chapter 3 can be accessed at

- https://github.com/wanxinjin/Pontryagin-Differentiable-Programming

The content of Chapter 4 appears in:

- Wanxin Jin, Todd D. Murphey, Dana Kulić, Neta Ezer, Shaoshuai Mou. "Learning from Sparse Demonstrations". Submitted to: *IEEE Transactions on Robotics*. 2020. Also access at arXiv preprint arXiv:2008.02159

The code and experiments developed for the content of Chapter 4 can be accessed at

- https://github.com/wanxinjin/Learning-from-Sparse-Demonstrations

The content of Chapter 5 appears in:

- Wanxin Jin, Todd D. Murphey, and Shaoshuai Mou. "Learning from Incremental Directional Corrections". Submitted to: *IEEE Transactions on Robotics*. 2020. Also access at arXiv preprint arXiv:2011.15014

The code and experiments developed for the content of Chapter 5 can be accessed at

- https://github.com/wanxinjin/Learning-from-Directional-Corrections

Other research work (pre-prints/submissions) which is not included in this thesis:

- Wanxin Jin, Shaoshuai Mou, and George J. Pappas. "Safe Pontryagin Differentiable Programming". Access at arXiv preprint arXiv:2105.14937
- Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. "Neural Certificates for Safe Control Policies". Access at arXiv preprint arXiv:2006.08465

PART I

CONTROL-INDUCED LEARNING

2. NEW METHODS FOR INVERSE OPTIMAL CONTROL

This chapter focuses on the problem of inverse optimal control (IOC), which seeks to learn an objective function of an optimal control system from its optimal trajectory. This is the key to achieve higher robot autonomy, and its significance includes two aspects. First, IOC enables a robot to automatically learns its control objective function (then from which deriving its control policy) by observing a demonstration of the task, thus eliminating the need of explicit robot programming. Second, since optimal control models achieve arguably success in explaining human's rational behavior, the ability of inferring control objective function makes it possible to perform human-robot missions, where learning of objective functions is the key for human motion prediction and human-robot coordination.

In this chapter, we present a series of new methods for inverse optimal control, including incremental objective learning from limited data; learning multi-phase objective functions; and distributed objective learning. The final section of this chapter presents some novel applications of the developed IOC methods in human motion analysis. The success of these inverse optimal control methods is based on fully exploiting and integrating optimal control structures into the learning formulation, as we have mentioned in the introduction chapter. The contents of this chapter have been published at [11, 12, 13, 14]. The code developed for this chapter can be accessed at https://github.com/wanxinjin/IOC-from-Incomplete-Trajectory-Observations.

2.1 Introduction

As described in Introduction (Section 1), the key aspects that characterize the behavior of an autonomous robot includes control objective, dynamics, and policy. Such a robot is typically modeled as an optimal control system [15] in the control field, and a reinforcement learning agent [1] in machine learning field. In both fields, a large number of algorithms have focused on enabling a robot to find its trajectory/policy that optimizes an given objective function. Providing an appropriate control objective function is the first step and the key to task accomplishment. However, finding a good objective function can be challenging—desired robot behaviors can take hundreds of person-hours in objective parameter tuning, and

this challenge is especially true when multi-attribute goals need to be harmonized. Therefore, there is an urgent need for automating the process of designing a robot control objective function. Another source that motivates the objective function learning is the need to analyze the behavior of natural agents, such as human motor control [16], which has extensively shown behavioral optimality [16, 17, 18, 19, 20, 21, 22, 23]. Discovering the control objective functions underlying the observed behaviors will facilitate the understanding and prediction of human motion and make it possible to achieve seamless human-robot autonomy. All these practical needs motivate a fundamental problem: can we finding the underlying objective function from optimal behavioral demonstrations?

In the control field, the above problem of finding objective functions is referred to as inverse optimal control (IOC), which was firstly posed in [4]. In the machine learning field, within the context of reinforcement learning, the above problem is referred to as inverse reinforcement learning (IRL), which was first studied in [3]. With the progress achieved in both fields, the techniques of IOC and IRL have been successfully applied in various applications, including learning from demonstrations (imitation learning) [24, 25], where a learner mimics an expert by inferring an objective function from the expert's demonstrations, autonomous driving [26], where human driving preference is learned and transferred to a vehicle controller, human-robot systems [23, 27], where the intentionality of a human partner is estimated to enable motion prediction and smooth coordination, and human motion analysis [11, 28], where principles of human motor control are investigated.

Both IOC and IRL aim to achieve the same goal of learning objective, the main aspect to distinguish each other is the context and description of the forward problem. IOC describes an autonomous robot using optimal control formulation, which is typically (dynamics) model-based, while IRL using reinforcement learning framework, which can be data-driven (some popular IRL algorithms [3, 29, 30, 31] still directly or indirectly reply on dynamics models for learning efficiency). Since the gist of this thesis is control-induced learning for autonomous robots, we preferably use IOC to refer to the general objective learning. We will not distinguish between both in the following literature review when how to solve the forward problem is not important.

2.1.1 Related work

The most common strategy in IOC/IRL is to parametrize an unknown objective function as a weighted sum of relevant features (or basis functions) with unknown weights [3, 22, 29, 32, 33, 34]. Different approaches have been developed to estimate the weights given observations of the system optimal trajectory over a complete time horizon. Existing IOC/IRL methods can be categorized based on whether the forward optimal control (or reinforcement learning) problem needs to be solved during learning process.

The first category of existing work is based on a nested architecture, where the feature weights are updated in an outer loop while the corresponding forward problem (optimal control or reinforcement learning) is solved in an inner loop. Different methods of this type focus on different strategies to update the feature weights in the outer layer. Representative strategies include feature matching [32], where the feature weights are updated towards matching the feature values of the reproduced optimal trajectories with the demonstrations, maximum margin [33], where the feature weights are solved by maximizing the margin between the objective function value of the observed trajectories and the value of any simulated optimal trajectories maximum entropy [29], where the feature weights are optimized such that the probability distribution of system's trajectories maximizes the entropy while matching the empirical feature values of demonstrations; and direct loss minimization [22, 35], where the weights are learned by directly minimizing the distance between the reproduced trajectory and demonstrations are minimized. The above nested IOC methods have been successfully applied to humanoid locomotion [20, 21] and arm motion [36], autonomous vehicles [26], robot navigation [37], learning from human corrections [38, 39], etc.

The nested IOC methods require to solve optimal control problems repeatedly at each update of objective function parameters, thus those methods usually suffer from relatively high computational cost. This motivates the direct IOC methods, which directly solve for the unknown feature weights. A key idea used in the direct methods is to establish optimality conditions which the observed optimal data must satisfy. For example, in [40], the Karush-Kuhn-Tucker (KKT) optimality conditions [41] are established, based on which the feature weights are then solved by minimizing a loss that quantifies the violation of such conditions by

the observed data. In [42], the authors apply the KKT-based method to solve IOC problems and study the control objective function for human locomotion. In [43], the Pontryagin's Minimum Principle [44] is utilized to formulate a residual optimization over the unknown weights. These methods have been successfully applied to the locomotion analysis [42, 45], walking path generation [46], human motion segmentation [11, 28], etc. In [34], the authors propose an inverse KKT method to enable a robot to learn manipulation tasks. Recently, along this direction, the recoverability for IOC problems has been investigated. For example, when an optimal control system remains at an equilibrium point, although its trajectory still satisfies the optimality conditions, it is uninformative for learning the objective function. This issue is discussed in [47, 48], where a sufficient condition for recovering weights from full trajectory observations is proposed.

2.1.2 Challenges

Despite the significant progress of IOC/IRL techniques, these still exist the following technical challenges unresolved.

(1) *Requiring **complete** trajectory observations.* To find the objective function, existing IOC/IRL techniques require as input complete episodes of trajectory demonstrations. This brings out the following limitations. First, in some cases the full episodes of demonstrations are not accessible due to such as limited sensing capabilities or occlusions, and only part of episodes are available. Second, processing full trajectory data always leads to large consumption of memory space and also expensive computational costs. Third, by requiring full trajectory observation, objective learning thus can not be performed in online settings.

(2) *Assuming **single-phase** objective function.* Existing IOC/IRL techniques assume that the unknown objective function is static and not changing. Such assumption might not be valid for the autonomous agents that perform complex and long-term tasks. In such cases, the objective function adopted by an autonomous agent may vary depending on different motion phases or contextual conditions. For example,

the study of human motor motion has already shown various supporting evidences in human motor control [16, 28].

(3) *Assuming* ***centralized*** *learning.* Existing IOC/IRL techniques learn the objective function in a centralized way, meaning that both data acquisition and computational process are performed by a single processor. This centralized learning paradigm is restricted given limited processing and memory capacity of a single processor. Thus, there is need to develop distributed learning algorithms that enable to distribute the learning task to multiple processors, in which each processor only accounts for partial data and has computation.

2.1.3 Chapter Organization

In the following sections are organized as follows. Section 2.2 presents the basic formulation for IOC problems, and then develop some foundational theories for solving IOC problems. Section 2.3 presents the method and algorithm of IOC with incomplete trajectory observations, which address the first research gap identified above. Section 2.4 presents the method and algorithm of IOC for multi-phase objective functions, addressing the second research gap identified above. Section 2.5 presents a distributed inverse optimal control technique, which addresses the third technical challenge stated above. Finally, Section 2.6 presents some novel applications of the developed IOC algorithms.

2.2 IOC and Recovery Matrix

In this section, we present the general formulation of inverse optimal control, and the concept of *recovery matrix*, a key quantity that establishes the bridge between data and the objective function parameters, which will serve as a foundation for developing efficient inverse optimal control in later sections. Also, we will present the properties of the recovery matrix, which will provide insights of objective learning process.

2.2.1 IOC Problem Formulation

Consider an autonomous agent with the following dynamics and initial condition:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad \boldsymbol{x}_0 \in \mathbb{R}^n, \tag{2.1}$$

where the vector function $\boldsymbol{f} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ is differentiable; $\boldsymbol{x}_k \in \mathbb{R}^n$ is the system state; $\boldsymbol{u}_k \in \mathbb{R}^m$ is the control input; and $k = 0, 1, \cdots$ is the time step. Suppose that the agent trajectory of states and inputs over a horizon T, denoted as,

$$\boldsymbol{\xi} = \{\boldsymbol{\xi}_k : k = 0, 1, ..., T\} \quad \text{with} \quad \boldsymbol{\xi}_k{=}(\boldsymbol{x}_k^*,\, \boldsymbol{u}_k^*), \tag{2.2}$$

(locally) minimizes a control cost function

$$J(\boldsymbol{x}_{0:T}, \boldsymbol{u}_{0:T}) = \sum_{k=0}^{T} \boldsymbol{\omega}' \boldsymbol{\phi}^*(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{2.3}$$

where $\boldsymbol{\omega}'\boldsymbol{\phi}^*(\cdot, \cdot)$ is the *running (stage) cost*. Here $\boldsymbol{\phi}^* : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^s$ is called a *relevant feature vector* and defined as a column of a *relevant feature set*

$$\mathcal{F}^* = \{\phi_1^*, \phi_2^*, \cdots, \phi_s^*\}, \tag{2.4}$$

that is, $\boldsymbol{\phi}^* = \text{col}\, \mathcal{F}^*$, with ϕ_i^* being the ith *feature* for the running cost, and $\boldsymbol{\omega} \in \mathbb{R}^s$ is called the *weight vector*, with the ith entry ω_i corresponding to ϕ_i^*. This type of weighted-feature objective function is commonly used in objective learning problems [29, 32, 48], and has been successfully applied in a wide range of real-world applications [26, 28, 34, 49].

The dynamics (2.1) and cost function (2.3) can represent different optimal control settings as follows. (I) Finite-horizon free-end optimal control: the finite horizon T is given but the final state $\boldsymbol{x}_{T+1}$ is free, i.e., no constraint on $\boldsymbol{x}_{T+1}$; (II) finite-horizon fixed-end optimal control: both the finite horizon T and the final state $\boldsymbol{x}_{T+1} = \boldsymbol{x}_{\text{goal}}$ are given; and (III) infinite-horizon optimal control: $T = \infty$. Besides, one can consider the finite-horizon optimal

control, where the final state $\boldsymbol{x}_{T+1}$ is penalized using a final cost term added to (2.3), and this case can be viewed as an extension similar to (II).

In IOC problems, one is given a relevant feature set $\mathcal{F}^*$, the goal is to obtain an estimate of the weights $\boldsymbol{\omega}$ from observations of $\boldsymbol{\xi}$. Note that $\boldsymbol{\omega}$ can only be determined up to a non-zero scaling factor [40, 48], because any $c\boldsymbol{\omega}$ with $c > 0$ will lead to the same trajectory $\boldsymbol{\xi}$. Hence we say an estimate $\hat{\boldsymbol{\omega}}$ is a *successful estimate* of $\boldsymbol{\omega}$ if $\hat{\boldsymbol{\omega}} = c\boldsymbol{\omega}$ with $c \neq 0$, and the specific $c > 0$ can be determined by normalization [34, 40].

2.2.2 Recovery Matrix

In this part, we will introduce the key concept of the *recovery matrix* and show its relationship to solving IOC. The recovery matrix is defined on a segment of the agent optimal trajectory in (2.2),

$$\boldsymbol{\xi}_{t:t+l} = \{\boldsymbol{\xi}_k : t \leq k \leq t+l\} \subseteq \boldsymbol{\xi}, \tag{2.5}$$

Here, $\boldsymbol{\xi}_{t:t+l}$ is a segment of $\boldsymbol{\xi}$ in (2.2) within the time interval $[t, t+l] \subseteq [0, T]$, with t called the *starting time* of the segment and $l = 1, 2, \cdots$ called the *segment length*, $0 \leq t < t+l \leq T$. We first present the definition of the recovery matrix, then show its relationship to solving IOC, which is also the motivation of the recovery matrix.

Definition 2.2.1. *Let a segment of the trajectory, $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$ in (2.5), and a candidate feature set $\mathcal{F} = \{\phi_1, \phi_2, \cdots, \phi_r\}$ be given. Let $\boldsymbol{\phi} = \operatorname{col} \mathcal{F}$. Then the recovery matrix, denoted by $\boldsymbol{H}(t, l)$, is defined as:*

$$\boldsymbol{H}(t,l) = \begin{bmatrix} \boldsymbol{H}_1(t,l) & \boldsymbol{H}_2(t,l) \end{bmatrix} \in \mathbb{R}^{ml \times (r+n)}, \tag{2.6}$$

with

$$\boldsymbol{H}_1(t,l) = \boldsymbol{F}_u(t,l)\boldsymbol{F}_x^{-1}(t,l)\boldsymbol{\Phi}_x(t,l) + \boldsymbol{\Phi}_u(t,l), \tag{2.7}$$

$$\boldsymbol{H}_2(t,l) = \boldsymbol{F}_u(t,l)\boldsymbol{F}_x^{-1}(t,l)\boldsymbol{V}(t,l). \tag{2.8}$$

Here, $\boldsymbol{F}_x(t,l)$, $\boldsymbol{F}_u(t,l)$, $\boldsymbol{\Phi}_x(t,l)$, $\boldsymbol{\Phi}_u(t,l)$ *and* $\boldsymbol{V}(t,l)$ *are defined as*

$$
\begin{aligned}
\boldsymbol{F}_x(t,l) &= \begin{bmatrix} \boldsymbol{I} & \frac{-\partial \boldsymbol{f}'}{\partial \boldsymbol{x}_{t+1}^*} & & \\ \boldsymbol{0} & \boldsymbol{I} & \ddots & \\ & & \ddots & \frac{-\partial \boldsymbol{f}'}{\partial \boldsymbol{x}_{t+l-1}^*} \\ & & & \boldsymbol{I} \end{bmatrix} \in \mathbb{R}^{nl\times nl}, \qquad \boldsymbol{\Phi}_x(t,l) = \begin{bmatrix} \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{x}_{t+1}^*} \\ \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{x}_{t+2}^*} \\ \cdots \\ \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{x}_{t+l}^*} \end{bmatrix}' \in \mathbb{R}^{nl\times r}, \\
\boldsymbol{F}_u(t,l) &= \begin{bmatrix} \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_t^*} & & & \\ & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_{t+1}^*} & & \\ & & \ddots & \\ & & & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_{t+l-1}^*} \end{bmatrix} \in \mathbb{R}^{ml\times nl}, \qquad \boldsymbol{\Phi}_u(t,l) = \begin{bmatrix} \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{u}_t^*} \\ \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{u}_{t+1}^*} \\ \cdots \\ \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{u}_{t+l-1}^*} \end{bmatrix}' \in \mathbb{R}^{ml\times r}, \\
\boldsymbol{V}(t,l) &= \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \cdots & \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_{t+l}^*} \end{bmatrix}' \in \mathbb{R}^{nl\times n},
\end{aligned}
\tag{2.9}
$$

respectively.

Before showing the relationship between the recovery matrix and IOC, we impose the following assumption on the given candidate feature set $\mathcal{F}$ in Definition 2.2.1.

Assumption 2.2.1. *In Definition 2.2.1, the candidate feature set* $\mathcal{F} = \{\phi_1, \phi_2, \cdots, \phi_r\}$ *contains as a subset the relevant features* $\mathcal{F}^*$ *in (2.4), i.e.,* $\mathcal{F}^* \subseteq \mathcal{F}$.

Assumption 2.2.1 requires that the relevant features $\mathcal{F}^*$ in (2.4) are contained by the given candidate feature set $\mathcal{F}$, which means that $\mathcal{F}$ also allows for including additional features that are irrelevant to the optimal control system. Although restrictive for choice of features, this assumption is likely to be fulfilled in implementation by providing a larger set including many features when the knowledge of exact relevant features is not available. Under Assumption 2.2.1, without loss of generality, we let

$$
\mathcal{F} = \{\phi_1^*, \phi_2^*, \cdots, \phi_s^*, \tilde{\phi}_{s+1}, \cdots, \tilde{\phi}_r\}, \tag{2.10}
$$

that is, the first s elements are from $\mathcal{F}^*$ in (2.4). Then we have

$$\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{u}) = \text{col } \mathcal{F} = \begin{bmatrix} \boldsymbol{\phi}^*(\boldsymbol{x},\boldsymbol{u}) \\ \tilde{\boldsymbol{\phi}}(\boldsymbol{x},\boldsymbol{u}) \end{bmatrix} \in \mathbb{R}^r, \tag{2.11}$$

where $\boldsymbol{\phi}^* \in \mathbb{R}^s$ are the relevant feature vector in (2.3) while $\tilde{\boldsymbol{\phi}} \in \mathbb{R}^{(r-s)}$ corresponds to the features that are not in $\mathcal{F}^*$. We define a weight vector

$$\bar{\boldsymbol{\omega}} = \text{col } \{\boldsymbol{\omega}, \boldsymbol{0}\} \in \mathbb{R}^r \tag{2.12}$$

corresponding to (2.11), where $\boldsymbol{\omega}$ are the weights in (2.3) for $\boldsymbol{\phi}^*$. Based on (2.3), we can say that the agent optimal trajectory $\boldsymbol{\xi}$ in (2.2) also (locally) minimize the cost function of

$$J(\boldsymbol{x}_{0:T}, \boldsymbol{u}_{0:T}) = \sum_{k=0}^{T} \bar{\boldsymbol{\omega}}' \boldsymbol{\phi}(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{2.13}$$

with the dynamics and initial condition in (2.1). Next, we will distinguish the three optimal control settings, as described in the IOC problem formulation in the previous part, and establish the relationship between the recovery matrix and the IOC problem solution.

Case I: Finite-Horizon Free-End Optimal Control

We first consider the optimal control setting with finite horizon T and free final state $\boldsymbol{x}_{T+1}$. In this case, given the cost function (2.13) and the dynamics constraint (2.1), one can define the following Lagrangian:

$$L = J(\boldsymbol{x}_{0:T}, \boldsymbol{u}_{0:T}) + \sum_{k=0}^{T} \boldsymbol{\lambda}_{k+1}' \Big(\boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) - \boldsymbol{x}_{k+1} \Big), \tag{2.14}$$

where $\boldsymbol{\lambda}_{k+1} \in \mathbb{R}^n$, $k = 0, 1, \ldots, T$, is Lagrange multipliers. According to the Karush-Kuhn-Tucker (KKT) conditions [41], there exist multipliers $\boldsymbol{\lambda}^*_{1:T+1} = \text{col } \{\boldsymbol{\lambda}^*_1, \boldsymbol{\lambda}^*_2, ..., \boldsymbol{\lambda}^*_T, \boldsymbol{\lambda}^*_{T+1}\}$,

also referred to as costates, such that the optimal trajectory $\boldsymbol{\xi}$ must satisfy the following conditions

$$\frac{\partial L}{\partial \boldsymbol{x}^*_{1:T+1}} = \mathbf{0}, \tag{2.15a}$$

$$\frac{\partial L}{\partial \boldsymbol{u}^*_{0:T}} = \mathbf{0}. \tag{2.15b}$$

Based on the definitions in (2.9), the equations in (2.15a) and (2.15b) can be written as

$$-\boldsymbol{F}_x(0,T)\boldsymbol{\lambda}^*_{1:T}+\boldsymbol{\Phi}_x(0,T)\bar{\boldsymbol{\omega}} = \mathbf{0} = -\boldsymbol{V}(0,T)\boldsymbol{\lambda}^*_{T+1}, \tag{2.16a}$$

$$\boldsymbol{F}_u(0,T)\boldsymbol{\lambda}^*_{1:T}+\boldsymbol{\Phi}_u(0,T)\bar{\boldsymbol{\omega}} = \mathbf{0}, \tag{2.16b}$$

respectively, where in (2.16a), $\boldsymbol{\lambda}^*_{T+1} = \mathbf{0}$ directly results from extending (2.15a) at the final state $\boldsymbol{x}_{T+1}$. The optimality equations in (2.16) are established for complete trajectory $\boldsymbol{\xi}$. Given any segment of the trajectory, say $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$ in (2.5), the following equations can be obtained by partitioning (2.16a) and (2.16b) in the corresponding rows,

$$-\boldsymbol{F}_x(t,l)\boldsymbol{\lambda}^*_{t+1:t+l} + \boldsymbol{\Phi}_x(t,l)\bar{\boldsymbol{\omega}} = -\boldsymbol{V}(t,l)\boldsymbol{\lambda}^*_{t+l+1}, \tag{2.17a}$$

$$\boldsymbol{F}_u(t,l)\boldsymbol{\lambda}^*_{t+1:t+l} + \boldsymbol{\Phi}_u(t,l)\bar{\boldsymbol{\omega}} = \mathbf{0}, \tag{2.17b}$$

respectively. For the above (2.17), we note that when $\boldsymbol{\xi}_{t:t+l} = \boldsymbol{\xi}_{0:T}$, i.e., when the observation is the complete trajectory data $\boldsymbol{\xi}$, (2.17) will become (2.16). Thus, a complete trajectory observation can be viewed as a special case of an incomplete trajectory observation.

Case II: Finite-Horizon Fixed-End Optimal Control.

We next consider the optimal control setting with a finite horizon T and a given fixed final state $\boldsymbol{x}_{T+1} = \boldsymbol{x}_{\text{goal}}$. Given the cost function (2.13), the dynamics (2.1), and the final state constraint $\boldsymbol{x}_{T+1} = \boldsymbol{x}_{\text{goal}}$, one can define the following Lagrangian:

$$L=J(\boldsymbol{x}_{0:T}, \boldsymbol{u}_{0:T}) + \sum_{k=0}^{T} \boldsymbol{\lambda}'_{k+1}\Big(\boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k)-\boldsymbol{x}_{k+1}\Big) + \boldsymbol{\lambda}'_{\text{goal}}(\boldsymbol{x}_{T+1}-\boldsymbol{x}_{\text{goal}}), \tag{2.18}$$

where the difference from (2.14) is that the term $\boldsymbol{\lambda}'_{\text{goal}}(\boldsymbol{x}_{T+1} - \boldsymbol{x}_{\text{goal}})$ is added since the final state is subject to the given $\boldsymbol{x}_{\text{goal}}$ constraint, and $\boldsymbol{\lambda}_{\text{goal}} \in \mathbb{R}^n$ is the associated Lagrangian multiplier. Following a similar derivation as in Case I, one obtains the same equations in (2.17) for any segment data of the trajectory $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$. Here, the only difference from Case I is that when $\boldsymbol{\xi}_{t:t+l} = \boldsymbol{\xi}_{0:T}$, one usually has $\boldsymbol{\lambda}^*_{T+1} = \boldsymbol{\lambda}^*_{\text{goal}} \neq \mathbf{0}$ in this case due to the fixed final state constraint, while $\boldsymbol{\lambda}^*_{T+1} = \mathbf{0}$ in Case I. In addition, for the finite-horizon optimal control, in which the final state $\boldsymbol{x}_{T+1}$ is penalized using a final cost term added to (2.3), we can derive the similar result of $\boldsymbol{\lambda}^*_{T+1} \neq \mathbf{0}$.

Case III: Infinite-Horizon Optimal Control.

For the infinite-horizon optimal control setting, the optimal trajectory $\boldsymbol{\xi}$ is more conveniently characterized by the Bellman optimality condition [2]:

$$V(\boldsymbol{x}^*_k) = \bar{\boldsymbol{\omega}}'\boldsymbol{\phi}(\boldsymbol{x}^*_k, \boldsymbol{u}^*_k) + V(\boldsymbol{f}(\boldsymbol{x}^*_k, \boldsymbol{u}^*_k)), \tag{2.19}$$

where $V(\boldsymbol{x}^*_k)$ is the (unknown) optimal cost-to-go function evaluated at state $\boldsymbol{x}^*_k$. Next, we differentiate the Bellman optimality equation in (2.19) on both sides with respect to $\boldsymbol{x}^*_k$ while denoting $\boldsymbol{\lambda}^*_k = \frac{\partial V(\boldsymbol{x}_k)}{\partial \boldsymbol{x}^*_k} \in \mathbb{R}^n$, and then obtain

$$\boldsymbol{\lambda}^*_k = \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_k}\bar{\boldsymbol{\omega}} + \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_k}\boldsymbol{\lambda}^*_{k+1}. \tag{2.20}$$

Differentiating the Bellman optimality equation (2.19) on both sides with respect to $\boldsymbol{u}^*_k$ yields

$$\mathbf{0} = \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_k}\bar{\boldsymbol{\omega}} + \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_k}\boldsymbol{\lambda}^*_{k+1}. \tag{2.21}$$

For any available trajectory segment $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$, we stack equation (2.20) for all $\boldsymbol{x}^*_{t+1:t+l}$ and stack equation (2.21) for all $\boldsymbol{u}^*_{t:t+l-1}$, and obtain the same equations in (2.17).

From the above analysis, we conclude that, for any trajectory segment $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$, regardless of the corresponding optimal control problem type, we can always use the segment data $\boldsymbol{\xi}_{t:t+l}$ to establish the equations (2.17). Thus, in what follows, we do not distinguish the specific

optimal control settings, and only focus on equations (2.17) to show the relationship between the recovery matrix in Definition 2.2.1 and IOC problem solution.

By noticing that $\boldsymbol{F}_x(t,l)$ in (2.17a) is always invertible, we combine (2.17a) with (2.17b) and eliminate $\boldsymbol{\lambda}^*_{t+1:t+l}$, which then yields

$$\Big(\boldsymbol{F}_u(t,l)\boldsymbol{F}_x^{-1}(t,l)\boldsymbol{\Phi}_x(t,l) \;+\; \boldsymbol{\Phi}_u(t,l)\Big)\bar{\boldsymbol{\omega}} \;+\; \Big(\boldsymbol{F}_u(t,l)\boldsymbol{F}_x^{-1}(t,l)\boldsymbol{V}(t,l)\Big)\boldsymbol{\lambda}^*_{t+l+1} \;=\; \mathbf{0}. \quad (2.22)$$

Considering the definition of the recovery matrix in (2.6)-(2.8), (2.22) can be written as

$$\boldsymbol{H}_1(t,l)\bar{\boldsymbol{\omega}} + \boldsymbol{H}_2(t,l)\boldsymbol{\lambda}^*_{t+l+1} = \boldsymbol{H}(t,l)\begin{bmatrix}\bar{\boldsymbol{\omega}}\\ \boldsymbol{\lambda}^*_{t+l+1}\end{bmatrix} = \mathbf{0}. \quad (2.23)$$

Equation (2.23) reveals that the weights $\bar{\boldsymbol{\omega}}$ and costate $\boldsymbol{\lambda}^*_{t+l+1}$ must satisfy a linear equation, where the coefficient matrix is exactly the recovery matrix that is defined on the trajectory segment $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$, and candidate feature set $\mathcal{F}$. Here, the costate $\boldsymbol{\lambda}^*_{t+l+1}$ can be interpreted as a variable encoding the *unseen future information* beyond the observational interval $[t, t+l]$. In fact, from the discussions for Case III, we note that costate $\boldsymbol{\lambda}^*_{t+l+1}$ is the gradient of the optimal cost-to-go function with respect to the state evaluated at $\boldsymbol{x}^*_{t+l+1}$.

In IOC problems, in order to obtain an estimate of the unknown weights $\bar{\boldsymbol{\omega}}$ only using the available segment data $\boldsymbol{\xi}_{t:t+l}$, one also needs to account for the unknown $\boldsymbol{\lambda}^*_{t+l+1}$, as in (2.23). The following lemma establishes a relationship between a trajectory segment $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$ and a successful estimate of the weights $\bar{\boldsymbol{\omega}}$ for given candidate features $\mathcal{F}$.

Lemma 2.2.1. *Given a trajectory segment $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$, let the recovery matrix $\boldsymbol{H}(t,l)$ be defined as in Definition 2.2.1 with the candidate feature set $\mathcal{F}$ satisfying Assumption 2.2.1. Let a vector* $\operatorname{col}\{\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{\lambda}}\} \neq \mathbf{0}$ *satisfy* $\operatorname{col}\{\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{\lambda}}\} \in \ker \boldsymbol{H}(t,l)$ *with $\hat{\boldsymbol{\omega}} \in \mathbb{R}^r$. If*

$$\operatorname{rank} \boldsymbol{H}(t,l) = r + n - 1, \quad (2.24)$$

then there exists a constant $c \neq 0$ *such that the* i*th entry of* $\hat{\boldsymbol{\omega}}$ *satisfies*

$$\hat{\omega}_i = \begin{cases} c\omega_i, & \text{if} \quad \phi_i \in \mathcal{F}^* \\ 0, & \text{otherwise} \end{cases}, \tag{2.25}$$

and vector col $\{\hat{\omega}_i : \phi_i \in \mathcal{F}^*, i = 1, 2, \cdots, r\} = c\boldsymbol{\omega}$ *thus is a successful estimate of* $\boldsymbol{\omega}$ *in (2.3).*

Proof. Based on the equations in (2.17), we note that for a trajectory segment $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$, there always exists $\boldsymbol{\lambda}^*_{t+l+1} \in \mathbb{R}^n$ such that col $\{\bar{\boldsymbol{\omega}}, \boldsymbol{\lambda}^*_{t+l+1}\}$ satisfies (2.23), i.e., col $\{\bar{\boldsymbol{\omega}}, \boldsymbol{\lambda}^*_{t+l+1}\} \in \ker \boldsymbol{H}(t, l)$. Due to (2.24) which means that the kernel of $\boldsymbol{H}(t, l)$ is one-dimensional, any nonzero vector col $\{\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{\lambda}}\} \in \ker \boldsymbol{H}(t, l)$ will have $\hat{\boldsymbol{\omega}} = c\bar{\boldsymbol{\omega}}$ $(c \neq 0)$. Thus, one can conclude that $\hat{\boldsymbol{\omega}}$ is a scaled version of $\bar{\boldsymbol{\omega}}$, and that the entries in $\hat{\boldsymbol{\omega}}$ corresponding to the relevant features in $\mathcal{F}^*$ will stack a successful estimate of $\boldsymbol{\omega}$ (2.3). This completes the proof. □

Remark. *Lemma 2.2.1 states that the recovery matrix bridges trajectory segment data to the unknown objective function parameter. First, the rank of the recovery matrix* $\boldsymbol{H}(t, l)$ *indicates whether one is able to use the trajectory segment* $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$ *to obtain a successful estimate of weights* $\bar{\boldsymbol{\omega}}$ *for the given candidate features* $\mathcal{F}$*. In particular, if the rank condition (2.24) for the recovery matrix* $\boldsymbol{H}(t, l)$ *is satisfied, then any nonzero vector* col $\{\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{\lambda}}\}$ *in the kernel of* $\boldsymbol{H}(t, l)$ *has that: the vector of the first* r *entries in* col $\{\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{\lambda}}\}$*, i.e.,* $\hat{\boldsymbol{\omega}}$*, satisfies* $\hat{\boldsymbol{\omega}} = c\bar{\boldsymbol{\omega}}$*. Second, including additional irrelevant features in* $\mathcal{F}$ *will not influence the weight estimate for the relevant features, since the weight estimates in* $\hat{\boldsymbol{\omega}}$ *for these irrelevant features will be zeros.*

2.2.3 Properties of Recovery Matrix

Since the recovery matrix connects trajectory segment data to the unknown cost function parameter, we next investigate the properties of the recovery matrix, which will provide us a better understanding of how the data and the selected features are incorporated in IOC process. We first present an iterative formula for the recovery matrix.

Lemma 2.2.2 (Iterative Property). *For a trajectory segment $\boldsymbol{\xi}_{t:t+l} \subset \boldsymbol{\xi}$ and the subsequent data point $\boldsymbol{\xi}_{t+l+1} = \{\boldsymbol{x}^*_{t+l+1}, \boldsymbol{u}^*_{t+l+1}\}$, one has*

$$\begin{aligned}\boldsymbol{H}(t, l+1) &= \begin{bmatrix}\boldsymbol{H}_1(t, l+1) & \boldsymbol{H}_2(t, l+1)\end{bmatrix} \\ &= \begin{bmatrix}\boldsymbol{H}_1(t, l) & \boldsymbol{H}_2(t, l) \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}}\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{0} \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+l+1}} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+l+1}}\end{bmatrix},\end{aligned} \tag{2.26}$$

with $\boldsymbol{H}(t, 1)$ corresponding to $\boldsymbol{\xi}_{t:t+1} = (\boldsymbol{x}^*_{t:t+1}, \boldsymbol{u}^*_{t:t+1})$:

$$\boldsymbol{H}(t, 1) = \begin{bmatrix}\boldsymbol{H}_1(t, 1) & \boldsymbol{H}_2(t, 1)\end{bmatrix} = \begin{bmatrix}(\frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_t}\frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+1}} + \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_t}) & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_t}\frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+1}}\end{bmatrix}. \tag{2.27}$$

Proof. Consider the recovery matrix $\boldsymbol{H}(t, l)$ for the trajectory segment $\boldsymbol{\xi}_{t:t+l} = (\boldsymbol{x}^*_{t:t+l}, \boldsymbol{u}^*_{t:t+l})$. When a subsequent point $\boldsymbol{\xi}_{t+l+1} = (\boldsymbol{x}^*_{t:t+l+1}, \boldsymbol{u}^*_{t:t+l+1})$ is observed, from Definition 2.2.1, the updated recovery matrix is $\boldsymbol{H}(t, l+1) = [\boldsymbol{H}_1(t, l+1), \boldsymbol{H}_2(t, l+1)]$, where

$$\boldsymbol{H}_1(t, l{+}1) = \boldsymbol{F}_u(t, l{+}1)\boldsymbol{F}_x^{-1}(t, l{+}1)\boldsymbol{\Phi}_x(t, l{+}1) + \boldsymbol{\Phi}_u(t, l{+}1), \tag{2.28}$$

and

$$\boldsymbol{H}_2(t, l{+}1) = \boldsymbol{F}_u(t, l{+}1)\boldsymbol{F}_x^{-1}(t, l{+}1)\boldsymbol{V}(t, l{+}1). \tag{2.29}$$

Here, $\boldsymbol{F}_x(t, l+1)$, $\boldsymbol{F}_u(t, l+1)$, $\boldsymbol{\Phi}_x(t, l+1)$, $\boldsymbol{\Phi}_u(t, l+1)$, and $\boldsymbol{V}(t, l+1)$, defined in (2.9), are updated as follow:

$$\boldsymbol{\Phi}_u(t, l{+}1){=}\begin{bmatrix}\boldsymbol{\Phi}_u(t, l) \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}}\end{bmatrix},\ \boldsymbol{\Phi}_x(t, l{+}1){=}\begin{bmatrix}\boldsymbol{\Phi}_x(t, l) \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+l+1}}\end{bmatrix},\ \boldsymbol{F}_u(t, l{+}1){=}\begin{bmatrix}\boldsymbol{F}_u(t, l) & \boldsymbol{0} \\ \boldsymbol{0} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}}\end{bmatrix}, \tag{2.30a}$$

$$\boldsymbol{F}_x^{-1}(t, l{+}1) = \begin{bmatrix}\boldsymbol{F}_x(t, l) & -\boldsymbol{V}(t, l) \\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}^{-1} = \begin{bmatrix}\boldsymbol{F}_x^{-1}(t, l) & \boldsymbol{F}_x^{-1}(t, l)\boldsymbol{V}(t, l) \\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}, \tag{2.30b}$$

respectively. Here (2.30b) is based on the fact

$$\begin{bmatrix}A & B \\ C & D\end{bmatrix}^{-1} = \begin{bmatrix}A^{-1} + A^{-1}BK^{-1}CA^{-1} & -A^{-1}BK^{-1} \\ -K^{-1}CA^{-1} & K^{-1}\end{bmatrix}$$

with $K = D - CA^{-1}B$ being the Schur complement of the above block matrix with respect to A. Combining (2.30), we have

$$\begin{aligned}\boldsymbol{H}_1(t,l+1) &= \boldsymbol{F}_u(t,l+1)\boldsymbol{F}_x^{-1}(t,l+1)\boldsymbol{\Phi}_x(t,l+1) + \boldsymbol{\Phi}_u(t,l+1) \\ &= \begin{bmatrix} \boldsymbol{F}_u(t,l)\boldsymbol{F}_x^{-1}(t,l)\boldsymbol{\Phi}_x(t,l) + \boldsymbol{\Phi}_u(t,l) + \boldsymbol{F}_u(t,l)\boldsymbol{F}_x^{-1}(t,l)\boldsymbol{V}(t,l)\frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+l+1}} \\ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}}\frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+l+1}} + \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} \end{bmatrix}. \end{aligned} \tag{2.31}$$

According to (2.7) and (2.8), the above (2.31) becomes

$$\boldsymbol{H}_1(t,l+1) = \begin{bmatrix} \boldsymbol{H}_1(t,l) + \boldsymbol{H}_2(t,l)\frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+l+1}} \\ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}}\frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+l+1}} + \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} \end{bmatrix}. \tag{2.32}$$

According to (2.8), we have

$$\begin{aligned}\boldsymbol{H}_2(t,l+1) &= \boldsymbol{F}_u(t,l+1)\boldsymbol{F}_x^{-1}(t,l+1)\boldsymbol{V}(t,l+1) \\ &= \begin{bmatrix} \boldsymbol{F}_u(t,l)\boldsymbol{F}_x^{-1}(t,l)\boldsymbol{V}(t,l)\frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+l+1}} \\ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}}\frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+l+1}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{H}_2(t,l)\frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+l+1}} \\ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}}\frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+l+1}} \end{bmatrix}. \end{aligned} \tag{2.33}$$

Finally joining (2.32) and (2.33) and writing them in the matrix form lead to (2.26).

When $l = 1$, that is, $\boldsymbol{\xi}_{t:t+1} = (\boldsymbol{x}^*_{t:t+1}, \boldsymbol{u}^*_{t:t+1})$ is available, we have $\boldsymbol{F}_x(t,1) = I$, $\boldsymbol{F}_u(t,1) = \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_t}$, $\boldsymbol{\Phi}_x(t,1) = \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+1}}$, $\boldsymbol{\Phi}_u(t,1) = \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_t}$, and $\boldsymbol{V}(t,1) = \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+1}}$. According to the definition of recovery matrix in (2.7) and (2.8), we thus obtain (2.27). This completes the proof. □

The iterative property shows that the recovery matrix can be calculated by incrementally integrating each subsequent data point $\boldsymbol{\xi}_{t+1+1}$ into the current recovery matrix $\boldsymbol{H}(t,l)$. Due to this property, the computation of matrix inversions in the recovery matrix in Definition 2.2.1 can be avoided.

The recovery matrix is defined on two components: one is the segment data $\boldsymbol{\xi}_{t:t+l}$ and the other are the selected candidate features $\mathcal{F}$. In what follows, we will show how these two components affect the recovery matrix and further the IOC process. For data observations,

we expect that including more data points into $\boldsymbol{\xi}_{t:t+l}$ may contribute to enabling the successful estimation of the unknown weights. This is implied by the following lemma.

Lemma 2.2.3 (Rank Nondecreasing Property)**.** *For a trajectory segment $\boldsymbol{\xi}_{t:t+l} \subset \boldsymbol{\xi}$ and any $\mathcal{F}$, one has*

$$\operatorname{rank} \boldsymbol{H}(t, l) \leq \operatorname{rank} \boldsymbol{H}(t, l+1), \tag{2.34}$$

if the new trajectory point $\boldsymbol{\xi}_{t+l+1} = (\boldsymbol{x}^*_{t+l+1}, \boldsymbol{u}^*_{t+l+1})$ has $\det(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}^*_{t+l+1}}) \neq 0$.

Proof. From Lemma 2.2.2, we have

$$\operatorname{rank} \boldsymbol{H}(t, l+1) = \operatorname{rank} \begin{bmatrix} \boldsymbol{H}_1(t,l) & \boldsymbol{H}_2(t,l) \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+l+1}} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+l+1}} \end{bmatrix}. \tag{2.35}$$

If $\det(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}^*_{t+l+1}}) \neq 0$, the last block matrix in (2.35) is non-singular. Consequently

$$\begin{aligned} \operatorname{rank} \boldsymbol{H}(t, l+1) &= \operatorname{rank} \begin{bmatrix} \boldsymbol{H}_1(t,l) & \boldsymbol{H}_2(t,l) \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}} \end{bmatrix} \\ &\geq \operatorname{rank} \begin{bmatrix} \boldsymbol{H}_1(t,l) & \boldsymbol{H}_2(t,l) \end{bmatrix} = \operatorname{rank} \boldsymbol{H}(t, l). \end{aligned} \tag{2.36}$$

Note that both (2.35) and the inequality (2.36) are independent of the choice of $\boldsymbol{\phi}$. This completes the proof. □

We have noted in Lemma 2.2.1 that the rank of the recovery matrix is related to whether one is able to use segment data $\boldsymbol{\xi}_{t:t+l}$ to achieve a successful estimate of the weights. Thus the rank of the recovery matrix can be viewed as an indicator of the capability of the available segment data $\boldsymbol{\xi}_{t:t+l}$ to reflect the unknown weights. Lemma 2.2.3 postulates that additional data, if its Jacobian matrix of the dynamics is non-singular, tends to contribute to solving the IOC problem by increasing the rank of the recovery matrix towards satisfying (2.24), or at least will not make a degrading contribution. In the later experiments in Section 2.3.3, we will analytically and experimentally demonstrate in which cases the additional observation data can increase the rank of the recovery matrix, and in which cases the additional data points cannot increase (i.e. maintain the recovery matrix rank).

The next lemma provides a necessary condition for the rank of the recovery matrix if the candidate feature set $\mathcal{F}$ contains as a subset the relevant features $\mathcal{F}^*$, i.e., $\mathcal{F}^* \subseteq \mathcal{F}$.

Lemma 2.2.4 (Rank Upper Bound Property)**.** *If Assumption 2.2.1 holds, then for any trajectory segment $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$,*

$$\operatorname{rank} \boldsymbol{H}(t,l) \leq r + n - 1 \tag{2.37}$$

always holds. If there exists another relevant feature subset $\widetilde{\mathcal{F}} \subseteq \mathcal{F}$ with corresponding weights $\tilde{\boldsymbol{\omega}}$, here $\widetilde{\mathcal{F}} \neq \mathcal{F}^*$ or $\tilde{\boldsymbol{\omega}} \neq c\boldsymbol{\omega}$, then the above inequality (2.37) holds strictly:

$$\operatorname{rank} \boldsymbol{H}(t,l) < r + n - 1. \tag{2.38}$$

Proof. We first prove (2.37). Without losing generality, we consider the feature set in (2.10). For any trajectory segment $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$, from (2.23), we have known that there exists a costate $\boldsymbol{\lambda}^*_{t+l+1}$ such that

$$\boldsymbol{H}(t,l) \begin{bmatrix} \bar{\boldsymbol{\omega}} \\ \boldsymbol{\lambda}^*_{t+l+1} \end{bmatrix} = \mathbf{0} \tag{2.39}$$

holds, where $\bar{\boldsymbol{\omega}} \neq \mathbf{0}$ is defined in (2.12). Thus, the nullity of $\boldsymbol{H}(t,l)$ is at least one, which means

$$\operatorname{rank} \boldsymbol{H}(t,l) \leq r + n - 1.$$

We then prove (2.38). When another relevant feature subset $\widetilde{\mathcal{F}}$ exists in $\mathcal{F}$ with associated weight vector $\tilde{\boldsymbol{\omega}}$, we can similarly construct a weight vector $\breve{\boldsymbol{\omega}}$ corresponding to col $\mathcal{F}$ as in (2.12); that is, the weights in $\breve{\boldsymbol{\omega}}$ that correspond to $\widetilde{\mathcal{F}}$ are from $\tilde{\boldsymbol{\omega}}$ and zeros otherwise. Then following the similar derivations as from (2.13) to (2.23), we can obtain that there exists $\breve{\boldsymbol{\lambda}}_{t+l+1} \in \mathbb{R}^n$ such that

$$\boldsymbol{H}(t,l) \begin{bmatrix} \breve{\boldsymbol{\omega}} \\ \breve{\boldsymbol{\lambda}}_{t+l+1} \end{bmatrix} = \mathbf{0}. \tag{2.40}$$

Since $\widetilde{\mathcal{F}} \neq \mathcal{F}^*$ or $\widetilde{\boldsymbol{\omega}} \neq c_1\boldsymbol{\omega}$ implies $\breve{\boldsymbol{\omega}} \neq c_2\bar{\boldsymbol{\omega}}$ (c_1 and c_2 are some nonzero scalars), based on (2.39) and (2.40), it follows that the nullity of $\boldsymbol{H}(t,l)$ is at least two, i.e.,

$$\operatorname{rank} \boldsymbol{H}(t,l) \leq r + n - 2.$$

This completes the proof. □

Lemma 2.2.4 states that if a candidate feature set contains as a subset the relevant features under which the agent trajectory $\boldsymbol{\xi}$ is optimal, the kernel of the recovery matrix (for any data segment) is at least one-dimensional. Moreover, when there exist more than one combination of relevant features among the given candidate features, which means there exists another subset of relevant features or another independent weight vector, then the rank condition (2.24) in Lemma 2.2.1 is *impossible* to be fulfilled for the trajectory segment $\boldsymbol{\xi}_{t:t+l}$ regardless of the observation length l and starting time t. This also implies that though Assumption 2.2.1 is likely to be satisfied by using a larger feature set that covers all possible features, it may also lead to the non-uniqueness of relevant features. On the other hand, if Assumption 2.2.1 fails to hold, that is, the candidate feature set $\mathcal{F}$ does not contain a *complete set* of relevant features, then, due to the rank non-decreasing property in Lemma 2.2.3, the recovery matrix is more likely to have rank $\boldsymbol{H}(t,l) = r + n$ after increasing the observation length. To sum up, Lemma 2.2.4 can be leveraged to investigate whether the selection of candidate features is proper or not.

2.2.4 Connection to Prior Work

We next discuss the relationship between the above recovery matrix and existing IOC techniques [34, 40, 42, 43, 45, 47, 48]. In those methods, an *observation of the agent complete trajectory* $\boldsymbol{\xi}$ is considered, for which a set of optimality equations, such as the KKT conditions [41] or Pontryagin's Minimum principle [44], is then established. As developed in [34, 48],

based on optimality conditions, a general form for using complete trajectory data to establish a linear constraint on the unknown feature weights $\boldsymbol{\omega}$ can be summarized as

$$\boldsymbol{M}(\boldsymbol{\xi})\boldsymbol{\omega} = \mathbf{0}, \tag{2.41}$$

where $\boldsymbol{M}(\boldsymbol{\xi})$ is the coefficient matrix that depends on the trajectory data $\boldsymbol{\xi}$. An implicit requirement by those methods is that the *observed data* $\boldsymbol{\xi}$ *itself has to be optimal* with respect to the cost function, thus *complete trajectory data* $\boldsymbol{\xi}_{0:T}$ *is generally required* (otherwise, incomplete data $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$ *itself* in general does not optimize the cost function).

Conversely, through the recovery matrix developed in this chapter, any *trajectory segment* $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$ poses a linear constraint on $\boldsymbol{\omega}$ by

$$\boldsymbol{H}(t,l)\begin{bmatrix}\boldsymbol{\omega}\\ \boldsymbol{\lambda}_{t+l+1}\end{bmatrix} = \boldsymbol{H}_1(t,l)\boldsymbol{\omega} + \boldsymbol{H}_2(t,l)\boldsymbol{\lambda}_{t+l+1} = \mathbf{0}. \tag{2.42}$$

Comparing (2.41) with (2.42), we have the following comments.

1) If we consider the segment $\boldsymbol{\xi}_{t:t+l} = \boldsymbol{\xi}_{0:T}$, i.e., given the complete trajectory $\boldsymbol{\xi}$, then, due to $\boldsymbol{\lambda}_{T+1} = \mathbf{0}$ (assuming the end-free optimal control setting), (2.42) becomes

$$\boldsymbol{H}_1(0,T)\boldsymbol{\omega} = \mathbf{0}. \tag{2.43}$$

Comparing (2.43) with (2.41) we immediately obtain

$$\boldsymbol{H}_1(0,T) = \boldsymbol{M}(\boldsymbol{\xi}). \tag{2.44}$$

Thus, the coefficient matrix $\boldsymbol{M}(\boldsymbol{\xi})$ that is commonly used in existing IOC methods can be considered as a special case of the recovery matrix when the available data is the complete trajectory, i.e., $\boldsymbol{\xi}_{t:t+l} = \boldsymbol{\xi}_{0:T}$.

2) However, as in (2.44), the coefficient matrix $\boldsymbol{M}(\boldsymbol{\xi})$ only corresponds to the first term of the recovery matrix, i.e., $\boldsymbol{H}_1(t,l)$. A key difference of the recovery matrix is its ability to handle any incomplete data $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$. Since the incomplete data $\boldsymbol{\xi}_{t:t+l}$ itself may not be

optimal with respect to the objective function when $t+l < T$, the *unseen future* information thus must be taken care of if one wants to successfully learn the unknown weights $\boldsymbol{\omega}$. As in (2.42), the recovery matrix accounts for such unseen future information via its second term $\boldsymbol{H}_2(t,l)$ and the unknown costate $\boldsymbol{\lambda}_{t+l+1}$.

3) In addition to the capability of dealing with incomplete observation data, the recovery matrix can also provide the insights to the IOC process, as stated in Lemmas 2.2.3 and 2.2.4, and an efficient (iterative) way to compute the coefficient matrix for solving IOC problems, as stated in Lemma 2.2.2. Such properties and computational advantages cannot be achieved by existing IOC methods [34, 40, 42, 43, 45, 47, 48]. (Note that to compute $\boldsymbol{M}(\boldsymbol{\xi})$, existing IOC methods typically require the inverse of a large matrix whose size is proportional to the time horizon T).

2.3 Incremental IOC with Incomplete Trajectory Observations

Based on the previous theories of the recovery matrix, in this section, we will address the first research gap identified in Section 2.1.2. We will present an incremental IOC algorithm that enables to learn an objective function by automatically finding the minimal trajectory observations.

2.3.1 Problem Formulation

Consider an optimal control agent with dynamics (2.1) and that its trajectory (2.2) optimizes an unknown cost function (2.3). Given a relevant feature set $\mathcal{F}^*$ in (2.4), we aim to develop technique to estimate $\boldsymbol{\omega}$ only using an *incomplete trajectory observation*

$$\boldsymbol{\xi}_{t:t+l} = \{\boldsymbol{\xi}_k : t \le k \le t+l\} \subseteq \boldsymbol{\xi}, \tag{2.45}$$

which is a segment of $\boldsymbol{\xi}$ within the time interval $[t, t+l] \subseteq [0, T]$. Here, t is called the *observation starting time* and $l = 1, 2, \cdots$ called the *observation length*, with $0 \le t < t+l \le T$. Moreover, for any observation starting time t, we aim to find the minimal required observation, denoted as $l_{\min}$, to achieve a successful estimate of $\boldsymbol{\omega}$. Note that in the above

problem setting, we *only know* that the data $\boldsymbol{\xi}_{t:t+l}$ is a segment of a system trajectory $\boldsymbol{\xi}$; we do not know the value of t (i.e., the observation starting time relative to the start of the trajectory), and do not require knowledge of any other information about $\boldsymbol{\xi}$ such as the time horizon T or which type of optimal control problem $\boldsymbol{\xi}$ is a solution to, as described in Section 2.2.1.

2.3.2 The Method and Algorithm

The following corollary states a method to use an observation of the incomplete trajectory to achieve a successful estimate of weights for given relevant features.

Corollary 2.3.1 (IOC using Incomplete Trajectory Observations)**.** *For the optimal control agent in (2.1), given an incomplete trajectory observation $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$ in (2.45) and a relevant feature set $\mathcal{F} = \mathcal{F}^*$ in (2.4), the recovery matrix $\boldsymbol{H}(t,l)$ is defined as in Definition 2.2.1. If*

$$\operatorname{rank} \boldsymbol{H}(t,l) = s + n - 1, \tag{2.46}$$

and a nonzero vector $\operatorname{col}\{\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{\lambda}}\} \in \ker \boldsymbol{H}(t,l)$ *with $\hat{\boldsymbol{\omega}} \in \mathbb{R}^s$, then $\hat{\boldsymbol{\omega}}$ is a successful estimate of $\boldsymbol{\omega}$, i.e., there must exist a non-zero constant c such that $\hat{\boldsymbol{\omega}} = c\boldsymbol{\omega}$.*

Proof. Corollary 2.3.1 is a special case of Lemma 2.2.1. □

Remark. *Suppose that in Corollary 2.3.1, (2.46) is not satisfied. According to Lemma 2.2.4,* $\operatorname{rank} \boldsymbol{H}(t,l) < s + n - 1$ *holds and thus dimension of* $\ker \boldsymbol{H}(t,l)$ *is at least two. This means that another weight vector, independent of $\boldsymbol{\omega}$, could be found in* $\ker \boldsymbol{H}(t,l)$*, and the current segment $\boldsymbol{\xi}_{t:t+l}$ may be generated by this different weight vector. In this case, true weights are not distinguishable or recoverable with respect to $\boldsymbol{\xi}_{t:t+l}$. The reason for this case can be insufficient observations or low data informativeness, both of which may be remedied by including additional data (i.e., increase l) according to Lemma 2.2.3 (we will illustrate this later in experiments in Section 2.3.3).*

Combining Lemma 2.2.1 and the properties of the recovery matrix, one has the following conclusions: (i) as observations of more data points may contribute to increasing the rank of

the recovery matrix (Lemma 2.2.3), which is bounded from above (Lemma 2.2.4), thus the minimal required observation length $l_{\min}$ that reaches the rank upper bound can be found; (ii) from Lemmas 2.2.3 and 2.2.4, the minimal required observation length can be found even if additional irrelevant features exist; and (iii) from Lemma 2.2.2, the minimal required observation length can be found efficiently. In sum, we have the following incremental IOC approach.

Corollary 2.3.2 (Incremental IOC Algorithm)**.** *Given candidate features $\mathcal{F}$ satisfying Assumption 2.2.1, the recovery matrix $\boldsymbol{H}(t,l)$, starting from t, is updated at each time step with a new observed point $\boldsymbol{\xi}_{t+l+1} = (\boldsymbol{x}^*_{t+l+1}, \boldsymbol{u}^*_{t+l+1})$ via Lemma 2.2.2. Then the minimal segment length that suffices for a successful estimate of the feature weights is*

$$l_{\min}(t) = \min\Big\{l \,|\, \operatorname{rank} \boldsymbol{H}(t,l) = |\mathcal{F}| + n - 1\Big\}. \tag{2.47}$$

For any nonzero vector $\operatorname{col}\{\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{\lambda}}\} \in \ker \boldsymbol{H}(t, l_{\min}(t))$ *with* $\hat{\boldsymbol{\omega}} \in \mathbb{R}^{|\mathcal{F}|}$, $\hat{\boldsymbol{\omega}}$ *is a successful estimate of the weights for $\mathcal{F}$ with the weights for irrelevant features being zeros.*

Proof. Corollary 2.3.2 is a direct application of Lemma 2.2.1 and Lemmas 2.2.2-2.2.4. □

From Corollary 2.3.2, we note that starting from time t, the minimal required observation length $l_{\min}(t)$ to solve IOC problems is the one satisfying (2.47). As we will show later in experiments in Section 2.3.3, $l_{\min}(t)$ varies depending on the informativeness of data $\boldsymbol{\xi}_{t:t+l}$ and the selected candidate features. Whatever influences $l_{\min}(t)$, one can always find a necessary lower bound of the minimal required observation length due to the size of the recovery matrix, $\boldsymbol{H}(t,l) \in \mathbb{R}^{ml \times (|\mathcal{F}|+n)}$, and matrix rank properties, that is,

$$l_{\min}(t) \geq \left\lceil \frac{|\mathcal{F}| + n - 1}{m} \right\rceil, \tag{2.48}$$

where $\lceil \cdot \rceil$ is the ceiling operation. (2.48) implies that including additional irrelevant features to $\mathcal{F}$ will require more data in order to successfully solve IOC problems (as shown later in experiments in Section 2.3.3).

In practice, directly checking the rank condition of the recovery matrix in (2.47) in Corollary 2.3.2 is challenging due to (i) data noise; (ii) near-optimality of demonstrations,

i.e., the observed trajectory slightly deviates from the optimal one; and (iii) computational error. Thus, one can use the following strategies to evaluate the rank of the recovery matrix. First, we perform a normalization of the recovery matrix before verifying its rank, this is because when the observed data is of low magnitude, the recovery matrix may have the entries rather close to zeros, which may affect the matrix rank evaluation due to computing rounding error.

$$\bar{\boldsymbol{H}}(t,l) = \frac{\boldsymbol{H}(t,l)}{\|\boldsymbol{H}(t,l)\|_F}, \tag{2.49}$$

where $\|\cdot\|_F$ is the Frobenius norm and we only consider the recovery matrix that is not a zero matrix. Then

$$\operatorname{rank} \bar{\boldsymbol{H}}(t,l) = \operatorname{rank} \boldsymbol{H}(t,l). \tag{2.50}$$

Second, since we are only interested in whether the rank of the recovery matrix satisfies $\operatorname{rank} \boldsymbol{H}(t,l) = r+n-1$, instead of directly investigating the rank, we choose to look at the singular values of $\bar{\boldsymbol{H}}(t,l)$ by introducing the following rank index

$$\kappa(t,l) = \begin{cases} 0, & \text{if} \quad \sigma_2(\bar{\boldsymbol{H}}(t,l)) = 0, \\ \sigma_2(\bar{\boldsymbol{H}})/\sigma_1(\bar{\boldsymbol{H}}), & \text{otherwise.} \end{cases} \tag{2.51}$$

The condition $\operatorname{rank} \boldsymbol{H}(t,l) = r+n-1$ is thus equivalent to $\kappa(t,l) = +\infty$. However, due to data noise, $\kappa(t,l) = +\infty$ usually cannot be reached and thus is a finite value (we will demonstrate this in later experiments). We thus pre-set a threshold γ and verify

$$\kappa(t,l) \geq \gamma \tag{2.52}$$

to decide whether $\operatorname{rank} \boldsymbol{H}(t,l) = r+n-1$ is fulfilled or not. Later in experiments in Section 2.3.3), we will show how observation data and noise levels influence the rank index $\kappa(t,l)$, and how to accordingly choose a proper γ.

The computation of a successful estimate in Corollary 2.3.2 can be implemented by solving the following constrained optimization

$$\hat{\boldsymbol{\omega}} = \arg\min_{\boldsymbol{\omega},\boldsymbol{\lambda}} \left\| \bar{\boldsymbol{H}}(t, l_{\min}(t)) \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{\lambda} \end{bmatrix} \right\|^2, \tag{2.53}$$

subject to

$$\sum_{i=1}^{|\mathcal{F}|} \omega_i = 1, \tag{2.54}$$

where $\|\cdot\|$ denotes the l_2 norm and $\bar{\boldsymbol{H}}$ is the normalized recovery matrix (see (2.49)). Here, to avoid trivial solutions, we add the constraint (2.54) to normalize the weight estimate to have sum of one, as used in [34].

In sum, the implementation of the proposed incremental IOC approach in corollary 2.3.2 is presented in Algorithm 1. Algorithm 1 permits arbitrary observation starting time, and the observation length is automatically found by checking the rank condition using (2.51) and (2.52). The algorithm can be viewed as an *adaptive-observation-length* IOC algorithm.

Algorithm 1: Incremental IOC Algorithm

Input: a candidate feature set $\mathcal{F}$, a threshold γ;
Initial: Any observation starting time t;
Initialize l=1, $\boldsymbol{H}(t,l)$ with $\boldsymbol{\xi}_{t:t+1}$=$(\boldsymbol{x}^*_{t:t+1}, \boldsymbol{u}^*_{t:t+1})$ via (2.27);
while $\boldsymbol{H}(t,l)$ *not satisfying (2.52)* **do**
  Obtain subsequent data $\boldsymbol{\xi}_{t+l+1} = (\boldsymbol{x}^*_{t+l+1}, \boldsymbol{u}^*_{t+l+1})$;
  Update $\boldsymbol{H}(t,l)$ with $\boldsymbol{\xi}_{t+l+1}$ via (2.26);
  $l \leftarrow l+1$;
end
minimal required observation length: $l_{\min}(t) = l$;
compute a successful estimate $\hat{\boldsymbol{\omega}}$ via (2.53)-(2.54).

2.3.3 Numerical Experiments

We evaluate the proposed method on two systems. First, on a linear quadratic regulator (LQR) system, we demonstrate the rank properties of the recovery matrix, show its capability of handling incomplete trajectory data by comparing with the related IOC methods, and

demonstrate its capability to solve IOC for infinite-horizon LQR. Second, on a simulated two-link robot arm, we evaluate the proposed techniques in terms of observation noise, including irrelevant features, and parameter settings. Throughout evaluations, we quantify the accuracy of a weight estimate $\hat{\boldsymbol{\omega}}$ by introducing the following estimation error:

$$e_{\boldsymbol{\omega}} = \inf_{c>0} \frac{\|c\hat{\boldsymbol{\omega}} - \boldsymbol{\omega}\|}{\|\boldsymbol{\omega}\|}, \tag{2.55}$$

where $\|\cdot\|$ denotes the l_2 norm, $\hat{\boldsymbol{\omega}}$ is the weight estimate, and $\boldsymbol{\omega}$ is the ground truth. Obviously, $e_{\boldsymbol{\omega}} = 0$ means that $\hat{\boldsymbol{\omega}}$ is a successful estimate of $\boldsymbol{\omega}$.

Evaluations on LQR Systems

Consider a finite-horizon free-end LQR system where the dynamics is

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} 1 \\ 3 \end{bmatrix} \boldsymbol{u}_k, \tag{2.56}$$

with initial $\boldsymbol{x}_0 = [2, -2]'$, and quadratic cost function is

$$J = \sum_{k=0}^{T} \Big(\boldsymbol{x}_k' Q \boldsymbol{x}_k + \boldsymbol{u}_k' R \boldsymbol{u}_k \Big), \tag{2.57}$$

with the time horizon $T = 50$. Here, Q and R are positive definite matrices and assumed to have the structure

$$Q = \begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix}, \quad R = r, \tag{2.58}$$

respectively. In the feature-weight form (2.3), the cost function (2.57) corresponds to the feature vector $\boldsymbol{\phi}^* = [x_1^2, x_2^2, u^2]'$ and weights $\boldsymbol{\omega} = [q_1, q_2, r]'$. We here set $\boldsymbol{\omega} = [0.1, 0.3, 0.6]'$ to generate the optimal trajectory of the LQR system, which is plotted in Fig. 2.1. In IOC problems, we are given the features $\boldsymbol{\phi}^*$; the goal is to solve a successful estimate of $\boldsymbol{\omega}$ using the optimal trajectory data in Fig. 2.1.

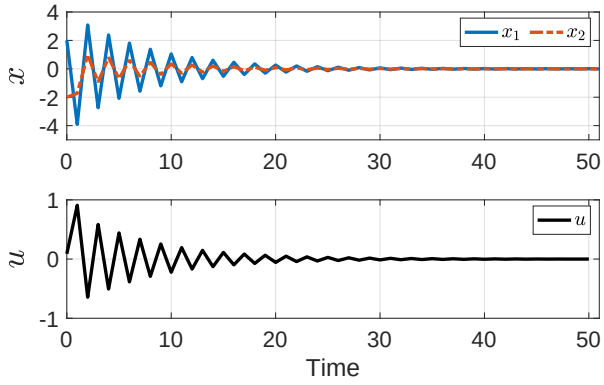


Figure 2.1. The optimal trajectory of a LQR system (2.56)-(2.57) using the weights $\boldsymbol{\omega} = [0.1, 0.3, 0.6]'$.

Minimal Required Observations for IOC. Based on the above LQR system, we here illustrate how the recovery matrix can be used to check whether incomplete trajectory data suffices for the minimal observation required for a successful weight estimation. Given the features $\boldsymbol{\phi}^* = [x_1^2, x_2^2, u^2]'$, we set the observation starting time $t = 0$, and incrementally increase the observation length l from 1 to horizon $T = 50$. For each observation length l, we check the rank of the recovery matrix $\boldsymbol{H}(0, l)$ and solve the weights from the kernel of $\boldsymbol{H}(0, l)$ (the weights are normalized to have sum of one). The results are plotted in Fig. 2.2.

As shown in the upper panel in Fig. 2.2, including additional trajectory data points, i.e., increasing the observation length l (from 1), leads to an increase of the rank of the recovery matrix. When $l = 4$ rank $\boldsymbol{H}(0, l)$ reaches to 4, which is the rank upper bound $n + r - 1 = 4$, and then rank $\boldsymbol{H}(0, l) = 4$ for all $l \geq 4$. This illustrates the properties of the recovery matrix in Lemma 2.2.3 and Lemma 2.2.4. From the bottom panel in Fig. 2.2, we see that when $l < 4$, for which rank $\boldsymbol{H}(0, l) < 4$, the weight estimate $\hat{\boldsymbol{\omega}}$ is not a successful estimate of $\boldsymbol{\omega}$. When rank $\boldsymbol{H}(0, l) < 4$, since the dimension of the kernel of $\boldsymbol{H}(0, l)$ is at least 2 and thus $\boldsymbol{H}(0, l)[\hat{\boldsymbol{\omega}}, \boldsymbol{\lambda}]' = \mathbf{0}$ has multiple solutions $[\hat{\boldsymbol{\omega}}, \boldsymbol{\lambda}]'$, we choose the solution $\hat{\boldsymbol{\omega}}$ from the kernel of $\boldsymbol{H}(0, l)$ randomly. After $l \geq 4$ when rank $\boldsymbol{H}(0, l) = 4$, the estimate converges to a successful estimate, thus indicating the effectiveness of using the rank condition in (2.47) to check whether an incomplete observation suffices for the minimal required observation.

Recovery Matrix Rank for Additional Observations. Based on the LQR system, we next show how additional observations affect the rank of the recovery matrix. Here, we

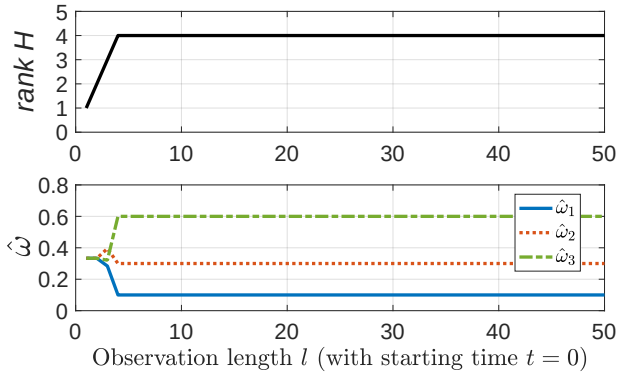
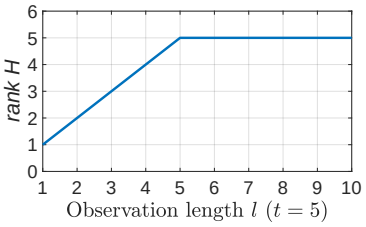


Figure 2.2. The rank of the recovery matrix and weight estimate when the observation starts at $t = 0$ and the observation length l increases from 1 to T. The upper panel shows the rank of the recovery matrix $\boldsymbol{H}(0, l)$ versus l; and the bottom panel shows the corresponding weight estimate for each l. Note that the given features are $\boldsymbol{\phi}^* = [x_1^2, x_2^2, u^2]'$ and the ground truth weights are $\boldsymbol{\omega} = [0.1, 0.3, 0.6]'$. For $l < 4$, since the dimension of the kernel of $\boldsymbol{H}(0, l)$ is at least 2 and thus $\boldsymbol{H}(0, l)[\hat{\boldsymbol{\omega}}, \boldsymbol{\lambda}]' = \mathbf{0}$ has multiple solutions of $[\hat{\boldsymbol{\omega}}, \boldsymbol{\lambda}]'$, we choose the solution $\hat{\boldsymbol{\omega}}$ from the kernel of $\boldsymbol{H}(0, l)$ randomly.

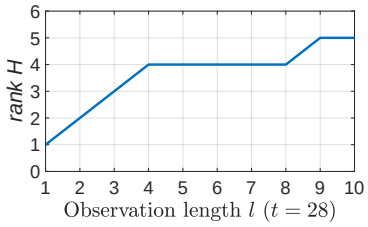
vary the observation starting time t and use different candidate feature sets $\mathcal{F}$, and for each case, we incrementally increase the observation length from $l = 1$ while checking the rank of the recovery matrix until the rank reaches its maximum. The results are presented in Fig. 2.3. For the first three cases in Fig. 2.3a-2.3c, we set the observation starting time at $t = 5$, $t = 28$, and $t = 30$, respectively, and use a candidate feature set $\mathcal{F} = \{x_1^2, x_2^2, u^2, u^3\}$; for the fourth case in Fig. 2.3d, we set the observation starting time at $t = 5$ and use a candidate feature set $\mathcal{F} = \{x_1^2, x_2^2, u^2, 2u^2\}$. Based on the results, we have the following observations and comments.

(1) From Fig. 2.3a, 2.3b, and 2.3c, we can see that additional observation (i.e., increasing observation length l) increases or maintains the rank of the recovery matrix, as stated in Lemma 2.2.3, and that continuously increasing the observation length will lead to the upper bound of the recovery matrix's rank, as stated in Lemma 2.2.4.

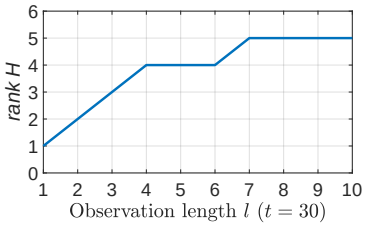
(2) Comparing Fig. 2.3a with Fig. 2.3d, we see that although the number of candidate features for both cases are the same, i.e., $|\mathcal{F}| = r = 4$, their corresponding maximum ranks are different: the case in Fig. 2.3a achieves $\max \operatorname{rank} \boldsymbol{H} = 5 = r + n - 1$ (which is the rank condition (2.24) for a successful estimate), while in Fig. 2.3d the rank reaches $\max \operatorname{rank} \boldsymbol{H} =$



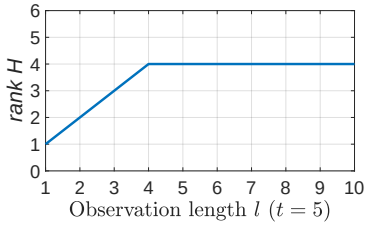
(a) t=5, $\mathcal{F} = \{x_1^2, x_2^2, u^2, u^3\}$



(b) t=28, $\mathcal{F} = \{x_1^2, x_2^2, u^2, u^3\}$



(c) t=30, $\mathcal{F} = \{x_1^2, x_2^2, u^2, u^3\}$



(d) t=5, $\mathcal{F} = \{x_1^2, x_2^2, u^2, 2u^2\}$

Figure 2.3. The rank of the recovery matrix versus the observation length l. For (a), (b), and (c), the observation starting time is at $t = 5$, $t = 28$, and $t = 30$, respectively, and the given candidate feature set is $\mathcal{F} = \{x_1^2, x_2^2, u^2, u^3\}$. For (d), the observation starting time is at $t = 5$ and the given candidate feature set is $\mathcal{F} = \{x_1^2, x_2^2, u^2, 2u^2\}$. In (d), since $\mathcal{F}$ contains two dependent features: u^2 and $2u^2$, thus multiple combinations of these features can be found in $\mathcal{F}$ to characterize the optimal trajectory, that is, $\{x_1^2, x_2^2, u^2\}$ and $\{x_1^2, x_2^2, 2u^2\}$, and the rank upper bound according to Lemma 2.2.4 is rank $\boldsymbol{H}(t, l) < r+n-1 = 5$ and cannot reach 5.

$4 < r + n - 1$. This is because $\mathcal{F} = \{x_1^2, x_2^2, u^2, 2u^2\}$ used in Fig. 2.3d contains two dependent features, i.e., u^2 and $2u^2$, thus multiple combinations of features, e.g., $\{x_1^2, x_2^2, u^2\}$ and $\{x_1^2, x_2^2, 2u^2\}$, can be found in $\mathcal{F}$ to characterize the optimal trajectory. Based on (2.38) in Lemma 2.2.4, rank $\boldsymbol{H}(t, l) \leq 4$ for all t and l and the condition rank $\boldsymbol{H}(t, l) = 5 = r+n-1$ for a successful recovery will never be fulfilled.

(3) Comparing Fig. 2.3b, Fig. 2.3c and Fig. 2.3a, we note that in some cases additional observations will not increase the rank of the recovery matrix, e.g., when the observation

length is $l = 5, 6, 7, 8$ in Fig. 2.3b and $l = 5, 6$ in Fig. 2.3c. This can be explained using the following relations:

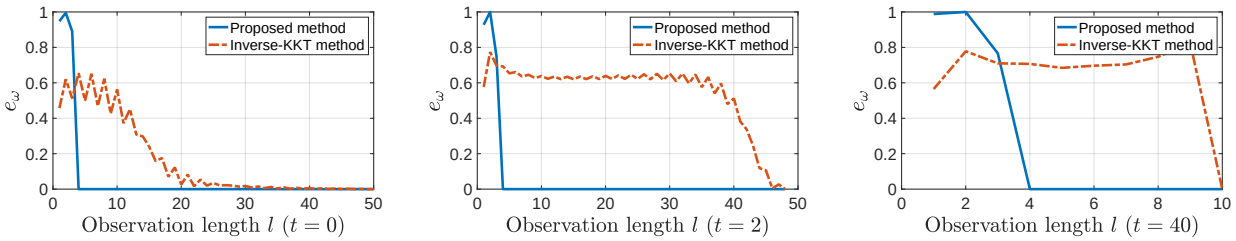
$$\begin{aligned}\operatorname{rank} \boldsymbol{H}(t, l+1) &= \operatorname{rank} \begin{bmatrix} \boldsymbol{H}_1(t,l) & \boldsymbol{H}_2(t,l) \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}} \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}^*_{t+l+1}} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+l+1}} \end{bmatrix} = \operatorname{rank} \begin{bmatrix} \boldsymbol{H}_1(t,l) & \boldsymbol{H}_2(t,l) \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}} \end{bmatrix} \\ &\geq \operatorname{rank} \begin{bmatrix} \boldsymbol{H}_1(t,l) & \boldsymbol{H}_2(t,l) \end{bmatrix} = \operatorname{rank} \boldsymbol{H}(t,l),\end{aligned}$$

for which the first line is directly from (2.26), and the second line is due to $\det(\frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{t+l+1}}) = \det(\begin{smallmatrix} -1 & 0 \\ 1 & 1 \end{smallmatrix}) \neq 0$ and matrix rank properties. The above equation says that the new observation $\boldsymbol{\xi}_{t+l+1} = (\boldsymbol{x}^*_{t+l+1}, \boldsymbol{u}^*_{t+l+1})$ is incorporated into the recovery matrix $\boldsymbol{H}(t, l)$ in the form of appending m row vectors $[\frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} \ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}}] \in \mathbb{R}^{m \times (r+n)}$ to the bottom of $\boldsymbol{H}(t, l)$. If the new observed data point $\boldsymbol{\xi}_{t+l+1} = (\boldsymbol{x}^*_{t+l+1}, \boldsymbol{u}^*_{t+l+1})$ is *non-informative*, in other words, if the appended rows in $[\frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} \ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}}]$ are *dependent* on the row vectors in $\boldsymbol{H}(t, l)$, then according to the matrix rank properties, one will have $\operatorname{rank} \boldsymbol{H}(t, l) = \operatorname{rank} \boldsymbol{H}(t, l+1)$, thus the new data $\boldsymbol{\xi}_{t+l+1}$ will not increase the rank of the recovery matrix. Otherwise, if the appended rows in $[\frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}^*_{t+l}} \ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{t+l}}]$ are *independent* of the row vectors in $\boldsymbol{H}(t, l)$, that is, the new observed data $\boldsymbol{\xi}_{t+l+1} = (\boldsymbol{x}^*_{t+l+1}, \boldsymbol{u}^*_{t+l+1})$ is *informative*, then this new $\boldsymbol{\xi}_{t+l+1}$ will increase the rank of the recovery matrix, i.e., $\operatorname{rank} \boldsymbol{H}(t, l) < \operatorname{rank} \boldsymbol{H}(t, l+1)$.

Comparison with Prior Work. We will show how the recovery matrix is able to solve IOC problems using incomplete observations by comparing with a recent inverse-KKT method developed in [34]. The idea of the inverse-KKT method is based on the optimality equations similar to (2.41) using full trajectory data $\boldsymbol{\xi}$. As suggested by [34], the weights are estimated by minimizing

$$\min_{\boldsymbol{\omega}} \|\boldsymbol{M}(\boldsymbol{\xi})\boldsymbol{\omega}\|^2, \tag{2.59}$$

subject to $\sum_i \omega_i = 1$. Although the inverse-KKT method [34] is developed based on full trajectory data $\boldsymbol{\xi}$, we here want to see its performance when only incomplete data $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$ is given.



(a) Observation starting from $t = 1$

(b) Observation starting from $t = 2$

(c) Observation starting from $t = 40$

Figure 2.4. Comparison between the inverse-KKT method (2.61) and proposed recovery matrix method (2.62) when given incomplete trajectory observation $\boldsymbol{\xi}_{t:t+l}$. Different observation starting time t is used: $t = 0$ in (a), $t = 2$ in (b), and $t = 40$ in (c). For each case, we increase the observation length l from 1 to the end of the horizon, i.e., $t + l = T$, and for each l, the estimation error $e_{\boldsymbol{\omega}}$ for both methods is evaluated, respectively. Note that the estimation error is defined in (2.55).

As analyzed in Section 2.2.4, the coefficient matrix $\boldsymbol{M}(\boldsymbol{\xi})$ is a special case of the recovery matrix when $\boldsymbol{\xi}_{t:t+l} = \boldsymbol{\xi}_{0:T}$, that is,

$$\boldsymbol{M}(\boldsymbol{\xi}) = \boldsymbol{H}_1(0, T). \tag{2.60}$$

Recall that this is because the LQR in (2.56)-(2.57) is a free-end optimal control system, as analyzed in Section 2.2.2, $\boldsymbol{\lambda}_{T+1} = \mathbf{0}$. Given incomplete observation data $\boldsymbol{\xi}_{t:t+l} \subseteq \boldsymbol{\xi}$, comparing the inverse-KKT method

$$\min_{\boldsymbol{\omega}} \|\boldsymbol{M}(\boldsymbol{\xi}_{t:t+l})\boldsymbol{\omega}\|^2 \quad \text{s.t.} \quad \sum_i \omega_i = 1, \tag{2.61}$$

with the proposed recovery matrix method

$$\min_{\boldsymbol{\omega},\boldsymbol{\lambda}} \|\boldsymbol{H}_1(t, l)\boldsymbol{\omega} + \boldsymbol{H}_2(t, l)\boldsymbol{\lambda}\|^2 \quad \text{s.t.} \quad \sum_i \omega_i = 1, \tag{2.62}$$

can show us how the *unseen future data* influences learning of the cost function.

For the LQR trajectory in Fig. 2.1, we use the feature set $\mathcal{F} = \{x_1^2, x_2^2, u^2\}$, set the observation starting time t to be 0, 2, and 40, respectively, and for each observation starting

time t, we increase the observation length l from 1 to the end of the trajectory, i.e., $t + l = T$. With each observation $\boldsymbol{\xi}_{t:t+l}$, we solve the weight estimate using the inverse-KKT method (2.61) and the proposed method (2.62) and evaluate the estimation error $e_{\boldsymbol{\omega}}$ in (2.55), respectively. Results are shown in Fig. 2.4, based on which we have the following comments.

(1) The inverse-KKT method is sensitive to the starting time of the observation sequence. When the observation starts from $t = 0$ (Fig. 2.4a), the inverse-KKT method achieves a successful estimate after observation length $l \geq 30$; when $t = 2$ (Fig. 2.4b) and $t = 40$ (Fig. 2.4c) only when l reaches the end of trajectory, can the inverse-KKT method obtain the successful estimate.

(2) As we have analyzed in Section 2.2.4, the success of the inverse-KKT method requires that the given data $\boldsymbol{\xi}_{t:t+l}$ itself minimizes the cost function, which is only guaranteed when the observation reaches the trajectory end, i.e., $t + l = T$. This explains the results in Fig. 2.4b and 2.4c. Given incomplete $\boldsymbol{\xi}_{t:t+l}$ $(t + l < T)$, although the inverse-KKT method still achieves a successful estimate in Fig. 2.4a, such performance is not guaranteed and heavily relies on 'informativeness' of the given incomplete data relative to *unseen future information*. In Fig. 2.1, since the trajectory data at beginning phase is more 'informative' than the rest, the inverse-KKT method starting from $t = 0$ uses less data to converge (Fig. 2.4a) than starting from $t = 2$ (Fig. 2.4b).

(3) In contrast, Fig. 2.4 shows the effectiveness of using the recovery matrix to deal with incomplete observations. The proposed method guarantees a successful estimate after a much smaller observation length (e.g., around $l = 4$ for all three cases). This advantage is because the unseen future information is accounted for by $\boldsymbol{H}_2(t, l)$ in the recovery matrix and the related unknown future variable $\boldsymbol{\lambda}_{t+l+1}$ is jointly estimated in (2.62).

In sum, we make the following conclusions. First, existing KKT-based methods generally require a full trajectory, and cannot deal with incomplete trajectory data. Second, the proposed recovery matrix method addresses this by jointly accounting for *unseen* future information; and the recovery matrix presents a systematic way to check whether a trajectory segment is sufficient to recover the objective function and if so, to solve it only using the segment data. Third, existing KKT-based methods can be viewed as a special case of the proposed recovery matrix method when the segment data is the full trajectory.

IOC for Infinite-horizon LQR. We demonstrate the ability of the proposed method to solve the IOC problem for an infinite-horizon control system. We still use the LQR system in (2.56)-(2.57) as an example, but here we set the time horizon $T = \infty$ (other conditions and parameters remain the same). The optimal trajectory in this case is a result of feedback control $\boldsymbol{u} = K\boldsymbol{x}$ with a constant control gain K solved by the algebraic Riccati equation [2]. For the above infinite-horizon LQR (with Q and R in (2.58)), the control gain is solved as $K = [-0.1472 \;\; -0.1918]$.

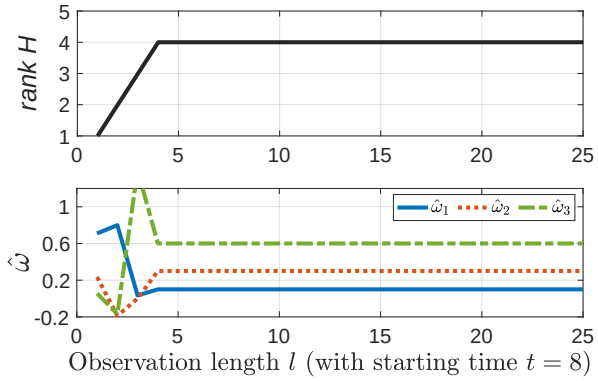


Figure 2.5. IOC results for infinite-horizon LQR system. The observation starting time is $t = 8$ and the observation length l increases from $l = 1$ to 25. The upper panel shows the rank of the recovery matrix versus increasing l, and the bottom panel is the corresponding weight estimate for each l.

In IOC, suppose that we observe an arbitrary segment from the infinite-horizon trajectory; here we use the segment data within the time interval $[t, t + l] = [8, 33]$, namely, $\boldsymbol{\xi}_{8:33}$ with $t = 8$ and $l = 25$. We set the candidate feature set $\mathcal{F} = \{x_1^2, x_2^2, u^2\}$. The IOC results using Algorithm 1 are presented in Fig. 2.5. Here we fix the observation starting time $t = 8$ while increasing l from 1 to the time interval end 25. The upper panel of Fig. 2.5 shows rank $\boldsymbol{H}(8, l)$ versus increasing observation length l, and the bottom panel shows the weight estimate $\hat{\boldsymbol{\omega}}$ of each l solved from the kernel of the recovery matrix $\boldsymbol{H}(8, l)$. As shown in the upper panel, with the observation length l increasing, rank $\boldsymbol{H}(8, l)$ quickly reaches the upper bound rank $r + n - 1 = 4$ after $l \geq 4$, indicting the successful estimate of the weights as shown in the bottom panel. The results demonstrate the ability of the proposed method to solve IOC problems for infinite-horizon optimal control systems.

Evaluation on a Two-link Robot Arm

To evaluate the proposed method on a non-linear agent, we use a two-link robot arm system which is given in Appendix A.2. Based on the robot dynamics in (A.2), we have

$$\ddot{\boldsymbol{\theta}} = M(\boldsymbol{\theta})^{-1}(-C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\theta} - \boldsymbol{g}(\boldsymbol{\theta}) + \boldsymbol{\tau}), \tag{2.63}$$

which can be further expressed in state-space representation

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}), \tag{2.64}$$

with the system state and input defined as

$$\boldsymbol{x} = \begin{bmatrix} \theta_1 & \theta_2 & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}', \qquad \boldsymbol{u} = \begin{bmatrix} \tau_1 & \tau_2 \end{bmatrix}', \tag{2.65}$$

respectively. We consider the following finite-horizon fixed-end optimal control for the above robot arm system:

$$\begin{aligned} \min_{\boldsymbol{x}_{1:T}} \quad & \sum_{k=0}^{T} \boldsymbol{\omega}\boldsymbol{\phi}^*(\boldsymbol{x}_k, \boldsymbol{u}_k), \\ \text{s.t.} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k), \\ & \boldsymbol{x}_0 = \boldsymbol{x}_{\text{start}}, \\ & \boldsymbol{x}_{T+1} = \boldsymbol{x}_{\text{goal}}, \end{aligned} \tag{2.66}$$

where $\Delta = 0.01$s is the discretization interval. In (2.66), we specify the initial state $\boldsymbol{x}_{\text{start}} = [0, 0, 0, 0]'$, goal state $\boldsymbol{x}_{\text{goal}} = [\frac{\pi}{2}, -\frac{\pi}{2}, 0, 0]'$, the time horizon $T = 100$, and the feature vector and the corresponding weights

$$\boldsymbol{\phi}^* = \begin{bmatrix} \tau_1^2 & \tau_2^2 & \tau_1\tau_2 \end{bmatrix}' \quad \boldsymbol{\omega} = \begin{bmatrix} 0.6 & 0.3 & 0.1 \end{bmatrix}', \tag{2.67}$$

respectively. We solve the above optimal control system (2.66) using the CasADi software [50] and plot the resulting trajectory in Fig. 2.6.

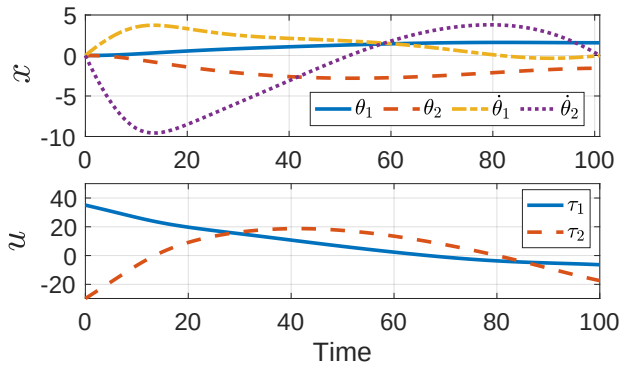


Figure 2.6. The optimal trajectory of the two-link robot arm optimal control system (2.66) with the cost function (2.67).

Observation Noise. We test the proposed incremental IOC approach (Algorithm 1) under different data noise levels. We add to the trajectory in Fig. 2.6 Gaussian noise of different levels that are characterized by different standard deviations from $\sigma = 10^{-5}$ to $\sigma = 10^{-1}$. In Algorithm 1, we use $\mathcal{F} = \{\tau_1^2, \tau_2^2, \tau_1\tau_2\}$ and set $\gamma = 45$ (the choice of γ will be discussed later in Section 2.3.3). We set the observation starting time t at all time instants except for those near the trajectory end which can not provide sufficient subsequent observation length. As an example, we present the experimental results for the case of noise level $\sigma = 10^{-2}$ in Fig 2.7. Here, the upper panel shows the minimal required observation length $l_{\min}(t)$ automatically found for each observation starting time t, and the bottom shows the corresponding weight estimate using the minimal required observation data $\boldsymbol{\xi}_{t:t+l_{\min}(t)}$.

From Fig. 2.7, we see that the automatically-found minimal required observation length $l_{\min}(t)$ varies depending on the observation starting time t. This can be interpreted by noting that the trajectory data in Fig. 2.6 in different intervals has different informativeness to reflect the cost function. For example, according to Fig. 2.7, we can postulate that the beginning and final portions of the trajectory data are more 'data-informative' than other portions, thus needing smaller $l_{\min}(t)$ to achieve the successful estimate. This can be understood if we consider that the beginning and final portions of the trajectory in Fig. 2.6 has richer patterns such as curvatures than the middle which are more smooth. Using the recovery matrix, the data informativeness about the cost function is quantitatively indicated by the recovery matrix's rank. Even under observation noise, the proposed method can

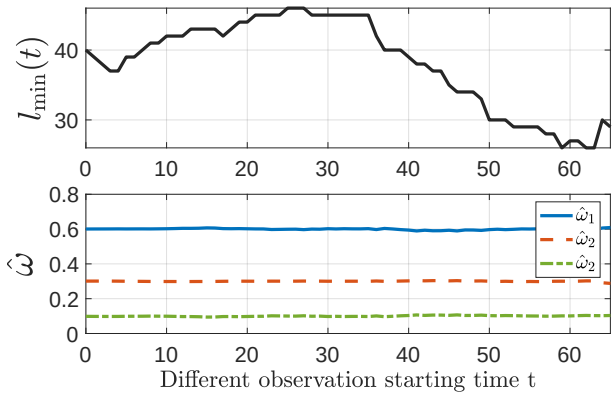


Figure 2.7. IOC by automatically finding the minimal required observation under noise level $\sigma = 10^{-2}$. The x-axis is the different observation starting time t. The upper panel shows the automatically-found minimal required observation length $l_{\min}(t)$ at different t, and the bottom panel shows the corresponding estimate $\hat{\boldsymbol{\omega}}$ via (2.53). Note that ground truth $\boldsymbol{\omega} = [0.6, 0.3, 0.1]'$.

adaptively find the sufficient observation length size such that the data is informative enough to guarantee a successful estimate of the weights, as shown by both upper and bottom panels in Fig. 2.7.

Table 2.1. Results of incremental IOC (Algorithm 1, $\gamma = 45$) under different noise levels.

Noise level σ	Averaged $l_{\min}/T$ (%) †	Averaged $e_{\boldsymbol{\omega}}$†
$\sigma = 10^{-5}$	8%	4.3×10^{-4}
$\sigma = 10^{-4}$	8.1%	4.0×10^{-3}
$\sigma = 10^{-3}$	12.61%	8.1×10^{-3}
$\sigma = 10^{-2}$	33.8%	8.5×10^{-3}
$\sigma = 10^{-1}$	70.0%	7.1×10^{-3}

† The average is calculated based on all successful estimations over all observation cases (varying observation starting time).

We summarize all results under different noise levels in Table 2.1. Here the minimal required observation length is presented in percentage with respect to the total horizon T. According to Table 2.1, under a fixed rank index threshold (here $\gamma = 45$), we can see that high noise levels, on average, will lead to larger minimal required observation length, but the estimation error is not influenced too much. This is because the increased observation length

can compensate for the uncertainty induced by data noise and finally produces a 'neutralized' estimate. Hence, the results prove the robustness of the proposed incremental IOC algorithm against the small observation noise. We will later show how to further improve the accuracy by adjusting γ.

Presence of Irrelevant Features. We here assume that exact knowledge of relevant features is not available, and we evaluate the performance of Algorithm 1 given a feature set including irrelevant features. We add all observation data with Gaussian noise of $\sigma = 10^{-3}$. In Algorithm 1, we set $\gamma = 45$ and construct a feature set $\mathcal{F}$ based on the following candidate features

$$\{\tau_1^2,\ \tau_2^2,\ \tau_1\tau_2,\ \tau_1^3,\ \tau_2^3,\ \tau_1\tau_2^2,\ \tau_1^2\tau_2,\ \tau_1^4, \tau_2^4,\ \tau_1^3\tau_2,\ \tau_1\tau_2^3,\ \tau_1^2\tau_2^2\}. \tag{2.68}$$

Algorithm 1 is applied the same way as in the previous experiment: by starting the observation at all time instants except for those near the trajectory end. We provide different candidate feature sets in the first column in Table 2.2, and for each case we compute the average of the minimal required observation length and the average of estimation error in (2.55). The results are summarized in second and third columns in Table 2.2.

Table 2.2. Results of incremental IOC (Algorithm 1, $\gamma = 45$) with different given feature sets.

Candidate feature set $\mathcal{F}$	Averaged $l_{\min}/T$[1]	Averaged $e_{\boldsymbol{\omega}}$[1]
$\{\tau_1^2,\ \tau_2^2,\ \tau_1\tau_2\}$	12.18%	4.2×10^{-3}
$\{\tau_1^2,\ \tau_2^2,\ \tau_1\tau_2,\ \tau_1^3, \tau_2^3\}$	14.7%	9.7×10^{-3}
$\{\tau_1^2,\ \tau_2^2,\ \tau_1\tau_2,\ \tau_1^3, \tau_2^3,\ \tau_1\tau_2^2,\ \tau_1^2\tau_2\}$	25.69%	8.7×10^{-3}
$\{\tau_1^2,\ \tau_2^2,\ \tau_1\tau_2,\ \tau_1^3, \tau_2^3,\ \tau_1\tau_2^2,\ \tau_1^2\tau_2, \tau_1^4, \tau_2^4\}$	35.97%	8.6×10^{-3}
$\{\tau_1^2,\ \tau_2^2,\ \tau_1\tau_2,\ \tau_1^3, \tau_2^3,\ \tau_1\tau_2^2,\ \tau_1^2\tau_2, \tau_1^4, \tau_2^4,\ \tau_1^3\tau_2,\ \tau_1\tau_2^3,\ \tau_1^2\tau_2^2\}$	45.53%	9.1×10^{-3}

[1] The average is calculated based on all successful estimations over all observation cases (varying observation starting time).

Table 2.2 indicates that on average, the minimal required observation length increases as additional irrelevant features are included to the feature set $\mathcal{F}$. This can be understood if we consider (2.47) and the rank non-decreasing property in Lemma 2.2.3: when a certain number of irrelevant features are added, the rank required for successful estimate will increase by the

same amount, thus needing additional trajectory data points. Due to increased observation length, the estimation accuracy is not much influenced by the additional irrelevant features. Thus we conclude that the proposed incremental IOC algorithm applies to the presence of irrelevant features.

Parameter Setting. We now discuss how to choose the rank threshold γ in Algorithm 1. Since in Algorithm 1 the rank index (2.51) for the recovery matrix is used to find the minimal required observation length, we first investigate how the rank index $\kappa(t, l)$ changes as the observation length l increases. We use the trajectory data in Fig. 2.6 with added Gaussian noise of $\sigma = 10^{-3}$, $\sigma = 2 \times 10^{-3}$, and $\sigma = 10^{-2}$, respectively. The candidate features set here is $\mathcal{F} = \{\tau_1^2, \tau_2^2, \tau_1\tau_2\}$. We fix the observation start time $t = 0$ and increase the observation length l from 1 to T. The rank index $\kappa(t = 0, l)$ for different l is shown in Fig. 2.8.

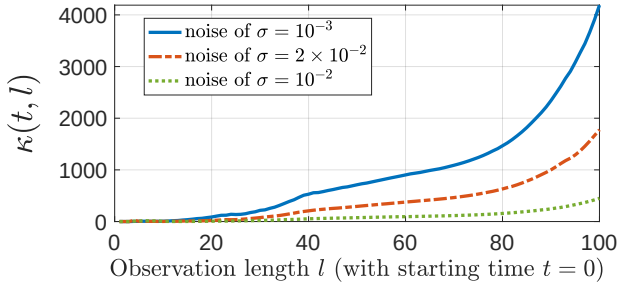


Figure 2.8. The rank index $\kappa(0, l)$ in (2.51) versus different observation length l under different noise levels.

From Fig. 2.8, we can see that although $\kappa(t, l)$ has different scales at different noise levels, it in general increases as the observation length l increases. This can be understood if we compare the above results to $\kappa(t, l)$ in noise-free cases: when there is no data noise, according to Lemma 2.2.3 and 2.2.4, as l increases, $\kappa(t, l)$ will first remain zero when $l < l_{\min}$, then increase to infinity after $l \geq l_{\min}$. In noisy settings, $\kappa(t, l)$ however will increase to a large finite value. From the plot, we can postulate that in practice choosing a larger threshold γ will lead to a larger minimal required observation length $l_{\min}$, thus more data points will be included into the recovery matrix to compute the estimate of the weights, which may finally

improve the estimation accuracy (similar to results in Table 2.1). In what follows, we will verify this postulation by showing how γ affects the performance of Algorithm 1.

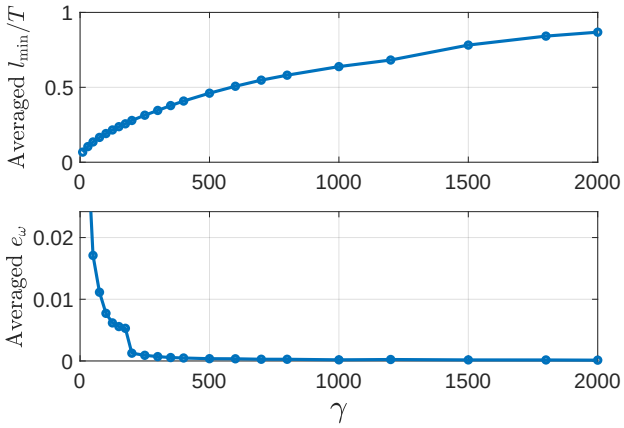


Figure 2.9. Averaged $l_{\min}$ (upper panel) and averaged estimation error $e_{\boldsymbol{\omega}}$ (bottom panel) for different choices of γ.

We add Gaussian noise $\sigma = 10^{-3}$ to the trajectory data in Fig. 2.6, and apply Algorithm 1 by starting the observation at all possible time steps, as performed in previous experiments. We vary γ to show its influence on the average of the minimal required observation length $l_{\min}$ and the average of the estimation error $e_{\boldsymbol{\omega}}$. The results are shown in Fig. 2.9, from which we can observe that first, a larger γ will lead to larger minimal required observation length; and second, due to the increased minimal required observation, the corresponding estimation accuracy is improved because data noise or other error sources can be compensated by additional observation data. These facts thus prove our previous postulation based on Fig. 2.8. Moreover, Fig. 2.9 also shows as γ exceeds a certain value, e.g., 200, continuously increasing γ will not improve the recovery accuracy significantly. This suggests that the choice of γ is not sensitive to the performance if γ is large. Therefore, in practice it is possible to find a proper γ without much manual effort such that both the estimation accuracy and computational cost are balanced.

2.4 IOC for Multi-phase Objective Functions

In this section, we will investigate the IOC problem for *multi-phase* cost functions, aiming to address the second research gap identified in Section 2.1.2. We consider the observed trajectory as a concatenation of multiple phases of motion, where each phase is characterized by a distinct cost function parameterized as a linear combination of given features with phase-dependent cost weights. We focus on not only recovering the cost weights of each phase, but also estimating the transition points between motion phases.

2.4.1 Problem Formulation

Consider an autonomous agent with dynamics and initial condition in (2.1). Suppose that the agent trajectory of states and inputs over a horizon T in (2.2), rewritten here

$$\boldsymbol{\xi} = \{\boldsymbol{\xi}_k : k = 0, 1, ..., T\} \quad \text{with} \quad \boldsymbol{\xi}_k{=}(\boldsymbol{x}_k^*,\, \boldsymbol{u}_k^*),$$

is a sequential concatenation of p phases, and each phase (locally) minimizes a *different* cost function. The overall cost function is

$$J(\boldsymbol{x}_{0:T}, \boldsymbol{u}_{0:T}) = \sum_{j=1}^{p} \sum_{k=T_j}^{T_{j+1}-1} C^{(j)}(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{2.69}$$

where $C^{(j)}(\boldsymbol{x}, \boldsymbol{u})$ is the running cost in the jth-phase motion, and interval $[T_j, T_{j+1})$ is the time horizon for the jth-phase motion with $T_1 = 0$ and $T_{p+1} = T + 1$. Here, we call T_j $(1 < j \leq p)$ a *phase transition point*. We construct the jth-phase running cost as

$$C^{(j)}(\boldsymbol{x}, \boldsymbol{u}) = {\boldsymbol{\omega}^{(j)}}' \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}), \tag{2.70}$$

where $\boldsymbol{\phi} = [\phi_1, \phi_2, \cdots, \phi_r]' \in \mathbb{R}^r$ is called the *union feature vector*, and $\boldsymbol{\omega}^{(j)} \in \mathbb{R}^r$ are the *cost weights* of the jth-phase motion. We say that the feature

$$\phi_i \text{ is } \begin{cases} relevant & \text{if } \omega_i^{(j)} \neq 0 \\ irrelevant & \text{otherwise} \end{cases} \tag{2.71}$$

for the jth-phase motion. Here, $\omega_i^{(j)}$, i.e., the ith entry of $\boldsymbol{\omega}^{(j)}$, is the cost weight for ϕ_i, $i = 1, 2, \cdots, r$.

Given the observed trajectory $\boldsymbol{\xi}$ and the union feature vector $\boldsymbol{\phi}$, we aim to (i) recover the cost weights $\boldsymbol{\omega}^{(j)}$ of each phase and (ii) estimate the location of the phase transition points T_j $(1 < j \leq p)$. Note that the cost weights of each phase can only be recovered up to a non-zero scaling. Thus, for the jth-phase motion $(1 \leq j \leq p)$, we call the recovered cost weights $\hat{\boldsymbol{\omega}}^{(j)}$ a *successful recovery* if $\hat{\boldsymbol{\omega}}^{(j)} = c\boldsymbol{\omega}^{(j)}$ with $c > 0$; specific c can be obtained by normalization [40].

2.4.2 The Method and Algorithm

Before we develop the method for estimating a multi-phase objective function, we first recall the learning of the cost function within a single phase using the technique of recovery matrix developed in Section 2.2.

Single-Phase Cost Function Recovery

Suppose a segment of the agent trajectory, denoted as $\boldsymbol{\xi}_{t:t+l} = (\boldsymbol{x}^*_{t:t+l}, \boldsymbol{u}^*_{t:t+l})$, where t represents the *observation starting time* and l is called the *observation length*, is from a single phase, say the jth phase (that is, $T_j \leq t < t + l < T_{j+1}$). Based on Corollary 2.3.2, the lemma below provides a method for using the minimal observation length to recover the cost weights within a single phase.

Lemma 2.4.1. *Suppose that the observations of agent trajectory start from time t and are within the jth phase, i.e., $T_j \leq t < t + l - 1 < T_{j+1}$. The recovery matrix $\boldsymbol{H}(t, l)$ is incrementally updated with new observations according to Lemma 2.2.2. The minimal*

observation length required for a successful recovery of the jth phase cost weights $\boldsymbol{\omega}^{(j)}$, defined as $l_{\min}(t)$, is

$$l_{\min}(t) = \min \left\{ l \,|\, \operatorname{rank} \boldsymbol{H}(t, l) = r + n - 1 \right\}. \tag{2.72}$$

If a vector $\operatorname{col}\{\hat{\boldsymbol{\omega}}, \hat{\boldsymbol{\lambda}}\} \neq \mathbf{0}$ *with* $\hat{\boldsymbol{\omega}} \in \mathbb{R}^r$ *is a solution to*

$$\boldsymbol{H}(t, l_{\min}(t)) \begin{bmatrix} \hat{\boldsymbol{\omega}} \\ \hat{\boldsymbol{\lambda}} \end{bmatrix} = \mathbf{0}, \tag{2.73}$$

then $\hat{\boldsymbol{\omega}}$ is a successful recovery for $\boldsymbol{\omega}^{(j)}$.

Proof. Please refer to Corollary 2.3.2 for the proof. □

In Lemma 2.4.1, we implicitly assume that the horizon of the jth phase satisfies $T_{j+1} - T_j - 1 \geq l_{\min}(t)$; and due to (2.2.3) and (2.2.4), $\operatorname{rank} \boldsymbol{H}(t, l_{\min}(t)) \leq \operatorname{rank} \boldsymbol{H}(T_j, T_{j+1} - T_j - 1) \leq r + n - 1$. We thus establish the following assumption for the cost weight 'recoverability' of each phase.

Assumption 2.4.1. *Given the union feature vector $\boldsymbol{\phi}$ in (2.70) and the recovery matrix defined in (2.2.1), the trajectory of states and inputs in the j-th phase $\boldsymbol{\xi}_{T_j:T_{j+1}-1}$ satisfies*

$$\operatorname{rank} \boldsymbol{H}(T_j, T_{j+1} - T_j - 1) = r + n - 1. \tag{2.74}$$

for all $j = 1, 2, \ldots, p$.

Assumption 2.4.1 provides the condition under which the cost weights of each phase can at least be recovered using the whole phase horizon. The validity of this assumption depends on the informativeness of the data in each phase, as analyzed in the previous Section 2.3 and also the union feature set used. To understand this, we consider the following contradiction: assume that given a very large union feature set, there exist non-unique feature combinations which can be used to characterize the jth phase, i.e. there exist two independent cost weight vectors, say $\boldsymbol{\omega}^{(j)}$ and $\tilde{\boldsymbol{\omega}}^{(j)}$, for which $\operatorname{rank} \boldsymbol{H}(T_j, T_{j+1} - T_j - 1) < r + n - 1$, as demonstrated in Section 2.3.3. Based on the previous discussions for the single-phase case, we now consider multiphase cost function recovery.

Multiple-phase Cost Function Recovery

Under Assumption 2.4.1, the idea for recovering the multiphase cost function is to compute the cost weights over time using an observation window moving along the trajectory $\boldsymbol{\xi} = \{\boldsymbol{x}^*_{0:T}, \boldsymbol{u}^*_{0:T}\}$. The procedure includes two ingredients: first, a window, with the starting position at time t with the adaptive length denoted as $l(t)$, moves forward along the trajectory while recovering the cost weights, denoted as $\hat{\boldsymbol{\omega}}(t)$, using the trajectory data within that window via (2.73); second, in an inner loop, the window length $l(t)$ is incrementally determined by finding the minimal observation length $l_{\min}(t)$ defined in (2.72). Therefore, the procedure is a recursive application of Lemma 2.4.1 along the trajectory, and the output is the recovered weights $\hat{\boldsymbol{\omega}}(t)$. Note that the index t in $\hat{\boldsymbol{\omega}}(t)$ and $l(t)$ is used to indicate where the corresponding observation window starts. For the above process, we analyze the following two cases: first, the observations and recovery are performed within the same phase, as shown in Fig. 2.10, and second, the observations are crossing a phase transition point, as shown in Fig. 2.11.

Observations within the Same Phase. Without loss of generality, we suppose that a window with starting time t is in the jth phase, as illustrated in Fig.2.10. When the minimal observation window length $l_{\min}(t)$ (2.72) is successfully found within the same phase, i.e. $T_j \leq t < t + l_{\min}(t) < T_{j+1}$, as shown in Fig. 2.10, the recovery process is just the single-phase case as discussed in the previous subsection.

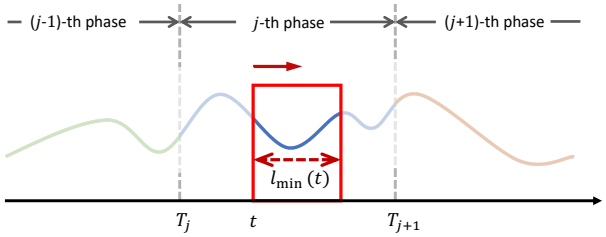


Figure 2.10. An illustration where the minimal observation length is found within the same phase. Here, the observation window is colored in red.

Observations over a Phase Transition Point. We now focus on the case where the window (with the starting time t) is over a phase transition point, say T_{j+1}, as illustrated in Fig. 2.11. This case only happens when the observations from t to $T_{j+1} - 1$ (i.e. the

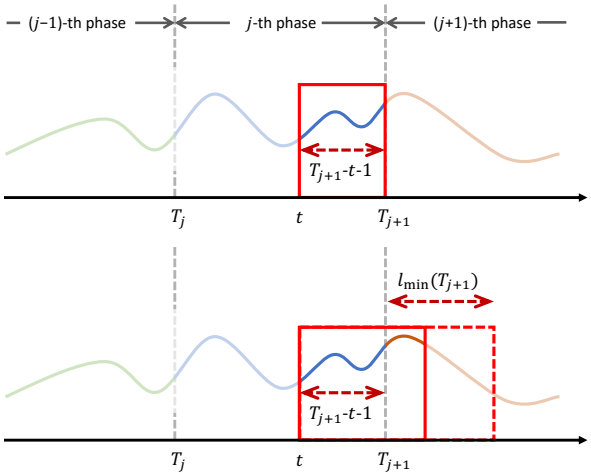


Figure 2.11. An illustration when the window is over a phase transition point. The upper panel shows the case where the window ends at T_{j+1}; the bottom panel shows that the window length increases to include the data of the $(j+1)$th phase.

corresponding observation length is $T_{j+1} - t - 1$ as shown in the upper panel in Fig. 2.11) cannot reach the minimal observation length in (2.72), that is, $\operatorname{rank} \boldsymbol{H}(t, T_{j+1} - t - 1) < n + r - 1$. In what follows, based on the bottom panel in Fig. 2.11, we fix the window starting time t and discuss the rank values of $\boldsymbol{H}(t, l)$ while increasing l from $(T_{j+1} - t - 1)$. First, we have the following lemma.

Lemma 2.4.2. *Suppose that Assumption 2.4.1 holds, and the window starts at time* $t \in [T_j, T_{j+1})$ *which satisfies* $\operatorname{rank} \boldsymbol{H}(t, T_{j+1} - t - 1) < n + r - 1$. *Then there exists an observation length* l *with*

$$l \in [T_{j+1} - t, T_{j+1} - t + l_{\min}(T_{j+1})] \tag{2.75}$$

such that

$$\operatorname{rank} \boldsymbol{H}(t, l) \geq r + n - 1 \tag{2.76}$$

where $l_{\min}(T_{j+1})$ *denotes the minimal observation length from the starting time* $T_{j+1} + 1$.

Proof. Proof by contradiction: assume that rank $\boldsymbol{H}(t,l) < r+n-1$ holds for all $T_{j+1}-t \leq l \leq T_{j+1}-t+l_{\min}(T_{j+1})$. However, due to the rank non-decreasing in (2.2.3), we have

$$\text{rank}\,\boldsymbol{H}(t, T_{j+1}-t+l_{\min}(T_{j+1})) \geq \text{rank}\,\boldsymbol{H}(T_{j+1}, l_{\min}(T_{j+1})) = r+n-1.$$

contradicting the assumption. This completes the proof. □

Based on Lemma 2.4.2, we consider the following two sub-cases when increasing l to include the data of the next phase:

- *Case A*: if there exists l such that rank $\boldsymbol{H}(t,l) = r+n-1$ holds, we still can compute non-trivial weights $\hat{\boldsymbol{\omega}}$ through (2.73); however, such $\hat{\boldsymbol{\omega}}$ may not be a successful recovery of $\boldsymbol{\omega}^{(j)}$ as it is computed using the data from two phases, in this case we call $\hat{\boldsymbol{\omega}}$ a *"degenerate recovery"*; and
- *Case B*: otherwise, increasing l will result in a *direct* jump to rank $\boldsymbol{H}(t,l) = r+n$. In this case, just from the rank value we can say that the window includes a phase transition point.

From the above discussions, we note that when the observations from the previous phase are not sufficient to produce a successful recovery (i.e. the window starting time t satisfying rank $\boldsymbol{H}(t, T_{j+1}-t-1) < n+r-1$), increasing the window length to include the data of the next phase may lead to rank $\boldsymbol{H}(t,l) = n+r-1$ due to degenerate recoveries. This will lead to a possible failure to detect the phase transition points by only observing the rank condition of the recovery matrix (as we shall discuss in experiments, in practice we have not encountered this issue). To circumvent the limitations due to degenerate recovery, we use the cost weights computed at each window to facilitate the estimation of phase transition points. Specifically, as the observation window moves along the trajectory, at any starting time $t \in [T_j, T_{j+1}), (1 \leq j \leq p)$, the cost weights, denoted as $\hat{\boldsymbol{\omega}}(t)$, are computed by:

- *Case A*: if increasing l results in rank $\boldsymbol{H}(t,l) = n+r-1$, here denoted as $l_{\min}(t)$, still compute the weights $\hat{\boldsymbol{\omega}}(t)$ via (2.73).

- *Case B*: otherwise (increasing l only leads to rank $\boldsymbol{H}(t,l) = n+r$, indicating that the window includes a phase transition point), set $\hat{\boldsymbol{\omega}}(t) = \hat{\boldsymbol{\omega}}(t-1)$.

Consequently, by checking the changes of $\hat{\boldsymbol{\omega}}(t)$ over time t, the phase transition point T_{j+1} can be estimated. Considering the degenerate recovery that may happen in Case A, the estimated phase transition point, denoted as $\hat{T}_{j+1}$, is always bounded by

$$t_{\max}^{(j)} \leq \hat{T}_{j+1} \leq T_{j+1} \tag{2.77}$$

where $t_{\max}^{(j)}$ is the last starting time in the jth phase such that rank $\boldsymbol{H}(t, T_{j+1}-t-1) = n+r-1$ holds, that is,

$$\begin{aligned} t_{\max}^{(j)} = \quad & \arg\max \ t \\ & \text{s.t.} \quad \operatorname{rank} \boldsymbol{H}(t, T_{j+1} - t - 1) = n + r - 1. \end{aligned}$$

As we will demonstrate later, when the observation window includes a phase transition point, degenerate recoveries happen infrequently. This is because in most cases the trajectory data of the next phase always violates the optimality condition of the previous phase. Thus, usually when the window includes a small number of data points of the next phase, the rank of the recovery matrix immediately jumps to $(n + r)$.

Implementation

We first describe our implementation for checking the rank condition (2.72) and computing the weights in (2.73). To verify rank $\boldsymbol{H}(t,l) = n + r - 1$, we can check the rank index (2.52), as used in Section 2.3.2, rewritten as below we check the metric

$$\kappa(t,l) = \frac{\sigma_2(\bar{\boldsymbol{H}}(t,l))}{\sigma_1(\bar{\boldsymbol{H}}(t,l))} \geq \gamma. \tag{2.78}$$

The choice of γ has been discussed in Section 2.3.3. To compute (2.73), we use

$$\begin{aligned}\begin{bmatrix}\hat{\boldsymbol{\omega}}\\ \hat{\boldsymbol{\lambda}}\end{bmatrix} &= \arg\min_{\text{col}\{\boldsymbol{\omega},\boldsymbol{\lambda}\}} \left\| \boldsymbol{H}(t, l_{\min}(t)) \begin{bmatrix}\boldsymbol{\omega}\\ \boldsymbol{\lambda}\end{bmatrix} \right\|^2 \\ &\text{s.t.} \quad \sum_{i=1}^{r} \omega_i = 1,\end{aligned} \tag{2.79}$$

where $\|\cdot\|$ denotes the l_2-norm.

Based on the above rules, we now consider the implementation of the recovery procedure to obtain $\hat{\boldsymbol{\omega}}(t)$. Suppose that the window starts at time t. We increase its length $l(t)$ while examining the validity of (2.78):

- *Case A*: if (2.78) is fulfilled for a certain increased $l(t)$, here, denoted as $l_{\min}(t)$, then we compute $\hat{\boldsymbol{\omega}}(t)$ via (2.79).
- *Case B*: otherwise; if (2.78) cannot be fulfilled for *any* $l(t)$ from 1 to $T-t$, set $\hat{\boldsymbol{\omega}}(t) = \hat{\boldsymbol{\omega}}(t-1)$.

To reduce computational cost, in Case B we do not necessarily need to verify for all $1 \le l \le T-t$; instead, we use a maximum window length $l_{\max}$ and only examine (2.78) for $l(t)$ from $\lceil \frac{r+n}{m} \rceil$ ($\lceil \cdot \rceil$ is ceiling operator) to $l_{\max}$. Here, $l_{\max}$ should be larger than any phase horizon, i.e.

$$l_{\max} > T_{j+1} - T_j, \quad \forall\ 1 \le j \le p. \tag{2.80}$$

Thus, we summarize the computation of $\hat{\boldsymbol{\omega}}(t)$ as

$$\hat{\boldsymbol{\omega}}(t) = \begin{cases} \text{computed via (2.79)}, & \text{if } l_{\min}(t) < l_{\max} \\ \hat{\boldsymbol{\omega}}(t-1), & \text{otherwise} \end{cases} \tag{2.81}$$

where $l_{\min}(t) < l_{\max}$ means that the window length satisfying (2.78) is found within $l_{\max}$. Note that in (2.81) the cost weights for the data points near the trajectory terminal, which do not suffice for a minimal observation length, are also considered: their values remain the same as the previous recovery results.

The overall implementation for recovering the multiphase cost functions is presented in Algorithm 2. The choice of γ has been discussed in Section 2.3.3.

Algorithm 2: IOC for multiphase cost functions

Input: trajectory observations $(\boldsymbol{x}^*_{1:T}, \boldsymbol{u}^*_{1:T})$;
a union feature vector $\boldsymbol{\phi}$.
Output: recovered cost weights $\hat{\boldsymbol{\omega}}(t)$ at time $t = 0, 2, \cdots, T$.
Parameter: rank index threshold γ (2.78);
maximum window length $l_{\max}$ (2.80).
for $t = 0 : T$ **do**
  initialize observation length $l(t) = \lceil \frac{r+n}{m} \rceil$;
  initialize the recovery matrix $\boldsymbol{H}(t, l(t))$ (2.26-2.27);
  while $l(t) < l_{\max}$ and *not satisfying (2.78)* **do**
    extend the observation size $l(t) = l(t) + 1$;
    take the next observation $(\boldsymbol{x}^*_{t+l(t)}, \boldsymbol{u}^*_{t+l(t)})$;
    update $\boldsymbol{H}(t, l(t))$ (2.26);
    normalize to obtain $\bar{\boldsymbol{H}}(t, l(t))$ (2.78);
  end
  compute the cost weights $\hat{\boldsymbol{\omega}}(t)$ via (2.81).
end

2.4.3 Numerical Experiments

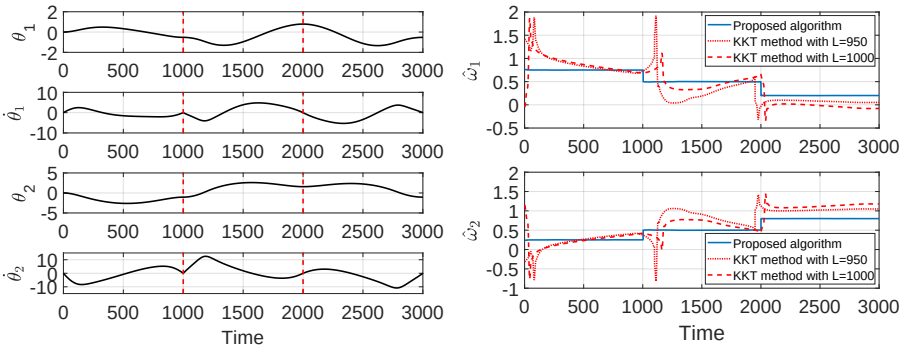
We evaluate the proposed method on a two-link robot arm, given in Appendix A.2. We define the recovery error e_ω to quantify the multiphase IOC accuracy:

$$e_\omega = \frac{\sum_{t=0}^{T} \inf_{c \neq 0} \| c\hat{\boldsymbol{\omega}}(t) - \boldsymbol{\omega}(t) \|}{T} \tag{2.82}$$

where T is the overall horizon, and true $\boldsymbol{\omega}(t) = \boldsymbol{\omega}^{(j)}$ for $T_j \leq t < T_{j+1}$ with $j = 1, 2, \cdots, p$.

The dynamics of a two-link arm is given in Appendix A.2, also used in Section 2.3.3. By defining

$$\boldsymbol{x} = \begin{bmatrix} \theta_1 & \dot{\theta}_1 & \theta_2 & \dot{\theta}_2 \end{bmatrix}' \quad \text{and} \quad \boldsymbol{u} = \boldsymbol{\tau} = \begin{bmatrix} \tau_1 & \tau_2 \end{bmatrix}', \tag{2.83}$$



(a) The state trajectory. (b) Recovered cost weights.

Figure 2.12. Recovery of a three-phase cost function for the robot motion: (a) the state trajectory of the two-link robot arm with the red dotted lines indicating the phase transition points of ground-truth; the ground-truth cost weights for each phase is $\boldsymbol{\omega}^{(1)} = [0.75, 0.25]'$, $\boldsymbol{\omega}^{(2)} = [0.5, 0.5]'$, and $\boldsymbol{\omega}^{(3)} = [0.2, 0.8]'$; and (b) shows the recovered results by the proposed method (blue solid lines) and the KKT method [28] (red dotted/dashed lines).

we write (2.63) in state-space representation and further approximate it to the following discrete-time form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \cdot \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_{k+1}) \tag{2.84}$$

where $\Delta = 0.001$s is the discretization interval.

The motion of the robot arm contains three phases, and each phase has different extents and cost functions. The union feature vector is $\boldsymbol{\phi} = [\tau_1^2, \tau_2^2]'$, where τ_i^2 $(i = 1, 2)$ denotes a quadratic basis function of torque τ_i. In Phase I, the robot moves from $\boldsymbol{x}_0 = \boldsymbol{x}_{T_1} = [0, 0, 0, 0]'$ at $T_1 = 0$ to $\boldsymbol{x}_{T_2} = [-\frac{\pi}{6}, 0, -\frac{\pi}{3}, 0]'$ at $T_2 = 1000$ (1s) with cost weights $\boldsymbol{\omega}^{(1)} = [0.75, 0.25]'$; in Phase 2, from $\boldsymbol{x}_{T_2}$ to $\boldsymbol{x}_{T_3} = [\frac{\pi}{4}, 0, \frac{\pi}{2}, 0]'$ at $T_3 = 2000$ (2s) with $\boldsymbol{\omega}^{(2)} = [0.5, 0.5]'$; and in Phase 3, from $\boldsymbol{x}_{T_3}$ to $\boldsymbol{x}_{T_4} = [-\frac{\pi}{6}, 0, -\frac{\pi}{3}, 0]'$ at $T_4 = 3000$ (3s) with $\boldsymbol{\omega}^{(3)} = [0.2, 0.8]'$. The multiphase optimal control is solved by GPOPS [51] and the optimal state trajectories are shown in Fig. 2.12a.

Given the union feature vector $\boldsymbol{\phi}$, we apply Algorithm 2 to recover the multiphase cost weights from the trajectory given in Fig. 2.12a. We set $\gamma = 100$ and $l_{\max} = 1000$. The results $\hat{\boldsymbol{\omega}}(t)$ are plotted in Fig. 2.12b in blue solid lines. From Fig. 2.12b, we can see that the cost weights of each phase and the phase boundaries are successfully recovered. For example, we

observe that the first phase is from time 0 to 1001 with the average weights $[0.7501, 0.2499]'$ (by averaging $\hat{\boldsymbol{\omega}}(t)$ for $1 \leq t \leq 1001$); the second phase from 1002 to 2000 with the average weights $[0.4998, 0.5002]'$; and the third phase from 2001 to 3000 with the average weights $[0.1998, 0.8012]'$.

Observation Noise. We evaluate the performance of the proposed method by adding varying levels of white Gaussian noise to the trajectory data, and the results are listed in Table 2.3. From Table 2.3, we can conclude that (i) the average minimum observation length increases if the trajectory noise is high; and (ii) since the increased window length includes more data points to mitigate noise, the recovery accuracy maintains high.

Table 2.3. Results of Multi-phase IOC (Algorithm 2, $\gamma = 100$, $l_{\max} = 1000$) under different noise levels.

Noise level	e_ω	Avg. $l_{\min}$ *	$\hat{T}_1$ and $\hat{T}_2$	e_ω for KKT method *
$1e-4$	0.0019	145.3	$1004, 2000$	0.15
$1e-3$	0.0016	236.0	$1004, 2000$	0.15
$1e-2$	0.0014	554.5	$1004, 2002$	0.18

* The averaged $l_{\min}$ is computed by averaging $l_{\min}(t)$ for $1 \leq t \leq T$; and e_ω for KKT method is evaluated with window length $L = 950$.

Comparison with Related Work. We compare the proposed method with the KKT method [28]. In [28], a window of manually-specified length moves along the trajectory, and the weights are computed by minimizing the violation of KKT conditions. Here, we set the window length $L = 950$ and $L = 1000$, and plot the corresponding recoveries in Fig. 2.12b using red dotted and red dashed lines, respectively. We also test the recovery error for $L = 950$ under different noise levels and summarize the results in Table 2.3.

The results illustrate that although being able to discriminate motion phases, the KKT method does not consistently produce the correct cost weights (the recovery errors for $L = 950$ and $L = 1000$ are 0.18 and 0.15, respectively), and the estimated phase transition points have high errors. We also find that the KKT method is sensitive to the choice of window length: a larger window length will improve recovery accuracy, but may lead to inaccuracy for the phase transition point estimation.

This is because the KKT method only uses current window data and does not consider the influence of future data beyond the window on the recovery, inevitably leading to a recovery error. This future information is encoded in the costate $\boldsymbol{\lambda}$ in (2.73) in our formulation. When the observed data is of 'low richness', e.g. of a small window length, the influence of future information becomes relatively significant, thus leading to a large recovery error. Thus, the KKT method always requires a large window, but this will potentially deteriorate the accuracy of phase boundary detection. In Fig. 2.12b, we also observe that the KKT method results in a detection delay for the first phase transition point. This may be because when the window is over the transition point, the included data from the first phase is more expressive compared to that of the second phase, thus contributing more to the computed cost weight and making the results look more like the ones of the first phase.

2.5 Distributed IOC

As identified in Section 2.1.2, existing IOC techniques are designed in a centralized way. This limits their implementations to the practical situation when the data storage exceeds the memory capacity of a single processor or the computational demand exceeds the capability of the processor. For the former restriction, although the recovery matrix developed in Section 2.2 allows us to handle trajectory segments, it still requires a large chunk of consecutive trajectory data to satisfy the rank condition. In this section, we address such challenges by developing a distributed IOC approach, which enables the learning of an objective function jointly by multiple processors, each of which only observes and processes smaller part of data that could be much smaller than that required by the recovery matrix.

2.5.1 Problem Formulation

Consider an optimal control agent with dynamics and initial condition in (2.1), and its trajectory, rewritten below,

$$\boldsymbol{\xi} = \{\boldsymbol{\xi}_t, t = 0, 1, ..., T\} \quad \text{with} \quad \boldsymbol{\xi}_t = \{\boldsymbol{x}_t^*, \boldsymbol{u}_t^*\},$$

is (locally) minimizing a control cost function (2.3) with unknown weights $\boldsymbol{\omega}$.

Consider a number of N processors where each processor i only communicates with its neighbors denoted as a set $\mathcal{N}_i$ (we here assume $i \in \mathcal{N}_i$). Let a graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$ denote the communication graph of all processors, where the node set $\mathcal{V} = \{1, 2, \cdots, N\}$ represents the N processors, and the edge set $\mathcal{E} = \{(i, j)\} \subset V \times V$ represents the available communication channels: there is an edge $(i, j) \in \mathcal{E}$ if and only if processors i and j are neighbors. We consider that $\mathbb{G}$ is connected, undirected, and time-invariant. As shown in Fig. 2.13, we suppose each processor i observes a collection of s_i *trajectory segments*, i.e., s_i segments of $\boldsymbol{\xi}$, defined as

$$\mathcal{S}_i = \{\boldsymbol{\xi}_{\underline{t}_{ij}:\bar{t}_{ij}}, j = 1, 2, ..., s_i\}, \tag{2.85}$$

where $\boldsymbol{\xi}_{\underline{t}_{ij}:\bar{t}_{ij}}$ is the jth segment observed by processor i, with $\underline{t}_{ij}$ and $\bar{t}_{ij}$ being its starting and ending time, $0 \leq \underline{t}_{ij} \leq \bar{t}_{ij} \leq T$. Note that we do not put any restriction to $\mathcal{S}_i$, which means that $\mathcal{S}_i$ could include only one trajectory segment, i.e. $s_i = 1$, as shown by Processor 3 in Fig. 2.13; or even permit a segment of length of 1, i.e., $s_i = 1$ and $\bar{t}_{ij} - \underline{t}_{ij} = 1$, as shown by Processor 2; Processors are also allowed to have overlaps in their trajectory segments as shown by Processors N-1 and N.

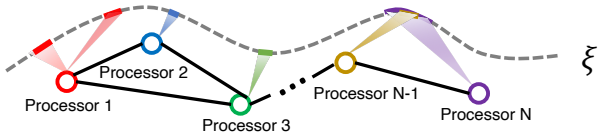


Figure 2.13. Illustration of distributed inverse optimal control.

Since the available trajectory segments $\mathcal{S}_i$ for each processor i is usually not sufficient to determine $\boldsymbol{\omega}$ alone, **the problem of interest** is to develop a distributed IOC, which enables all processors to collaboratively achieve $\boldsymbol{\omega}$ by communicating with its neighbors. Note that scaling $\boldsymbol{\omega}$ by a positive non-zero constant does not affect the IOC because a scaled $\boldsymbol{\omega}$ can lead to the same $\boldsymbol{\xi}$. Without losing generality, we here fix $\omega_1 = 1$, i.e., $[1, 0, \cdots, 0]\boldsymbol{\omega} = 1$.

2.5.2 The Method and Algorithm

In this part, we will first introduce the concept of IOC-effectiveness, a way to evaluate whether a trajectory segment can contribute to IOC, and if so, such segment will impose a liner constraint to the unknown objective function weights. We then establish IOC identifiability from trajectory segments. Finally, we develop a distributed algorithm to enable all processors to collaboratively learn the objective function weights exponentially fast by communicating with their neighbors.

IOC-Effective Trajectory Segments

According to the recovery matrix in Lemma 2.2.1, for any trajectory segment $\boldsymbol{\xi}_{\underline{t},\bar{t}} \subseteq \boldsymbol{\xi}$, one directly has

$$\boldsymbol{H}_1(\underline{t}, \bar{t} - \underline{t})\boldsymbol{\omega} + \boldsymbol{H}_2(\underline{t}, \bar{t} - \underline{t})\boldsymbol{\lambda}_{\bar{t}+1} = \mathbf{0}.$$

For clear exposition of data dependence, we write $\boldsymbol{H}_1(\underline{t}, \bar{t} - \underline{t})$ and $\boldsymbol{H}_2(\underline{t}, \bar{t} - \underline{t})$ as $\boldsymbol{H}_1(\boldsymbol{\xi}_{\underline{t}:\bar{t}})$ and $\boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}})$, respectively, i.e.,

$$\boldsymbol{H}_1(\boldsymbol{\xi}_{\underline{t}:\bar{t}})\boldsymbol{\omega} + \boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}})\boldsymbol{\lambda}_{\bar{t}+1} = \mathbf{0} \tag{2.86}$$

Equation (2.86) provides a relationship between any trajectory segment data $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$, the weights $\boldsymbol{\omega}$, and the costate $\boldsymbol{\lambda}_{\bar{t}+1}$. To further eliminate $\boldsymbol{\lambda}_{\bar{t}+1}$ and measure the contribution of $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$ to recovering $\boldsymbol{\omega}$, we define the following:

Definition 2.5.1 (IOC-Effective Trajectory Segment)**.** *Given the agent dynamics (2.1) and an arbitrary segment of the trajectory, $\boldsymbol{\xi}_{\underline{t}:\bar{t}} \subseteq \boldsymbol{\xi}$, we say $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$ is IOC-effective if*

$$\operatorname{rank} \boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}}) = n. \tag{2.87}$$

Given an IOC-effective $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$, one has the following result.

Lemma 2.5.1. *For a trajectory segment $\boldsymbol{\xi}_{\underline{t}:\bar{t}} \subseteq \boldsymbol{\xi}$ that is IOC-effective, $\boldsymbol{\omega}$ must satisfy the following linear equation:*

$$\boldsymbol{R}(\boldsymbol{\xi}_{\underline{t}:\bar{t}})\boldsymbol{\omega} = \mathbf{0}, \tag{2.88}$$

where

$$\boldsymbol{R}(\boldsymbol{\xi}_{\underline{t}:\bar{t}}) = \boldsymbol{H}_1 - \boldsymbol{H}_2\Big(\boldsymbol{H}_2{}'\boldsymbol{H}_2\Big)^{-1}\boldsymbol{H}_2'\boldsymbol{H}_1 \in \mathbb{R}^{r\times r}. \tag{2.89}$$

Proof. For an IOC-effective trajectory segment $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$, it follows that $\boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}})'\boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}})$ is non-singular. By multiplying $\boldsymbol{H}_2{}'$ on both sides of (2.86), one has $\boldsymbol{\lambda}_{\bar{t}+1} = -\Big(\boldsymbol{H}_2{}'\boldsymbol{H}_2\Big)^{-1}\boldsymbol{H}_2'\boldsymbol{H}_1\boldsymbol{\omega}$, which is then substituted back into (2.86), and this leads to (2.88). Thus Lemma 2.5.1 holds. □

Here, we would like to make a few comments on the above concept of the IOC-effectiveness of trjectory segments. First, as suggested by (2.8), matrix $\boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}})$ is uniquely determined by $\boldsymbol{F}_x(\boldsymbol{\xi}_{\underline{t}:\bar{t}})$, $\boldsymbol{F}_u(\boldsymbol{\xi}_{\underline{t}:\bar{t}})$ and $\boldsymbol{V}(\boldsymbol{\xi}_{\underline{t}:\bar{t}})$ in (2.9), which only depend on the segment $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$ and dynamics $\boldsymbol{f}$. Thus, whether a segment is effective or not is *independent of the choices of features* $\boldsymbol{\phi}$, but is only determined by the data in the trajectory segment and the specific dynamics model.

Second, the data effectiveness condition (2.87) can be satisfied by including more state-input points, as suggested by the iterative property of $\boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}+1})$ below.

Lemma 2.5.2. *For any* $1 \le \underline{t} < \bar{t} < T$*, one has*

$$\boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}+1}) = \begin{bmatrix} \boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}}) \\ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{\bar{t}+1}} \end{bmatrix} \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{\bar{t}+1}}, \tag{2.90}$$

with $\boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}}) = \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^*_{\underline{t}}}\frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{\underline{t}}}$ *for* $\bar{t} = \underline{t} + 1$.

Proof. The proof can be found in the proof of Lemma 2.2.2. □

Lemma 2.5.2 implies

$$\operatorname{rank} \boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}+1}) \ge \operatorname{rank} \boldsymbol{H}_2(\boldsymbol{\xi}_{\underline{t}:\bar{t}}), \tag{2.91}$$

if $\frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^*_{\bar{t}+1}}$ is non-singular. The above rank non-decreasing property suggests that the more data a trajectory segment includes, the more likely it will be IOC-effective. As shown later in simulation, a segment $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$ can easily be IOC-effective if $\bar{t} - \underline{t} \ge \lceil \frac{n}{m} \rceil$ ($\lceil \cdot \rceil$ is ceiling operator) due to the size of $\boldsymbol{H}_2$. Specifically, if $m = n$, even a segment with length of 1 (i.e. only including two points) can be IOC-effective.

Third, from Lemma 2.5.2, an interesting observation is that the IOC effectiveness in Definition 2.5.1 is equivalent to the controllability for liner time-invariant systems. Specifically, suppose that the dynamics model in (2.1) is $\boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t$ and a trajectory segment $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$ has the length $\bar{t} - \underline{t} = n$, then from Lemma 2.5.2, one has

$$E(\boldsymbol{\xi}_{\underline{t}:\bar{t}}) = \begin{bmatrix} A^{n-1}B & A^{n-2}B & \cdots & AB & B \end{bmatrix}', \tag{2.92}$$

which is equivalent to the controllability matrix. This means that if the linear system is not controllable, i.e., rank $\boldsymbol{E}(\boldsymbol{\xi}_{\underline{t}:\bar{t}}) < n$, then any segment of its trajectory $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$ will never be IOC-effective regardless of how many state-input points it contains. Thus, the IOC effectiveness depends on the specific dynamics model $\boldsymbol{f}$.

IOC Identifiability

Based on Lemma 2.5.1, we next present an important result stating the identifiability of the cost function weights from trajectory segments.

Theorem 2.5.1. *Given a collection of s trajectory segments $\mathcal{S} = \{\boldsymbol{\xi}_{\underline{t}_j:\bar{t}_j}, j = 1, 2, \cdots s\}$, define the matrix*

$$\boldsymbol{R}(\mathcal{S}) = \text{col}\,\{\boldsymbol{R}(\boldsymbol{\xi}_{\underline{t}_j,\bar{t}_j}) \,|\, \boldsymbol{\xi}_{\underline{t}_j,\bar{t}_j} \in \mathcal{S} \text{ and } \boldsymbol{\xi}_{\underline{t}_j,\bar{t}_j} \text{ is IOC-effective}\},$$

which is a stack of $\boldsymbol{R}(\boldsymbol{\xi}_{\underline{t}_j,\bar{t}_j})$ for all IOC-effective $\boldsymbol{\xi}_{\underline{t}_j:\bar{t}_j}$ with $\boldsymbol{R}(\boldsymbol{\xi}_{\underline{t}_j,\bar{t}_j})$ in Lemma 2.5.1. If

$$\text{rank}\,\boldsymbol{R}(\mathcal{S}) = r - 1, \tag{2.93}$$

then the weights $\boldsymbol{\omega}$ can be identified from $\mathcal{S}$, meaning that any nonzero $\boldsymbol{v} \in \ker \boldsymbol{R}(\mathcal{S})$ is a scaled version of $\boldsymbol{\omega}$, i.e., there exists a scalar $c \neq 0$ such that $\boldsymbol{v} = c\boldsymbol{w}$.

Proof. For any IOC-effective trajectory segment $\boldsymbol{\xi}_{\underline{t}:\bar{t}} \in \mathcal{S}$, since the cost function weights $\boldsymbol{\omega}$ must satisfy condition (2.88) by Lemma 2.5.1, then $\boldsymbol{R}(\mathcal{S})\boldsymbol{\omega} = \boldsymbol{0}$ and rank $\boldsymbol{R}(\mathcal{S}) \leq r - 1$. (2.93) indicates that the nullity of the matrix $\boldsymbol{R}(\mathcal{S})$ is one, and it follows that any non-zero vector $\boldsymbol{v} \in \ker \boldsymbol{R}(\mathcal{S})$ will have $\boldsymbol{v} = c\boldsymbol{w}$ with $c \neq 0$. This completes the proof. □

The above result states that given a set of trajectory segments $\mathcal{S}$, if (2.93) is satisfied, then the cost function weights $\boldsymbol{\omega}$ can be 'uniquely' determined by $\mathcal{S}$. Here the uniqueness means that the obtained weights are a scaled version of true weights, and this is best we can obtain because the scaled $\boldsymbol{\omega}$ does not affect IOC.

A Distributed Algorithm for IOC

We now consider a graph $\mathbb{G}$ of N processors and each processor i has only access to a set of trajectory segments $\mathcal{S}_i$ as in (2.85). By Theorem 2.5.1, one has

$$\boldsymbol{R}(\mathcal{S}_i)\boldsymbol{\omega} = \mathbf{0}, \quad i = 1, 2, ..., N. \tag{2.94}$$

Specially, $\boldsymbol{R}(\mathcal{S}_i) = \mathbf{0}$ if there is no IOC-effective trajectory segment in $\mathcal{S}_i$. Thus, each processor i with trajectory segments $\mathcal{S}_i$ only knows $\boldsymbol{R}(\mathcal{S}_i)$. Then all we need is to develop distributed algorithms in [52] for N processors to cooperatively solve the group of linear equations in (2.94). Note that these linear equations in (2.94) in practice may not have exact solutions because of noises and violation of the optimalilty KKT condition. We will instead develop a distributed algorithm for all processors to solve:

$$\min_{\boldsymbol{\omega}} \frac{1}{2}\sum_{i=1}^{N} \|\boldsymbol{R}(\mathcal{S}_i)\boldsymbol{\omega}\|_2^2 \quad \text{s.t.} \quad [1, 0, \cdots, 0]\,\boldsymbol{\omega} = 1 \tag{2.95}$$

In order to achieve the above least-square solution, we let each processor i control a state vector $\boldsymbol{\omega}_i \in \mathbb{R}^r$, which can be viewed as a guess of the solution to (2.95). This leads to the following distributed optimization problem.

$$\min_{\{\boldsymbol{\omega}_1, \cdots, \boldsymbol{\omega}_N\}} \quad \sum_{i=1}^{N} \frac{1}{2}\|\boldsymbol{R}(\mathcal{S}_i)\boldsymbol{\omega}_i\|_2^2 \tag{2.96}$$

$$\text{s.t.} \quad [1, 0, ..., 0]\boldsymbol{\omega}_i = 1, \quad i = 1, 2, \cdots, N, \tag{2.97}$$

$$\boldsymbol{\omega}_1 = \boldsymbol{\omega}_2 =, \cdots, = \boldsymbol{\omega}_N. \tag{2.98}$$

To solve (2.96-2.98), one may use existing distributed optimization algorithms in [53, 54], which normally require all processors share a step-size. To remove the requirement of shared

step-size while still achieving exponential convergence, we modify the algorithm in [55] and develop a new distributed update. Here we introduce extra states $\mu_i \in \mathbb{R}$ and $\boldsymbol{v}_i \in \mathbb{R}^r$ for each processor i to account for the constraints (2.97) and (2.98), respectively, and propose the following update (k is the iteration index):

$$\begin{aligned}
\boldsymbol{\omega}_i(k+1) &= \boldsymbol{G}_i \boldsymbol{y}_i(k) + \boldsymbol{G}_i \sum_{j\in\mathcal{N}_i} \Big(\boldsymbol{\omega}_j(k) + \boldsymbol{v}_j(k)\Big), \\
\mu_i(k+1) &= \mu_i(k) + \frac{1}{d_i}\boldsymbol{\alpha}'\boldsymbol{G}_i \sum_{j\in\mathcal{N}_i} \Big(\boldsymbol{\omega}_j(k) + \boldsymbol{v}_j(k)\Big) + \frac{1}{d_i}\boldsymbol{\alpha}'\boldsymbol{G}_i\boldsymbol{y}_i(k) - \frac{1}{d_i}, \\
\boldsymbol{v}_i(k+1) &= \boldsymbol{v}_i(k) + \boldsymbol{G}_i \sum_{j\in\mathcal{N}_i} \Big(\boldsymbol{\omega}_j(k) + \boldsymbol{v}_j(k)\Big) + \boldsymbol{G}_i\boldsymbol{y}_i(k) - \frac{1}{d_i}\sum_{j\in\mathcal{N}_i}\boldsymbol{\omega}_j(k)
\end{aligned} \tag{2.99}$$

where

$$\begin{aligned}
\boldsymbol{G}_i &= (\boldsymbol{R}(\mathcal{S}_i)'\boldsymbol{R}(\mathcal{S}_i) + \frac{1}{d_i}\boldsymbol{\alpha}\boldsymbol{\alpha}' + 2d_i\boldsymbol{I}_r)^{-1}, \\
\boldsymbol{y}_i(k) &= d_i\boldsymbol{\omega}_i(k) - \boldsymbol{\alpha}\mu_i(k) - d_i\boldsymbol{v}_i(k) + \frac{1}{d_i}\boldsymbol{\alpha},
\end{aligned} \tag{2.100}$$

with $\boldsymbol{\alpha} = [1, 0, ..., 0]' \in \mathbb{R}^r$ and $d_i = |\mathcal{N}_i|$, which is the cardinality of a set. Note that the update rule in (2.99 for each processor is distributed in a sense that it only utilizes information from its neighbors. Furthermore, we have the following result.

Theorem 2.5.2. *Consider N processors that coordinates over a graph $\mathbb{G}$, where each processor i obtains a set of trajectory segments $\mathcal{S}_i$ of the trajectory $\boldsymbol{\xi}$. Given an arbitrary initial state (and extra states) of each processor, the distributed update rule (2.99) enables all processors to achieve the solution to (2.95) exponentially fast.*

Proof. Let $W \in \mathbb{R}^{N\times N}$ denote the adjacency matrix for $\mathbb{G}$. The corresponding Laplacian matrix L is defined as $L = D - W$ with $D = \text{diag}\,\{d_1, ..., d_N\}$ and $d_i = |\mathcal{N}_i|$. Define

$$\begin{gathered}
\bar{\boldsymbol{D}} = D \otimes \boldsymbol{I}_r, \quad \bar{\boldsymbol{W}} = W \otimes \boldsymbol{I}_r, \\
\bar{\boldsymbol{\omega}} = \text{col}\,\{\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_N\}, \\
\bar{\boldsymbol{\mu}} = \text{col}\,\{\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_N\}, \quad \bar{\boldsymbol{v}} = \text{col}\,\{\boldsymbol{v}_1, ..., \boldsymbol{v}_N\}, \\
\bar{\boldsymbol{R}} = \text{diag}\,\{\boldsymbol{R}(\mathcal{S}_1), ..., \boldsymbol{R}(\mathcal{S}_N)\}, \\
\bar{\boldsymbol{A}} = \boldsymbol{I}_N \otimes [1, 0, ..., 0]', \quad \bar{\boldsymbol{L}} = L \otimes \boldsymbol{I}_r = \bar{\boldsymbol{D}} - \bar{\boldsymbol{W}},
\end{gathered}$$

where $\otimes$ is the Kronecker product, and $\boldsymbol{I}_r \in \mathbb{R}^{r\times r}$ is the identity matrix. The compact form for the distributed updates in (2.99) then can be equivalently written as

$$\boldsymbol{h}(k+1) = \boldsymbol{Q}\boldsymbol{h}(k) + \boldsymbol{q}, \tag{2.101}$$

with $\boldsymbol{h} = \text{col}\,\{\bar{\boldsymbol{\omega}}, \bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\lambda}}\}$, and

$$\boldsymbol{Q} = \begin{bmatrix} \bar{\boldsymbol{D}}+\bar{\boldsymbol{R}}'\bar{\boldsymbol{R}} & \bar{\boldsymbol{A}} & \bar{\boldsymbol{D}} \\ -\bar{\boldsymbol{A}}' & D & \boldsymbol{0} \\ -\bar{\boldsymbol{D}} & \boldsymbol{0} & \bar{\boldsymbol{D}} \end{bmatrix}^{-1} \begin{bmatrix} \bar{\boldsymbol{D}} & \boldsymbol{0} & \bar{\boldsymbol{W}} \\ -\boldsymbol{0}' & D & \boldsymbol{0} \\ -\bar{\boldsymbol{W}} & \boldsymbol{0} & \bar{\boldsymbol{D}} \end{bmatrix}, \tag{2.102}$$

$$\boldsymbol{q} = \begin{bmatrix} \bar{\boldsymbol{D}}+\bar{\boldsymbol{R}}'\bar{\boldsymbol{R}} & \bar{\boldsymbol{A}} & \bar{\boldsymbol{D}} \\ -\bar{\boldsymbol{A}}' & D & \boldsymbol{0} \\ -\bar{\boldsymbol{D}} & \boldsymbol{0} & \bar{\boldsymbol{D}} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{0} \\ -\boldsymbol{1}_N \\ \boldsymbol{0} \end{bmatrix}. \tag{2.103}$$

Let $\boldsymbol{h}^* = \text{col}\,\{\bar{\boldsymbol{\omega}}^*, \bar{\boldsymbol{\mu}}^*, \bar{\boldsymbol{v}}^*\}$ denotes an equilibrium the linear time-invariant system (2.101), namely, $\boldsymbol{h}^* = \boldsymbol{Q}\boldsymbol{h}^* + \boldsymbol{q}$. By following the similar argument to the distributed algorithm for least-square solutions in [55], one is able to show there exists $\boldsymbol{\omega}^* \in \mathbb{R}^r$ such that $\bar{\boldsymbol{\omega}}^*=\boldsymbol{1}_N \otimes \boldsymbol{\omega}^*$ globally minimizes (2.96-2.98). Thus, we only need to prove that given any initial condition $\boldsymbol{h}(0)$, $\boldsymbol{h}(k)$ governed by the dynamics (2.101) converges exponentially fast to the equilibrium $\boldsymbol{h}^*$.

To this end, we define the error $\boldsymbol{e}(k) = \boldsymbol{h}(k) - \boldsymbol{h}^*$, and from (2.101) we have

$$\boldsymbol{e}(k+1) = \boldsymbol{Q}\boldsymbol{e}(k). \tag{2.104}$$

The above error dynamics (2.104) and matrix $\boldsymbol{Q}$ share the similar structure with the ones in Lemma 2 in [55]. Thus, following a similar proof procedure, one can show that there exists a matrix $\boldsymbol{T}$ such that

$$\boldsymbol{Q} = \boldsymbol{T} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{P} \end{bmatrix} \boldsymbol{T}^{-1}, \tag{2.105}$$

where $\boldsymbol{P}$ has the eigenvalue of the maximum magnitude satisfying $\rho_2 = \max\{|\lambda| : \lambda \in \text{eig}(\boldsymbol{P})\} < 1$. From (2.105) and (2.104), we can write $\boldsymbol{e}(k) = \boldsymbol{e}^* + \boldsymbol{\eta}(k)$, where

$$\boldsymbol{e}^* = \boldsymbol{T}\begin{bmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\boldsymbol{T}^{-1}\boldsymbol{e}(0), \quad \boldsymbol{\eta}(k) = \boldsymbol{T}\begin{bmatrix}\boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{P}^k\end{bmatrix}\boldsymbol{T}^{-1}\boldsymbol{e}(0).$$

Here $\boldsymbol{e}^*$ satisfies $\boldsymbol{Q}\boldsymbol{e}^* = \boldsymbol{e}^*$, and $\boldsymbol{\eta}(k) \to \boldsymbol{0}$ when $k \to \infty$ with a diminishing rate of $\rho_2^k \to 0$. Thus, when $k \to \infty$, $\boldsymbol{e}(k) = \boldsymbol{h}(k) - \boldsymbol{h}^* \to \boldsymbol{e}^*$, yielding $\boldsymbol{h}(k) \to (\boldsymbol{e}^* + \boldsymbol{h}^*) = \bar{\boldsymbol{h}}^*$ with a diminishing rate of $\rho_2^k \to 0$. Since $\bar{\boldsymbol{h}}^*$ also is an equilibrium of (2.101), the theorem assertion follows. □

2.5.3 Numerical Experiments

We evaluate the proposed method on a simulated two-link robot arm given in Appendix A.2, with dynamics of a two-link arm in (A.2). Define the system state variable $\boldsymbol{x} = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]'$ and control input $\boldsymbol{u} = [\tau_1,\ \tau_2]'$, and discretize the dynamics by Euler method with a time interval 0.05s. The arm is controlled to minimize (2.3) here with

$$\boldsymbol{\phi} = [\tau_1^2 + \tau_2^2, (\theta_1 - \frac{\pi}{2})^2, \theta_2^2, \dot{\theta_1}^2 + \dot{\theta_2}^2]' \quad \text{and} \quad \boldsymbol{\omega} = [1, 5, 4, 1]'. \tag{2.106}$$

The initial state $\boldsymbol{x}_0 = [-\frac{\pi}{2}, 0, 0, 0]'$ and the overall horizon $T = 100$. The optimal trajectory of the arm system is in Fig. 2.14.

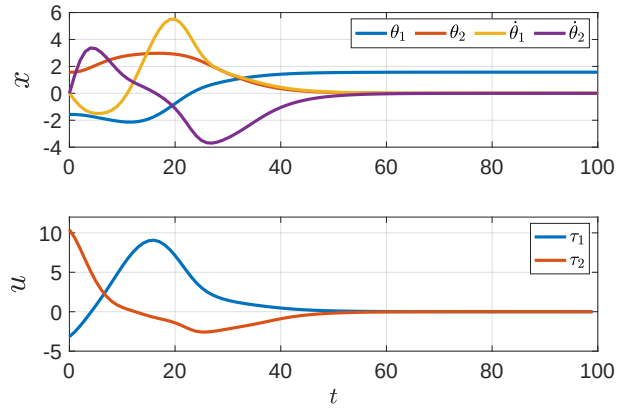


Figure 2.14. Optimal trajectory of robot arm with $\boldsymbol{\omega} = [1, 5, 4, 1]'$.

IOC Identifiability

We first demonstrate the IOC identifiability using segments. We arbitrarily choose a trajectory segment $\boldsymbol{\xi}_{\underline{t}:\bar{t}}$ in Fig. 2.14, $[\underline{t}, \bar{t}] = [10, 20]$, and then check the rank of the corresponding matrix $\boldsymbol{R}(\boldsymbol{\xi}_{10:20})$, rank $\boldsymbol{R}(\boldsymbol{\xi}_{10:20}) = 3 = r - 1$. By Theorem 2.5.1, this confirms that the weights are identifiable from $\boldsymbol{\xi}_{10:20}$. Thus by solving the linear equation $\boldsymbol{R}(\boldsymbol{\xi}_{10:20})\boldsymbol{\omega} = \mathbf{0}$ and let $\omega_1 = 1$, we exactly obtain the true weight $\boldsymbol{\omega} = [1.00, 5.00, 4.00, 1.00]'$. In another case we choose a trajectory segment in $[\underline{t}, \bar{t}] = [10, 12]$, we check rank $\boldsymbol{R}(\boldsymbol{\xi}_{10:12}) = 2 < r - 1$, which indicates that we cannot obtain the true $\boldsymbol{\omega}$ because $\boldsymbol{\omega}$ is not identifiable from $\boldsymbol{\xi}_{10:12}$ according to Theorem 2.5.1.

Distributed IOC

Table 2.4. Processor graph and segments (the overall time horizon $T = 100$)

Processor i	Neighbors $\mathcal{N}_i$	Intervals $[\underline{t}, \bar{t}]$ of segments
1	{1, 2, 3}	[10, 15], [20, 25]
2	{2, 1, 4}	[30, 31]
3	{3, 1, 4}	[50, 52], [61, 65]
4	{4, 2, 3, 5}	[65, 66]
5	{5, 4}	[75, 78], [90, 97]

In distributed IOC problems, we consider $N = 5$ processors in a network cooperatively solve the weights $\boldsymbol{\omega}$. For each processor, its neighbors are listed in Table 2.4, and the time intervals of the trajectory segments observed by each processor are also shown in Table 2.4. For each trajectory segment, the processor i checks its IOC-effectiveness by Definition 2.5.1 before integrating it to the matrix $\boldsymbol{R}(\mathcal{S}_i)$. It shows that all trajectory segments in Table 2.4 are effective. Here, we note that although the segments by Processors 2 and 4 each only has length of 1 (i.e., only two trajectory points), which just reaches the necessary lower bound length $\bar{t} - t = 1$, they are IOC-effective (obviously they cannot suffice to determine the weights alone, as shown in the previous simulation). This indicates the mild requirement of the effectiveness condition (2.87).

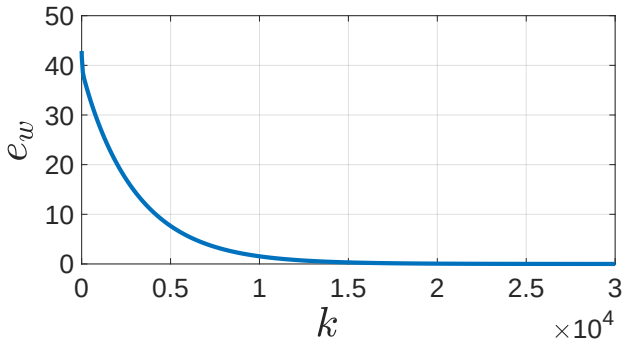


Figure 2.15. Error index $e_{\boldsymbol{\omega}}$ versus iteration k.

Each processor i $(1 \leq i \leq 5)$ generates $\boldsymbol{R}(\mathcal{S}_i)$. Then we apply the proposed distributed IOC approach in Theorem 2.5.2 to let all processors collaboratively solve the weights, given an random initial guess of the states for each processor. To quantify recovery accuracy, we define the following error

$$e_{\boldsymbol{\omega}} = \frac{1}{N}\sum\nolimits_{i=1}^{N} \|\boldsymbol{\omega}_i - \boldsymbol{\omega}\|_2^2. \tag{2.107}$$

We plot the error versus iteration in Fig. 2.15, from which we conclude that the weights of each processor exponentially converge to the true $\boldsymbol{\omega} = [1, 5, 4, 1]'$. At 3×10^4th iteration, $\boldsymbol{e}_{\boldsymbol{\omega}} = 4.1072 \times 10^{-6}$, and the average weights for all processors are $\sum_{i=1}^{5} \boldsymbol{\omega}_i/5 = [1.0000\, 4.998\, 3.999\, 1.000]'$, which shows the effectiveness of the proposed method.

2.6 Applications

This section presents the applications of the previous developed IOC techniques in human motion analysis. A well-justified assumption is that human motor control follows certain optimality [17, 56], and a common practice in biological movement and behavioral studies is to use optimal control models to characterize human motor control (see [16] for an overview of related work). Under such assumption, in this section, we use the IOC techniques developed in the previous sections to perform human motion segmentation and prediction.

2.6.1 Human Motion Segmentation

The first part of this section focuses on the application of human motion segmentation. We segment a consecutive human motion trajectory into different phases by identifying the different cost functions underlying the motion. Here, we apply the multi-phase objective IOC technique developed in Section 2.4, which enables to identify cost functions of different motion phases and also estimate the phase transition points.

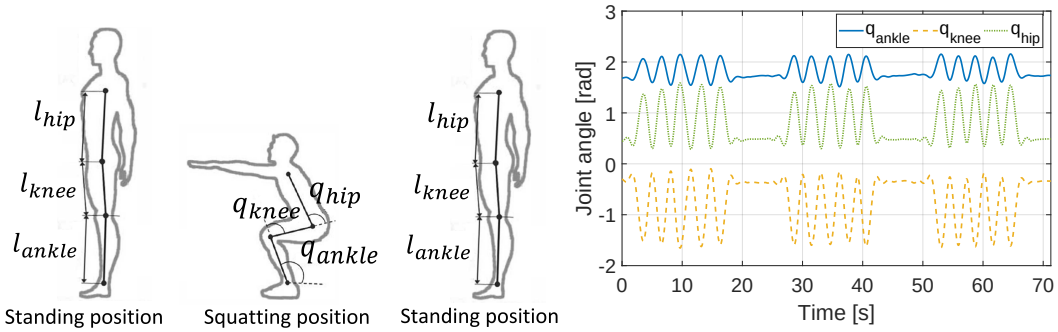
Data Collection

We apply the multi-phase objective IOC technique developed in Section 2.4 to a human motion dataset. We choose the human squat motion [57] (Fig. 2.16a) as it is a common and full-body exercise studied in both athletics and rehabilitation [58]. The squat dataset was collected from 6 (5M, 1F, $\mu_{age} = 26.2$) healthy participants. Each participant performed 15 squats (Fig. 2.16a) in 3 sets, with 5 repetitions in each set. All squats are recorded *in a single recording* via the motion capture system, where an 80-marker model was used, providing Cartesian positions. Joint angles were then computed via inverse kinematics [57] and converted to a 3 DOF planar model, as shown in Fig. 2.16a, corresponding to q_{ankle}, q_{knee}, and q_{hip}. The motion capture system has a sampling rate $\Delta = 0.01$s. The obtained joint trajectories were smoothed by a moving Savitzky-Golay filter [59] (span 2s and degree 10). This allows to suppress noise and compute smooth trajectory derivatives. The joint velocity and acceleration are then computed by numerical differentiation. Fig. 2.16b plots the joint trajectories for a sample participant.

Body Dynamics Model and Feature Selection

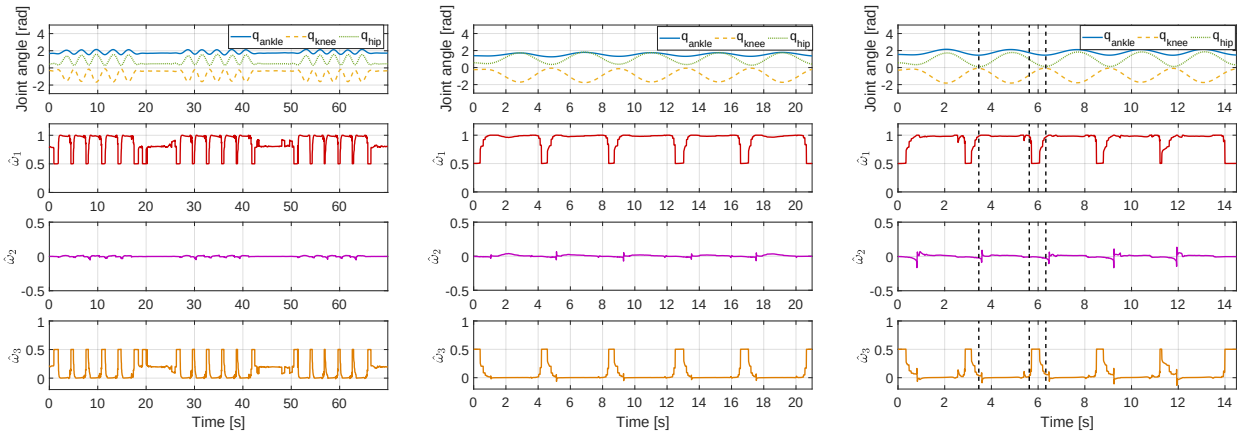
As shown in Fig. 2.16a, the human body is modeled as a 3DOF (ankle-knee-hip joints) fixed-base articulated system. The dynamics of the modeled human body [28] is given by

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{2.108}$$



(a) A repetition of squat motion[28]. (b) Joint trajectory of 15 squats.

Figure 2.16. Squat exercise and a sample trajectory for 15 squats.



(a) 15 squats by Participant 1. (b) 5 squats by Participant 3. (c) 5 squats by Participant 5.

Figure 2.17. Multiphase cost function recovery for three sample participants. Joint trajectory (filtered) of each participant is plotted in first row: (a) 15 squats in 3 sets by Participant 1; (b) one 5-squat set by Participant 3; and (c) one 5-squat set by Participant 5. The corresponding recovery results are shown below respectively, where $\hat{\omega}_1$, $\hat{\omega}_2$, and $\hat{\omega}_3$ are the cost weights for the acceleration ϕ_1, joint jerk ϕ_2, and power ϕ_3 in Table 2.5, respectively.

where $\boldsymbol{q} = [q_{ankle}, q_{knee}, q_{hip}]'$ is the joint angle vector; $M(\boldsymbol{q}) \in \mathbb{R}^{3\times3}$ is the inertia matrix; $C(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{3\times3}$ is the Coriolis matrix; $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^3$ is the gravity vector; and $\boldsymbol{\tau} \in \mathbb{R}^3$ are the torques generated by each joint. The anthropometrics parameters [60] are used in (2.108).

The torques (trajectories) are computed from (2.108). We represent (2.108) in state-space form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ with the state $\boldsymbol{x}$ and input $\boldsymbol{u}$ defined as

$$\boldsymbol{x} = \left[\boldsymbol{q}', \dot{\boldsymbol{q}}'\right]' \quad \text{and} \quad \boldsymbol{u} = \boldsymbol{\tau}, \tag{2.109}$$

respectively, and then discretize it into $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \cdot f(\boldsymbol{x}_k, \boldsymbol{u}_{k+1})$ with the discretization interval $\Delta = 0.01$s (i.e. sampling rate of the motion capture system).

The unit feature vector $\boldsymbol{\phi}$ in this experiment is chosen based on the previous work [28], where the following features in Table 2.5 were demonstrated to play significant roles in human squat motion [28]. Thus, $\boldsymbol{\phi} = [\phi_1, \phi_2, \phi_3]'$.

Table 2.5. The selected features for human motion segmentation [28]

Criterion	Feature function (ϕ_i)
Joint acceleration	$\phi_1 = \sum_{i=1}^{3} \ddot{q}_i^2$
Joint jerk	$\phi_2 = \sum_{i=1}^{3} \dddot{q}_i^2$
Joint power	$\phi_3 = \sum_{i=1}^{3} (\tau_i \dot{q}_i)^2$

q_1, q_2, and q_3 correspond to q_{ankle}, q_{knee}, and q_{hip}, respectively.

Recovery Results

Note that for each participant, all 15 squats are in a single recording and we apply the proposed method on the trajectory without manual segmentation. In Algorithm 2, following the rules given in Section V.A.3, we set $\gamma = 6$ (as described before, the value of γ in practice is always smaller because of the imperfection of union feature set selection); since in each motion set (around 6s) the number of phases is estimated around 10, we set $l_{\max} = 60$ (that is, (6s)/10/(0.01s)).

We use the data from Participant 1 (P1), Participant 3 (P3), and Participant 5 (P5) as examples to demonstrate the recovery results. In Fig. 2.17, the joint trajectories of P1, P3, and P5 are shown in the first row; here, we present the entire motion data (i.e. 3 sets and a total of 15 squats) for P1 and only one set (5 squats) for P3 and P5 to show both overall and

Table 2.6. Motion segmentation and multiphase cost function recovery for all participants. Active-squat phases and between-squats phases are segmented by ω_{th}, and the corresponding segmentation accuracy is computed. Using successful segmentations, the average cost weights for both phases are computed. The results by the KKT method [28] are also compared.

Participant	Average $\hat{\boldsymbol{\omega}}$ for active-squat phases		Average $\hat{\boldsymbol{\omega}}$ for between-squats phases		Segmentation accuracy [%]		
	$\omega_{th} = 0.8$	$\omega_{th} = 0.9$	$\omega_{th} = 0.8$	$\omega_{th} = 0.9$	$\omega_{th} = 0.8$	$\omega_{th} = 0.9$	KKT method [28]
1	[0.98, 0.00, 0.02]	[0.98, 0.00, 0.01]	[0.55, −0.01, 0.45]	[0.61, −0.01, 0.40]	96.88%	96.88%	89.54%
2	[0.97, 0.00, 0.03]	[0.98, 0.00, 0.02]	[0.63, −0.01, 0.38]	[0.69, −0.01, 0.32]	100.0%	100.0%	83.51%
3	[0.98, 0.00 , 0.02]	[0.99, 0.01, 0.00]	[0.56, −0.01, 0.44]	[0.63, −0.01, 0.38]	96.67%	96.67%	85.80%
4	[0.96, 0.00, 0.03]	[0.98, 0.01, 0.01]	[0.62, −0.01, 0.38]	[0.70, −0.01, 0.31]	94.44%	100.0%	67.62%
5	[0.98, 0.00, 0.02]	[0.98, 0.00, 0.01]	[0.64, −0.01, 0.37]	[0.65, −0.01, 0.36]	92.89%	93.33%	76.39%
6	[0.98, 0.01, 0.02]	[0.98, 0.01, 0.01]	[0.69, −0.01, 0.31]	[0.73, −0.01, 0.27]	91.15%	90.00%	89.05%

local details of the recovery results. Corresponding to the motion data, the recovered cost weights $\hat{\boldsymbol{\omega}}(t)$ are presented in the panels below; here, cost weight $\hat{\omega}_1$, $\hat{\omega}_2$, and $\hat{\omega}_3$ correspond to ϕ_1, ϕ_2, and ϕ_3 in Table 2.5, respectively. We have the following observations:

(a) Overall, Fig. 2.17a shows a reliable multiphase cost function recovery performance. During each squat repetition (i.e. standing-squatting-standing in Fig. 2.16a), the cost weights $\hat{\boldsymbol{\omega}}(t)$ remain at the value around $[1, 0, 0]$, which indicates that one squat belongs to the same phase in terms of sharing the same cost function. Between two squats where a participant is near (approaching) standing position, the weights change to (around) $[0.6, 0, 0.4]$, indicating that the participants switch to a different control strategy after finishing one squat but before starting the next. Fig. 2.17a also shows that the cost weights in between two motion sets (where the participants are in the standing position) are around $[0.8, 0, 0.2]$.

(b) Recovery results in Fig. 2.17b and Fig. 2.17c show in more detail the changes of the cost weights within a 5-squat set. Below, we use Fig. 2.17c for analysis. As labeled by the dotted black (vertical) lines, we divide a squat repetition into two motion phases according to different cost functions used:

- *Active-Squat (AS)*: between the first and second dotted lines, during which the participant is flexing hips and knees (to squatting position) and then extending the hips and knees. The recovered results show that the control objective of this phase is to minimize the joint acceleration ϕ_1 (as both $\hat{\omega}_2$ and $\hat{\omega}_3$ are near zeros).

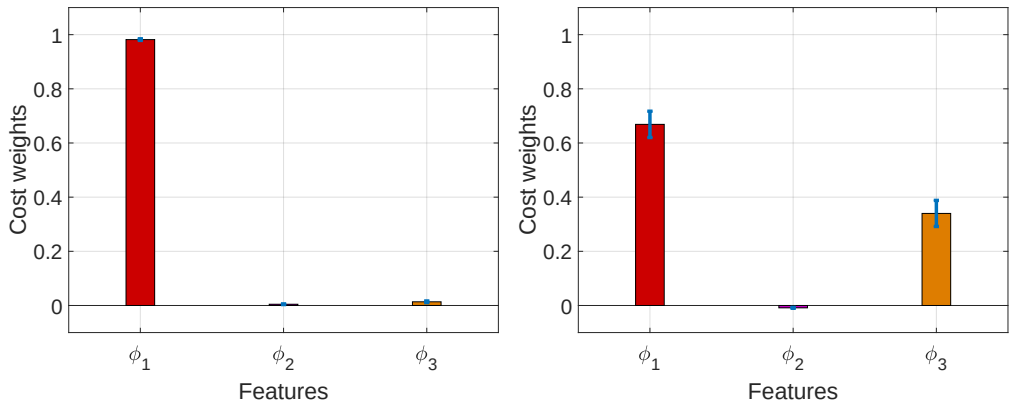
- *Between-Squats (BS)*: between the second and third dotted lines, during which the participant is finishing the hip and knee extension from the previous active squat and then preparing for next one. The cost function to be minimized for this phase is (approximately) $0.6\phi_1 + 0.4\phi_3$.

Segmentation Results

In order to automate the segmentation of the active-squat phase and the between-squats phase in each motion set, we define a *segmentation threshold* ω_{th} for $\hat{\omega}_1(t)$ (the most influential weight), and then the segmentation is performed using the following rules: if $\hat{\omega}_1(t) > \omega_{th}$; the current phase is classified as active-squat; otherwise, as between-squats. We evaluate the segmentation accuracy by $\Big(0.5 \times (\frac{T_{AS}}{T_{AS}+F_{BS}} + \frac{T_{BS}}{T_{BS}+F_{AS}})\Big)$[28], where T_{AS} is the count of the cases where a true active-squat phase is segmented into active-squat (True Positive); T_{BS} when a true between-squats phase is segmented into between-squats (True Negative), F_{AS} when a true active-squat phase is classified as between-squats (False Positive), and F_{BS} when a true between-squats phase is segmented into active-squat (False Negative).

The segmentation results for all participants are summarized in Table 2.6. Here, two thresholds $\omega_{th} = 0.8$ and $\omega_{th} = 0.9$ are used, and the average cost weights for active-squat and between-squats are computed based on all successful segmentations. It can be seen that the proposed method demonstrates a high reliability and accuracy in segmenting different motion phases. The difference in the average cost weights of between-squats phase for two thresholds is due to the fact that the actual period of a between-squats phase is small (Fig. 2.17c), thus is more likely to be affected by segmentation threshold values.

Using $\omega_{th} = 0.9$, we summarize the average cost weights for active-squat and for between-squats over all participants in Fig. 2.18a and 2.18b, respectively. It shows that all participants adopt a similar control policy in squat exercise: during active squat, participants focus on minimizing the joint acceleration, while in between squats, they adopt a balanced control policy minimizing both joint acceleration and power. This finding is consistent with previous human motion studies [22, 28, 36].



(a) Active-squat phase. (b) Between-squats phase.

Figure 2.18. Recovery results over all participants for active-squat and between-squats phases. Bars denote the mean cost weights, and top line segments denote the standard deviation.

For comparison, we also perform the motion segmentation using the KKT method [28], as shown in Table 2.5. The proposed method can achieve a segmentation accuracy above 90%, which is higher than that of the KKT method [28] with average accuracy 81.99%.

Result Validation

To validate the recovery and segmentation results, we simulate the trajectory of each segmented phase by solving the optimal control problem based on the recovered cost functions. Considering the consistency of the recovery among different squat repetitions (Fig. 2.17) and different participants (Fig. 2.18), we just use one squat repetition of a sample participant for illustration. We consider one squat repetition performed by Participant 5 as labeled by the black dotted lines in Fig. 2.17c. Under segmentation threshold $\omega_{th} = 0.9$, the active-squat phase is from time 3.46s to 5.62s with the average cost weights $[0.99, 0.00, 0.00]$ (by averaging $\hat{\boldsymbol{\omega}}(t)$ within this active-squat phase), and the between-squats is from 5.62s to 6.33s with average cost weights $[0.62, -0.01, 0.39]$. We solve the optimal control problem using these cost functions for both phases [51] and plot the results in Fig. 2.19. The results show that the simulated trajectory using the recovered cost functions fits well the real data, indicating the validity of the recovered cost functions in reproducing squat motion.

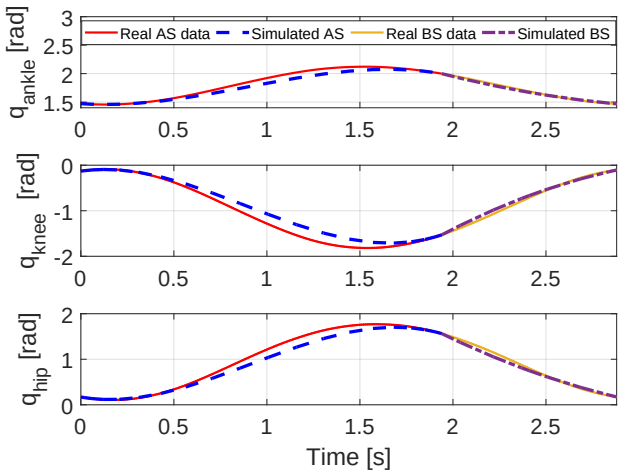


Figure 2.19. Simulated trajectory using the recovered multiphase cost functions. Solid lines are real motion data (second squat repetition in Fig. 2.17c): red for the active-squat phase and yellow for the between-squats phase. Dotted lines are the simulated motion: blue for the active-squat phase and brown for the between-squats phase.

2.6.2 Human Motion Prediction

In this application, we will develop an *on-the-fly* human motion prediction approach based on the incremental inverse optimal control method developed in Section 2.3. Given an observed ongoing human motion, the approach recovers the cost function from early observations of the human motion and then predicts the remaining motion based on the recovered cost function. The whole framework can be directly applied to predict a single ongoing motion, as opposed to [27, 61, 62] which requires a dedicated offline process to learn a prediction model a priori.

On-the-fly Human Motion Prediction Method

As shown in Fig. 2.20, our human motion prediction framework consists of two stages. The first stage is to apply the incremental IOC algorithm (Algorithm 1) to recover the cost function (the unknown weights $\boldsymbol{\omega}$ of a given feature vector $\boldsymbol{\phi} \in \mathbb{R}^{|\mathcal{F}|}$) from the early observations of the ongoing motion $\boldsymbol{\xi}_{0:T_c}$; here the observation length T_c is determined by the

minimal observation length, i.e., $T_c = l_{\min}(0)$. The second stage is to predict the remaining trajectory using an optimal control planner with the obtained cost function, that is,

$$\begin{aligned} \boldsymbol{\xi}_{T_c:T} = \arg \min_{\{\boldsymbol{x}_{T_c:T}, \boldsymbol{u}_{T_c:T}\}} & \sum_{k=T_c}^{T} \boldsymbol{\omega}' \boldsymbol{\phi}(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \text{s.t.} \quad & \dot{\boldsymbol{x}}_k = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \quad \text{given} \quad \boldsymbol{x}_{T_c}, \\ & \boldsymbol{x}_T = \boldsymbol{x}_{\text{goal}}, \end{aligned} \tag{2.110}$$

where we consider the total motion horizon T and goal state $\boldsymbol{x}_{\text{goal}}$ are given.

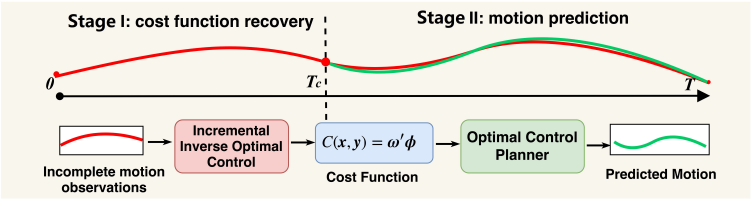


Figure 2.20. The paradigm of the on-the-fly human motion predication.

Data Collection

We still use the human squat exercise data, as shown in Fig. 2.16a. The squat exercise data was collected from 10 (6 M, 4 F, $\mu_{age} = 23.8$) participants [63] from the University of Waterloo. All participants were healthy and had no lower body injuries for six months prior to the data collection. The experiment was approved by the University of Waterloo Research Ethics Board, and signed consent was obtained from all participants. Prior to the data collection, the participants were verbally instructed on how to conduct exercise. Each participant conducted squat exercise around 8 repetitions. The inverse kinematics computing the joint angles from the Cartesian marker data were obtained using a Kalman filter estimator [64]. The joint angles in each exercise are defined in Fig. 2.16a. The initially obtained joint sequence for each motion was fitted using 5^{th} order polynomials [28], which not only suppressed capture noise but also provided smooth first-derivative (joint angular velocity) and second-derivative (joint acceleration) trajectories. The trajectory sample time for all

motions is 0.1s and the joint angular velocity and acceleration are obtained by numerical differentiation.

Body Dynamics Model and Feature Selection

As shown in Fig. 2.16a, the human body is modeled as a 3 DoF (ankle-knee-hip joints) fixed-base articulated system, with the dynamics model in (2.108) and state and input defined in (2.109). The anthropometrics parameters [65] for each participant are generated according to [60]. Since inverse optimal control requires the control inputs, the joint torques in each motion are calculated via the body (inverse) dynamics. The features used for the motion prediction is given in Table 2.7.

Table 2.7. The selected features for human motion prediction.

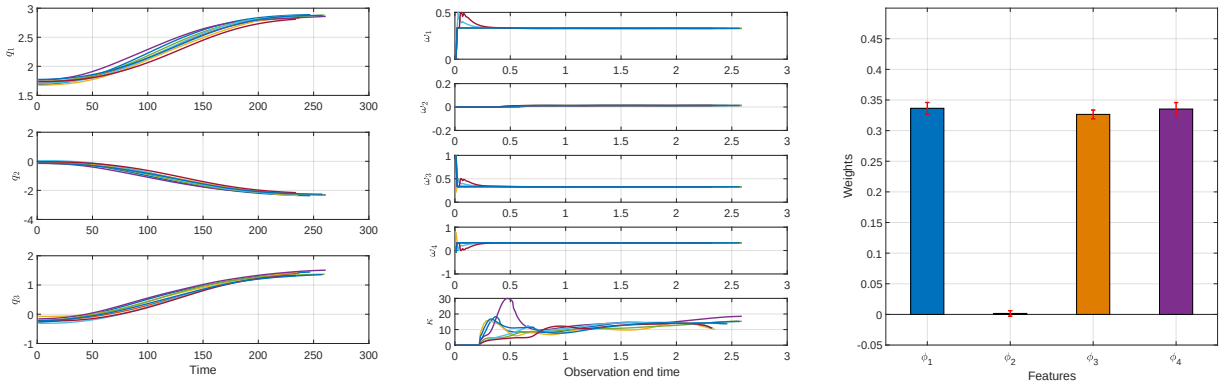
Criterion	Feature function (ϕ_i)
Joint acceleration	$\phi_1 = \sum_{k=1}^{T} \sum_{i=1}^{n} \ddot{q}_{i,k}^2$
Joint jerk	$\phi_2 = \sum_{k=1}^{T} \sum_{i=1}^{n} \dddot{q}_{i,k}^2$
End-effector acceleration	$\phi_3 = \sum_{k=1}^{T} (\ddot{x}_k^2 + \ddot{y}_k^2)$
Power	$\phi_4 = \sum_{k=1}^{T} \sum_{i=1}^{n} (\tau_i \dot{q}_i)^2$

q_1, q_2, and $q_{n=3}$ correspond to q_{ankle}, q_{knee}, and q_{hip}, respectively.

Cost Function Recovery Stage

We choose Participant S1's squat motion data to illustrate the cost function recovery. The joint angle trajectories of 8 repetitions of S1's squat motion are plotted in Fig. 2.21a. For each trajectory in Fig. 2.21a, we perform incremental IOC by increasing the observation length l from 1 to T while solving the weights via (2.53) and (2.54). The incremental recovery results for all trajectories in Fig. 2.21a are shown in Fig. 2.21b, where κ is the rank index (2.51) for the corresponding recovery matrix. From the incremental results in Fig. 2.21b, we note that the weights can be successfully recovered when the observation length reaches 16% of the entire horizon when $\kappa \geq 4$. Thus, we set $\gamma = 4$ in Algorithm 1 to determine the minimal required observation length $l_{\min}(0) = T_c$ throughout the following experiments.

For all trajectories in Fig. 2.21a, the recovered weights recovered by applying Algorithm 1 are summarized in Fig. 2.21c, where the bar bars denote their mean values and the red line segments represent the standard deviations. From Fig. 2.21c, the recovered weights show a good consistency over different repetitions.



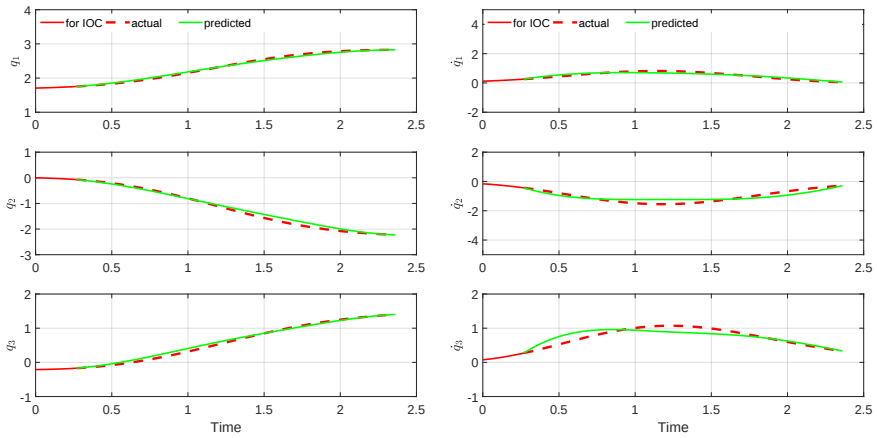
(a) Squat trajectories of S1 (b) Incremental IOC results (c) IOC result summary

Figure 2.21. IOC results for one participant: (a) the trajectories of participant S1's squat trajectories over 8 repetitions; (b) incremental IOC results, and κ is the rank index in (2.51); (c) summary of the recovered weights by applying Algorithm 1 with $\gamma = 4$: bars denote the mean values, top line segments denote the standard deviation;

Motion Prediction Stage

Using the cost function obtained at the cost function recovery stage, we predict the remaining motion trajectory by solving the optimal control problem in (2.110) while assuming that the time horizon T and the goal state $\boldsymbol{x}_{\text{goal}}$ are known. In our experiment case, the goal state $\boldsymbol{x}_{\text{goal}} = [\pi/2, 0, 0]$ (standing status), and $T = 2 \approx 2.4$s.

We take one trajectory in Fig.2.21a for illustration. The minimal observation length for the cost function recovery stage is $T_c = 0.12$s (given by Algorithm 1 with $\gamma = 4$), and the recovered weights are $\boldsymbol{\omega} = [0.33, 0.00, 0.35, 0.32]'$. By solving (2.110), we generate the trajectory from T_c to T and plot the results (green lines) in Fig. 2.22a and 2.22b. The actual trajectory for the remaining motion is shown in red dash lines. The prediction results show that the weights recovered by Algorithm 1 in the cost function recovery stage can achieve



(a) Joint angle. (b) Joint velocity.

Figure 2.22. Motion prediction for one trajectory of Participant S1. The red solid lines denote the trajectory segment used for IOC in the cost function recovery stage; the red dash lines denote the human actual trajectory of the remaining motion; the green lines are the predicted trajectory based on the recovered cost function.

high accuracy of predicting the remaining human motion (the prediction error in (2.111) is 0.25).

Next, we will investigate the performance of the human prediction approach across different participants. We use the squat motion data from the 10 participants with each conducting for around 8 repetitions. For each motion trajectory of each participant, Algorithm 1 is used to recovered the cost function weights with $\gamma = 4$. Then based on the recovered cost function, the trajectory data after the recovery convergence point is predicted by solving the optimal control in (2.110). To evaluate the quality of a predicted trajectory with respect to actual motion, we define the prediction error

$$e = \frac{\sum_{k=T_c}^{T} \|\boldsymbol{x}_k^{\text{pred}} - \boldsymbol{x}_k^{\text{actual}}\|}{T - T_c}, \tag{2.111}$$

where T_c is the recovery convergence point; $\|\cdot\|$ is the l_2 norm; $\boldsymbol{x}_k^{\text{pred}}$ and $\boldsymbol{x}_k^{\text{actual}}$ are the predicted and captured states at the time k, respectively. Table 2.8 summarizes results for each participant. From Table 2.8, one can observe that (i) on average, the cost function recovery stage needs 19.22% (i.e., T_c/T) of the trajectory data to recover the weights; (ii)

Table 2.8. Prediction results for human squat motions

Par.	ω_1	ω_2	ω_3	ω_4	average T_c/T	average e
S1	0.336±0.010	0.002±0.004	0.327±0.007	0.335±0.012	13.65%	0.264
S2	0.301±0.091	0.000±0.000	0.347±0.038	0.352±0.053	17.55%	0.306
S3	0.332±0.004	0.000±0.000	0.337±0.002	0.331±0.005	19.73%	0.317
S4	0.333±0.000	0.000±0.000	0.333±0.000	0.333±0.000	19.29%	0.272
S5	0.294±0.006	0.000±0.000	0.340±0.009	0.367±0.123	24.58%	0.268
S6	0.282±0.124	0.000±0.000	0.388±0.111	0.331±0.183	18.24%	0.511
S7	0.331±0.004	0.000±0.000	0.335±0.003	0.335±0.002	15.31%	0.407
S8	0.335±0.004	0.000±0.000	0.335±0.004	0.329±0.007	20.48%	0.455
S9	0.330±0.010	0.000±0.000	0.335±0.005	0.335±0.005	24.15%	0.312
S10	0.339±0.006	0.000±0.000	0.339±0.006	0.323±0.012	19.09%	0.176

Note that the weights in the table are computed using the minimal observation length $T_c = l_{\min}(0)$.

the recovered cost function weights have small variation among different participants; and (iii) on average, the motion prediction stage has the prediction error of 0.329. Those results demonstrate the efficacy of the on-the-fly motion prediction approach.

2.7 Conclusions

In this chapter, we have developed a series of new theories and approaches to addressing the technical gaps in existing IOC techniques towards efficient objective learning. First, we introduce the concept of the recovery matrix and present its proprieties and relationship to solving IOC. Based on the recovery matrix, we present the incremental IOC method to enable learning an objective function by finding the minimal observations. Next, we present the IOC method for learning multi-phase objective functions. Then, we develop a distribution IOC approach to enable efficient objective learning in the case where both data and computation are distributed. Finally, we present some novel applications of the developed IOC techniques, including human motion segmentation and on-the-fly human motion prediction.

3. PONTRYAGIN DIFFERENTIABLE PROGRAMMING

In Chapter 1, we characterize an autonomous robot as a decision-making system (agent) consisting of the aspects of task objective, dynamics, and policies, as in Fig. 1.1. Many machine learning and control topics have been focused on how to automate the programming of different aspects of a decision-making system. Recently, both fields have begun to explore the complementary benefits of each other: control theory may provide abundant models and structures that allow for efficient or certificated algorithms for high-dimensional tasks, while learning enables to obtain these models from data, which are otherwise not readily attainable via classic control tools. Examples that enjoy both benefits include model-based reinforcement learning [66, 67], where dynamics models are used for sample efficiency; and Koopman-operator control [68, 69], where via learning, nonlinear systems are lifted to a linear observable space to facilitate control design. Inspired by those, this chapter aims to exploit the advantage of integrating learning and control to develop a Pontryagin Differentiable Programming (PDP) methodology – a unified end-to-end framework capable of efficiently solving a wide range of learning and control problems. The content of this chapter appears in [35]. The code and experiments developed for this chapter is available at https://github.com/wanxinjin/Pontryagin-Differentiable-Programming.

3.1 Introduction and Background

While both machine learning and control communities are working towards higher robot autonomy, method developments in both fields seem largely disjoint, as listed below.

Table 3.1. Topic connections between control theory and machine learning

UNKNOWNS IN A SYSTEM	MACHINE LEARNING	CONTROL METHODS
Dynamics $\boldsymbol{x}_{t+1}=\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{u}_t)$	Markov decision processes	System identification
Policy $\boldsymbol{u}_t = \boldsymbol{\pi}_{\boldsymbol{\theta}}(t, \boldsymbol{x}_t)$	Reinforcement learning (RL)	Optimal control (OC)
Control objective $J=\sum_t c_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{u}_t)$	Inverse RL	Inverse OC

Learning Dynamics. This is usually referred as to as system identification in the control field, which typically considers linear systems represented by transfer functions [10].

For nonlinear systems, the Koopman theory [70] provides a way to lift states to a linear observable space [68, 71]. In machine learning, dynamics is characterized by Markov decision proceses and implemented using linear regression [72], observation-transition modeling [73], latent-space modeling [74], (deep) neural networks [75], Gaussian process [5], transition graphs [76], etc. Most of these methods need to trade off between data efficiency and long-term prediction accuracy. Towards achieving both, physically-informed learning [77, 78, 79, 80] injects physics laws into learning models, but they mainly focus on mechanical systems. Recently, a trend of work starts to use dynamical systems to explain (deep) neural networks, and some new algorithms [9, 81, 82, 83, 84, 85, 86, 87] have been proposed.

Learning Optimal Polices. In machine learning, it relates to reinforcement learning (RL). Model-free RL provides a general-purpose framework to learn policies directly from interacting with environments [88, 89, 90], but usually suffers from high data complexity. Model-based RL [91, 92] focuses on first learning a dynamics model from experience and then integrating such model to policy improvement. Specifically, the learned model can be used for generating (simulating) the experience data [93, 94], performing back-propagation through time [5], or testing before deployment. Many studies [5, 66, 74, 95, 96, 97, 98] have shown that model-based RL is generally more data- and computation- efficient than model-free RL. Some challenges are still not well-addressed in existing model-based RL techniques [91]. For example, how to efficiently leverage imperfect models [6], and how to maximize the joint benefit by combining policy learning and motion planning [97, 99], where a policy has the advantage of execution coherence and fast deployment while the trajectory planning has the competence of adaption to unseen or future situations.

The counterpart topic in the control field is optimal control (OC), which is more concerned with characterizing optimal trajectories in presence of dynamics models. As in RL, the main strategy for OC is based on dynamic programming, and many valued-based methods are available, such as HJB [100], differential dynamical programming (DDP) [101] (by quadratizing dynamics and value function), and iterative linear quadratic regulator (iLQR) [7] (by linearizing dynamics and quadratizing value function). The second strategy to solve OC is based on the Pontryagin's Maximum/Minimal Principle (PMP) [44]. Derived from calculus of variations, PMP can be thought of as optimizing directly over trajectories, thus

avoiding solving for value functions. Popular methods in this vein include shooting methods [102] and collocation methods [51]. OC methods based on PMP are essentially *open loop* control and thus susceptible to model errors or disturbances in deployment. To address these, model predictive control (MPC) [103] generates controls given the system current state by repeatedly solving an OC problem over a finite prediction horizon (only the first optimal input is executed), leading to a *closed-loop* control strategy. Although MPC has dominated across many industrial applications [104], developing fast MPC implementations is still an active research direction [105].

Learning Control Objective Functions. In machine learning, this relates to inverse reinforcement learning (IRL), whose goal is to find a control objective function to explain the given optimal demonstrations. The unknown objective function is typically parameterized as a weighted sum of features [29, 32, 33]. Strategies to learn the unknown weights include feature matching [32] (matching the feature values between demonstrations and reproduced trajectories), maximum entropy [29] (finding a trajectory distribution of maximum entropy subject to empirical feature values), and maximum margin [33] (maximizing the margin of objective values between demonstrations and reproduced trajectories). The learning update in the above IRL methods is preformed on a selected feature space by taking advantage of linearity in feature weights, and thus cannot be directly applied to learning objective functions that are nonlinear in parameters. The counterpart topic in the control field is inverse optimal control (IOC) [11, 12, 22, 40]. With knowledge of system dynamics models, IOC focuses on more efficient learning paradigms. For example, by minimizing the violation of optimality conditions by the observed demonstration data, [11, 12, 34, 40] directly compute feature weights without repetitively solving the OC problems. Despite the efficiency, minimizing optimality violation does not directly assure the closeness between the final reproduced trajectory and the observed demonstrations.

Unified Perspective to Look at Learning Dynamics/Policy/Objective. Consider a decision-making agent that consists of aspects of dynamics, control policy, and control objective function. In a unified perspective, learning dynamics, policies, or control objective functions can be viewed as *the instantiations of the same learning problem* but with (i)

different unknown (parameterized) aspects and (ii) different losses. For example, for learning dynamics, a differential/difference equation is parameterized and the loss function can be defined as the prediction error between the model output and physical data; for learning policies, the unknown parameters are in a feedback policy and the loss function is just the control objective function; and for learning control objective, the control objective function is parameterized and the loss function can be the discrepancy between the reproduced optimal trajectory and the observed demonstrations. This unified perspective will motivate the formulation of PDP in Section 3.4.

3.2 Contributions of PDP

This chapter develops an end-to-end learning framework, named as Pontryagin Differentiable Programming (PDP), that is flexible enough to be customized for different learning and control tasks and capable enough to efficiently solve high-dimensional and continuous-space problems. The proposed PDP framework borrows the idea of 'end-to-end' learning [106] and chooses to optimize a loss function directly with respect to the tunable parameters in the aspect(s) of an optimal control system, including the dynamics, policy, or/and control objective function. The key contribution of the PDP is that we inject the optimal control theory as an inductive bias into the learning process to expedite the learning efficiency. Specifically, the PDP framework centers around the system's trajectory and *differentiates the trajectory through Pontryagin Maximum/Minimum Principle*, and this allows us to obtain the analytical derivative of the trajectory with respect to the tunable parameters, a key quantity for end-to-end learning of (neural) dynamics, (neural) policies, and (neural) control objective functions. Furthermore, we introduce an *auxiliary control system* in backward pass of the PDP framework, and its output trajectory is exactly the derivative of the trajectory with respect to the parameters, which can be iteratively solved using standard control tools.

The PDP framework can be customized for solving different types of learning and control problems. For each specialized application, we emphasize the advantage of PDP over existing methods in Section 3.1 as below.

- PDP has a special mode for learning dynamics. Compared to the system identification techniques in control, PDP allows for learning nonlinear dynamics models—either physical dynamics with unknown parameters or neural ordinary difference equations; and second, compared to existing machine learning techniques, PDP integrates the structures of optimal control theory into learning process, which leads to improved data- and computation- efficiency, as shown later in Section 3.7.

- PDP has a special mode for model-based policy optimization tasks. Such a mode can be viewed as a complement to classic open-loop OC methods, because, although derived from PMP (which is an open loop control strategy), PDP is able to learn a *feedback/closed-loop* control policy. Depending on the specific policy parameterization, PDP can also be used for motion planning. All these features will provide new perspectives for model-based RL or MPC control.

- PDP has a special mode for IOC/IRL tasks. PDP addresses the technical gaps of existing IOC/IRL techniques (see Section 3.1) and have the following advantages. First, it enables to learn complex control objective functions, e.g., neural objective functions. Second, it directly minimizes the distance between the reproduced trajectory and given demonstrations; thus, even though the given demonstrations are sub-optimal, PDP can still find a control objective function such that the reproduced trajectory has the closest distance to the demonstrations. Third, compared to existing IOC/IRL techniques, PDP integrates the structure of optimal control theory into learning process, thus is more data- and computation- efficient, especially for handling high-dimensional tasks, as shown later in Section 3.7.

3.3 Related Work

3.3.1 End-to-End Differentiable Learning

Two lines of recent work in the machine learning field are related to PDP. One is the recent work [107, 108, 109, 110, 111] that seeks to replace a layer within a deep neural

network by an *argmin layer*, in order to capture the information flow characterized by a solution of an optimization. Similar to the idea of PDP, these methods differentiate the argmin layer through KKT conditions, but they are not directly applicable to dynamical systems since these methods mainly deal with static optimization without time evolution. The second line is the recent RL development [112, 113, 114, 115] that embeds an implicit planner within a policy. The idea is analogous to MPC, because predictive OC systems (i.e., embedded planner) leads to better adaption to unseen situations. The key problem in these methods is to learn an OC system, which is similar to our formulation. We details their difference from PDP below.

Path Integral Network. [112] and [113] develop a differentiable end-to-end framework to learn path-integral optimal control systems [116], which are a special category of optimal control systems—dynamics is affine in control input and control objective is quadratic in control input. This framework adopts the 'unrolling' strategy, which means that the forward pass of solving optimal control is extended as a graph of multiple steps of applying gradient descent, and the solution of optimal control is considered as the output of the final step of the gradient descent operations. The advantage of this unrolling (gradient descent) computational graph is that it can immediately apply automatic differentiation techniques such as TensorFlow [117] to obtain the gradient in backward pass, but it needs to store and traverse all intermediate results throughout the gradient descent steps, which can be both memory- and computationally- expensive. We have provided its complexity comparison with PDP later in Section 3.8.1 (see Table 3.3).

Universal Planning Network. In [115], the authors develop an end-to-end imitation learning framework consisting of two layers: the inner layer is a planner, which is formulated as an optimal control system in a latent space and is solved by gradient descent, and an outer layer to minimize the imitation loss between the output of inner layer and expert demonstrations. This framework is also based on the 'unrolling' strategy. Specifically, the inner planning layer using gradient descent is considered as a large computation graph, which chains together the sub-graphs of each step of gradient descent. In the backward pass, the gradient derived from the outer layer back-propagates through the entire computation graph. Similar to the previous Path Integral Network, this unrolled learning strategy will

incur higher memory and computation costs in implementation. Please find its complexity analysis in Table 3.3 in Section 3.8.1

Different from the above 'unrolling' methods [112, 113, 115, 118], PDP handles the learning of optimal control systems in a 'direct and compact' manner. Specifically, in forward pass, PDP only obtains and stores the final solution of the optimal control system and does not care about the intermediate process of how such solution is obtained. Thus, the forward pass of PDP accepts any external optimal control solver such as CasADi [50]. Using the solution in the forward pass, PDP then automatically builds the auxiliary control system, based on which, the exact *analytical* gradient is solved efficiently in backward pass. Such features guarantee that the complexity of the PDP framework is only linearly scaled up to the time horizon of the system, which is more efficient than the above 'unrolling' methods. See the detailed analysis in Section 3.8.1.

Differentiable MPC. [114] develops an end-to-end differentiable MPC framework to jointly learn the system dynamics model and control objective function of an optimal control system. In forward pass, it uses iLQR [7] to solve optimal control and find a fixed point, and then approximates the optimal control system by a LQR at the fixed point. In backward pass, the gradient is obtain by differentiating the LQR approximation. This process may have two limitations: first, since the differentiation in backward pass is conducted on the LQR approximation instead of on the original system, the obtained gradient thus may not be accurate due to approximation discrepancy; and second, computing the gradient of the LQR requires the inverse of a coefficient matrix of size $T \times T$ with T the time horizon, which can cause high computational cost when handling systems with longer time horizon T.

Compared to differentiable MPC, the first advantage of PDP is that the differentiation in backward pass is directly performed on the parameterized optimal control system (by differentiating through Pontryagin Maximum/Minimum Principle). Second, we develop the auxiliary control system in backward pass, whose trajectory is exactly the gradient of the system trajectory. The gradient then is iteratively solved using control tools with the complexity only $\mathcal{O}(T)$. Those proposed techniques enables the PDP to have significant advantage in computational efficiency over differentiable MPC. To illustrate this, we will later compare the algorithm complexity between PDP and differentiable MPC in Section 3.8.1.

3.3.2 Adaptive Control

The idea of PDP of automatically tuning a control system for a specific task is similar to the idea of adaptive control techniques.

The general adaptive control design is to find an adaption rule for a system controller to cope with the variations in environmental conditions and system itself [119]. One popular adaptive control technique is the Model-Reference Adaptive Control (MRAC) [120, 121], for which the design goal is to force the controlled system to behave like a reference model by adjusting the controller parameters. MRAC measures the error between the actual output of the system and that of the reference model and then feed the error back to an adaption law to modify the controller parameters [122]. The adaption law here can be the MIT rule [123, 124], where the idea is to decrease the distance between the actual output of the system and that of the reference model at *current time*, or the Lyapunov Rule [125], which aims to make the error dynamics stable and converge to zeros. The above adaptive control techniques are fundamentally different from PDP in both problem formulation and methodologies, as detailed below.

1) Different update mechanisms. Adaptive control is an online adaption mechanism in a sense that the adjustment of the control parameters and the execution of the resulting controlled system are synchronized and coupled; the goal of adaptive control is to guarantee that the output error between the actual system and reference model converge to zeros as the system execution time goes to infinity. Instead, PDP is an iterative learning mechanism—PDP tunes system parameters by executing the optimal control system *repetitively*, and the system parameters are updated only after the finish of each execution (i.e., each pass of time horizon) and before the next one. At each update, PDP improves the system performance based on the previous execution. Thus, PDP is not an online parameter tuning framework.

2) Different design goal. The goal of adaptive control is to guarantee the asymptotic stability of the tracking error of the controlled system as it executes, and such stability does not guarantee that the cumulative error between the system trajectory and reference model trajectory over entire time horizon is minimal. Instead, the formulation of PDP is to minimize the loss defined on the system trajectory over entire time horizon. Thus, adaptive

control is more concerned about the 'stability' of the system tuning, while PDP focused on the 'optimality' of system tuning.

3) Different tunable aspects. Adaptive control only focuses on tuning controller parameters in a general (linear) closed-loop system, in order to achieve the desired performance of the closed-loop system. The formulation of PDP focuses on tuning a general optimal control system, where all aspects of the optimal control system can be set tunable, including system dynamics, control policies, and control objective functions. Therefore, in addition to tuning control policies, PDP can be applied to tuning system dynamics (system identification) and control objective function (inverse optimal control), which cannot be achieved using adaptive control techniques.

3.4 PDP Problem Formulation

We begin with formulating a base problem and then discuss how to accommodate the base problem to specific applications. Consider a class of optimal control systems $\boldsymbol{\Sigma}(\boldsymbol{\theta})$, which is parameterized by a tunable $\boldsymbol{\theta} \in \mathbb{R}^r$ in both dynamics and control (cost) objective function:

$$\boldsymbol{\Sigma}(\boldsymbol{\theta}): \quad \begin{array}{rl} \text{dynamics:} & \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\theta}) \quad \text{with} \quad \text{given} \quad \boldsymbol{x}_0, \\ \text{control objective:} & J(\boldsymbol{\theta}) = \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\theta}) + h(\boldsymbol{x}_T, \boldsymbol{\theta}). \end{array} \tag{3.1}$$

Here, $\boldsymbol{x}_t \in \mathbb{R}^n$ is the system state; $\boldsymbol{u}_t \in \mathbb{R}^m$ is the control input; $\boldsymbol{f}: \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r \mapsto \mathbb{R}^n$ is the dynamics model, which is assumed to be twice-differentiable; $t = 0, 1, \cdots, T$ is the time step with T being the time horizon; and $J(\boldsymbol{\theta})$ is the control objective function with $c_t: \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r \mapsto \mathbb{R}$ and $h: \mathbb{R}^n \times \mathbb{R}^r \mapsto \mathbb{R}$ denoting the stage/running and final costs, respectively, both of which are twice-differentiable. For a choice of $\boldsymbol{\theta}$, $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ will produce a trajectory of state-inputs:

$$\boldsymbol{\xi}_{\boldsymbol{\theta}} = \{\boldsymbol{x}^{\boldsymbol{\theta}}_{0:T}, \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T-1}\} \in \arg \begin{array}{ll} \min_{\{\boldsymbol{x}_{0:T}, \boldsymbol{u}_{0:T\text{-}1}\}} & J(\boldsymbol{\theta}) \\ \text{subject to} & \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\theta}) \text{ for all } t \text{ given } \boldsymbol{x}_0 \end{array}, \tag{3.2}$$

that is, $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ optimizes $J(\boldsymbol{\theta})$ subject to the dynamics constraint $\boldsymbol{f}(\boldsymbol{\theta})$. For many applications (we will show next), one evaluates the above $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ using a scalar-valued differentiable loss

$L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$. Then, the **problem of interest** is to tune the parameter $\boldsymbol{\theta}$, such that $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ has the minimal loss:

$$\min_{\boldsymbol{\theta}} L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) \quad \text{subject to} \quad \boldsymbol{\xi}_{\boldsymbol{\theta}} \text{ is in (3.2)}. \tag{3.3}$$

Under the above base formulation, for a specific learning or control task, one only needs to accordingly change precise details of $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ and define a specific loss function $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$, as we discuss below.

IRL/IOC Mode. Suppose that we are given optimal demonstrations $\boldsymbol{\xi}^{\mathrm{d}} = \{\boldsymbol{x}^{\mathrm{d}}_{0:T}, \boldsymbol{u}^{\mathrm{d}}_{0:T-1}\}$ of an expert optimal control system. We seek to learn the expert's dynamics and control objective function from $\boldsymbol{\xi}^{\mathrm{d}}$. To this end, we use $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.1) to represent the expert, and define the loss in (3.3) as

$$L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) = l(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\xi}^{\mathrm{d}}), \tag{3.4}$$

where l is a scalar function that penalizes the inconsistency of $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ with $\boldsymbol{\xi}^{\mathrm{d}}$, e.g., $l(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\xi}^{\mathrm{d}}) = \|\boldsymbol{\xi}_{\boldsymbol{\theta}} - \boldsymbol{\xi}^{\mathrm{d}}\|^2$. By solving (3.3) with (3.4), we can obtain a $\boldsymbol{\Sigma}(\boldsymbol{\theta}^*)$ whose trajectory is consistent with the observed demonstrations. It should be noted that even if the demonstrations $\boldsymbol{\xi}^{\mathrm{d}}$ significantly deviate from the optimal ones, the above formulation still finds the 'best' control objective function (and dynamics) within the parameterized set $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ such that its reproduced $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ in (3.2) has the *minimal distance* to $\boldsymbol{\xi}^{\mathrm{d}}$.

SysID Mode. Suppose that we are given data $\boldsymbol{\xi}^{\mathrm{o}} = \{\boldsymbol{x}^{\mathrm{o}}_{0:T}, \boldsymbol{u}_{0:T-1}\}$ collected from, say, a physical system (here, unlike $\boldsymbol{\xi}^{\mathrm{d}}$, $\boldsymbol{\xi}^{\mathrm{o}}$ is not necessarily optimal), and we wish to identify the system's dynamics. Here, $\boldsymbol{u}_{0:T-1}$ are usually externally supplied to ensure the physical system is of persistent excitation [126]. In order for $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.1) to only represent dynamics (as we do not care about its internal control law), we set $J(\boldsymbol{\theta}) = 0$. Then, $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ in (3.2) accepts any $\boldsymbol{u}^{\boldsymbol{\theta}}_{0:T-1} = \boldsymbol{u}_{0:T-1}$ as it always optimizes $J(\boldsymbol{\theta})$=0. In other words, by setting $J(\boldsymbol{\theta}) = 0$, $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.1) now only represents a class of dynamics models:

$$\boldsymbol{\Sigma}(\boldsymbol{\theta}): \quad \text{dynamics:} \quad \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\theta}) \quad \text{with } \boldsymbol{x}_0 \text{ and } \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T-1} = \boldsymbol{u}_{0:T-1}. \tag{3.5}$$

Now, $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ produces $\boldsymbol{\xi}_{\boldsymbol{\theta}} = \{\boldsymbol{x}^{\boldsymbol{\theta}}_{0:T}, \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T-1}\}$ subject to (3.5). To use (3.3) for identifying $\boldsymbol{\theta}$, we define

$$L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) = l(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\xi}^{\text{o}}), \tag{3.6}$$

where l is to quantify the prediction error between $\boldsymbol{\xi}^{\text{o}}$ and $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ under the same inputs $\boldsymbol{u}_{0:T-1}$.

Control/Planning Mode. Consider a system with its dynamics learned in the above SysID. We want to obtain a *feedback controller* or *trajectory* such that the system achieves a performance of minimizing a given cost function. To that end, we specialize $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.1) as follows: first, set $\boldsymbol{f}$ as the learned dynamics and $J(\boldsymbol{\theta}) = 0$; and second, through a *close-loop link*, we connect the input $\boldsymbol{u}_t$ and state $\boldsymbol{x}_t$ via a parameterized policy block $\boldsymbol{u}_t = \boldsymbol{u}(t, \boldsymbol{x}_t, \boldsymbol{\theta})$ (reminder: unlike SysID Mode with $\boldsymbol{u}_t$ supplied externally, the inputs here are from a policy via a feedback loop). $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ now becomes

$$\boldsymbol{\Sigma}(\boldsymbol{\theta}): \quad \begin{aligned} &\text{dynamics:} && \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) \quad \text{with} \quad \boldsymbol{x}_0, \\ &\text{control policy:} && \boldsymbol{u}_t = \boldsymbol{u}(t, \boldsymbol{x}_t, \boldsymbol{\theta}). \end{aligned} \tag{3.7}$$

Now, $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ produces a trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}} = \{\boldsymbol{x}^{\boldsymbol{\theta}}_{0:T}, \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T-1}\}$ subject to (3.7). We set the loss in (3.3) as

$$L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \sum_{t=0}^{T-1} l(\boldsymbol{x}^{\boldsymbol{\theta}}_t, \boldsymbol{u}^{\boldsymbol{\theta}}_t) + l_f(\boldsymbol{x}^{\boldsymbol{\theta}}_T), \tag{3.8}$$

where l and l_f are the stage and final costs, respectively. Then, (3.3) is an optimal control or planning problem: if $\boldsymbol{u}_t{=}\boldsymbol{u}(t, \boldsymbol{x}_t, \boldsymbol{\theta})$ (i.e., feedback policy explicitly depends on $\boldsymbol{x}_t$), (3.3) is a *model-based policy optimization* problem; otherwise if $\boldsymbol{u}_t{=}\boldsymbol{u}(t, \boldsymbol{\theta})$ (e.g., polynomial parameterization), (3.3) is an *open-loop motion planning* problem. This mode can also be used as a component to solve (3.1) in IRL/IOC Mode.

3.5 An End-to-End Learning Framework

To solve the generic problem in (3.3), the idea of end-to-end learning [106] seeks to optimize the loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ *directly* with respect to the tunable parameter $\boldsymbol{\theta}$, by applying the gradient descent

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k} \quad \text{with} \quad \frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k} = \frac{\partial L}{\partial \boldsymbol{\xi}}\Big|_{\boldsymbol{\xi}_{\boldsymbol{\theta}_k}} \frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k} + \frac{\partial L}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}. \tag{3.9}$$

Here, $k = 0, 1, \cdots$ is the iteration index; $\frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$ is the gradient of the loss with respect to $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_k$; and η_k is the learning rate. From (3.9), we can draw a learning architecture in Fig. 3.1. Each update of $\boldsymbol{\theta}$ consists of a *forward pass*, where at $\boldsymbol{\theta}_k$, the corresponding trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ is solved from $\boldsymbol{\Sigma}(\boldsymbol{\theta}_k)$ and the loss is computed, and a *backward pass*, where $\frac{\partial L}{\partial \boldsymbol{\xi}}\Big|_{\boldsymbol{\xi}_{\boldsymbol{\theta}_k}}$, $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$, and $\frac{\partial L}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$ are computed.

In the forward pass, $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ is obtained by solving an optimal control problem in $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ using any available OC methods, such as iLQR or Control/Planning Mode, (note that in SysID or Control/Planning modes, it is reduced to integrating difference equations (3.5) or (3.7)). In backward pass, $\frac{\partial L}{\partial \boldsymbol{\xi}}$ and $\frac{\partial L}{\partial \boldsymbol{\theta}}$ are easily obtained from the loss function $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$. The main challenge, however, is to solve $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$, i.e., *the derivative of a trajectory with respect to the parameters in the system*. Next, we will analytically solve $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ by proposing two techniques: *differential PMP* and *auxiliary control system*.

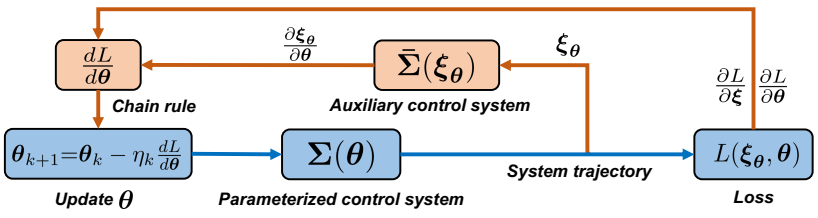


Figure 3.1. PDP end-to-end learning framework.

3.6 Key Contributions: Differential PMP & Auxiliary Control System

We first recall the discrete-time Pontryagin's Maximum/Minimum Principle (PMP) [44]. For the optimal control system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.1) with a fixed $\boldsymbol{\theta}$, PMP describes a set of optimality

conditions which the trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}} = \{\boldsymbol{x}^{\boldsymbol{\theta}}_{0:T}, \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T-1}\}$ in (3.2) must satisfy. To introduce these conditions, we first define the following *Hamiltonian*,

$$H_t = c_t(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}) + \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta})' \boldsymbol{\lambda}_{t+1}, \tag{3.10}$$

where $\boldsymbol{\lambda}_t \in \mathbb{R}^n$ $(t = 1, 2, \cdots, T)$ is called the *costate variable*, which can be also thought of as the Lagrange multipliers for the dynamics constraints. According to PMP, there exists a sequence of costates $\boldsymbol{\lambda}^{\boldsymbol{\theta}}_{1:T}$, which together with the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}} = \{\boldsymbol{x}^{\boldsymbol{\theta}}_{0:T}, \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T-1}\}$ satisfy

$$\text{dynamics equation:} \qquad \boldsymbol{x}^{\boldsymbol{\theta}}_{t+1} = \frac{\partial H_t}{\partial \boldsymbol{\lambda}^{\boldsymbol{\theta}}_{t+1}} = \boldsymbol{f}(\boldsymbol{x}^{\boldsymbol{\theta}}_t, \boldsymbol{u}^{\boldsymbol{\theta}}_t; \boldsymbol{\theta}), \tag{3.11a}$$

$$\text{costate equation:} \qquad \boldsymbol{\lambda}^{\boldsymbol{\theta}}_t = \frac{\partial H_t}{\partial \boldsymbol{x}^{\boldsymbol{\theta}}_t} = \frac{\partial c_t}{\partial \boldsymbol{x}^{\boldsymbol{\theta}}_t} + \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}^{\boldsymbol{\theta}}_t} \boldsymbol{\lambda}^{\boldsymbol{\theta}}_{t+1}, \tag{3.11b}$$

$$\text{input equation:} \qquad \boldsymbol{0} = \frac{\partial H_t}{\partial \boldsymbol{u}^{\boldsymbol{\theta}}_t} = \frac{\partial c_t}{\partial \boldsymbol{u}^{\boldsymbol{\theta}}_t} + \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}^{\boldsymbol{\theta}}_t} \boldsymbol{\lambda}^{\boldsymbol{\theta}}_{t+1}, \tag{3.11c}$$

$$\text{boundary conditions:} \qquad \boldsymbol{\lambda}^{\boldsymbol{\theta}}_T = \frac{\partial h}{\partial \boldsymbol{x}^{\boldsymbol{\theta}}_T}, \qquad \boldsymbol{x}^{\boldsymbol{\theta}}_0 = \boldsymbol{x}_0. \tag{3.11d}$$

For notation simplicity, $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}_t}$ means the derivative of function $\boldsymbol{g}(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ evaluated at $\boldsymbol{x}_t$.

3.6.1 Differential PMP

To begin, recall that our goal (in Section 3.5) is to obtain $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$, that is,

$$\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = \left\{ \frac{\partial \boldsymbol{x}^{\boldsymbol{\theta}}_{0:T}}{\partial \boldsymbol{\theta}}, \frac{\partial \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T-1}}{\partial \boldsymbol{\theta}} \right\}. \tag{3.12}$$

To this end, we are motivated to differentiate the PMP conditions in (3.11) on both sides with respect to $\boldsymbol{\theta}$. This leads to the following *differential PMP*:

$$\text{differential dynamics equation:} \quad \frac{\partial \boldsymbol{x}_{t+1}^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = F_t \frac{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + G_t \frac{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + E_t, \tag{3.13a}$$

$$\text{differential costate equation:} \quad \frac{\partial \boldsymbol{\lambda}_t^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = H_t^{xx} \frac{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_t^{xu} \frac{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + F_t' \frac{\partial \boldsymbol{\lambda}_{t+1}^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_t^{xe}, \tag{3.13b}$$

$$\text{differential input equation:} \quad \mathbf{0} = H_t^{ux} \frac{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_t^{uu} \frac{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + G_t' \frac{\partial \boldsymbol{\lambda}_{t+1}^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_t^{ue}, \tag{3.13c}$$

$$\text{differential boundary:} \quad \frac{\partial \boldsymbol{\lambda}_T^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = H_T^{xx} \frac{\partial \boldsymbol{x}_T^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_T^{xe}, \qquad \frac{\partial \boldsymbol{x}_0^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{x}_0}{\partial \boldsymbol{\theta}} = \mathbf{0}. \tag{3.13d}$$

Here, to simplify notations and distinguish knowns and unknowns, the coefficient matrices in the above differential PMP (3.13) are defined as follows:

$$F_t = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}}}, \quad G_t = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}}}, \quad H_t^{xx} = \frac{\partial^2 H_t}{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}} \partial \boldsymbol{x}_t^{\boldsymbol{\theta}}}, \quad H_t^{xe} = \frac{\partial^2 H_t}{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}} \partial \boldsymbol{\theta}}, \quad H_t^{xu} = \frac{\partial^2 H_t}{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}} \partial \boldsymbol{u}_t^{\boldsymbol{\theta}}} = (H_t^{ux})', \tag{3.14a}$$

$$E_t = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\theta}}, \quad H_t^{uu} = \frac{\partial^2 H_t}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}} \partial \boldsymbol{u}_t^{\boldsymbol{\theta}}}, \quad H_t^{ue} = \frac{\partial^2 H_t}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}} \partial \boldsymbol{\theta}}, \quad H_T^{xx} = \frac{\partial^2 h}{\partial \boldsymbol{x}_T^{\boldsymbol{\theta}} \partial \boldsymbol{x}_T^{\boldsymbol{\theta}}}, \quad H_T^{xe} = \frac{\partial^2 h}{\partial \boldsymbol{x}_T^{\boldsymbol{\theta}} \partial \boldsymbol{\theta}}, \tag{3.14b}$$

where we use $\frac{\partial^2 \boldsymbol{g}}{\partial \boldsymbol{x}_t \partial \boldsymbol{u}_t}$ to denote the second-order derivative of a function $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$ evaluated at $(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Since the trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}} = \{\boldsymbol{x}_{0:T}^{\boldsymbol{\theta}}, \boldsymbol{u}_{0:T-1}^{\boldsymbol{\theta}}\}$ is obtained in the forward pass (recall Fig. 3.1), all matrices in (3.14) are thus known (note that the computation of these matrices also requires $\boldsymbol{\lambda}_{1:T}^{\boldsymbol{\theta}}$, which can be obtained by iteratively solving (3.11b) and (3.11d) given $\boldsymbol{\xi}_{\boldsymbol{\theta}}$). From the differential PMP in (3.13), we note that to obtain $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ in (3.12), it is sufficient to compute the unknowns $\left\{ \frac{\partial \boldsymbol{x}_{0:T}^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}, \frac{\partial \boldsymbol{x}_{0:T-1}^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}, \frac{\partial \boldsymbol{\lambda}_{1:T}^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} \right\}$ in (3.13). Next we will show that how these unknowns are elegantly solved by introducing a new system.

3.6.2 Auxiliary Control System

One important observation to the differential PMP in (3.13) is that it shares a similar structure to the original PMP in (3.11); so it can be viewed as a new set of PMP equations corresponding to an 'oracle control optimal system' whose the 'optimal trajectory' is exactly (3.12). This motivates us to 'unearth' this oracle optimal control system, because by doing

so, (3.12) can be obtained from this oracle system by an OC solver. To this end, we first define the new 'state' and 'control' (matrix) variables:

$$X_t = \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{n\times r}, \qquad U_t = \frac{\partial \boldsymbol{u}_t}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{m\times r}, \tag{3.15}$$

respectively. Then, we 'artificially' define the following *auxiliary control system* $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$:

$$\begin{aligned} &\text{dynamics: } X_{t+1} = F_t X_t + G_t U_t + E_t \quad \text{with} \quad X_0 = \mathbf{0}, \\ \overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}}): \quad &\text{control objective: } \bar{J} = \operatorname{Tr} \sum_{t=0}^{T-1} \left(\frac{1}{2} \begin{bmatrix} X_t \\ U_t \end{bmatrix}' \begin{bmatrix} H_t^{xx} & H_t^{xu} \\ H_t^{ux} & H_t^{uu} \end{bmatrix} \begin{bmatrix} X_t \\ U_t \end{bmatrix} + \begin{bmatrix} H_t^{xe} \\ H_t^{ue} \end{bmatrix}' \begin{bmatrix} X_t \\ U_t \end{bmatrix} \right) \\ &\qquad + \operatorname{Tr} \left(\frac{1}{2} X_T' \, H_T^{xx} \, U_T + (H_T^{xe})' \, X_T \right). \end{aligned} \tag{3.16}$$

Here, $X_0 = \frac{\partial \boldsymbol{x}_0}{\partial \boldsymbol{\theta}} = \mathbf{0}$ because $\boldsymbol{x}_0$ in (3.1) is given; $\bar{J}$ is the defined control objective function which needs to be optimized in the auxiliary control system; and Tr denotes matrix trace. Before presenting the key results, we make some comments on the above auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$. First, its state and control variables are both matrix variables defined in (3.15). Second, its dynamics is linear and control objective function $\bar{J}$ is quadratic, for which the coefficient matrices are given in (3.14). Third, its dynamics and objective function are determined by the trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ of the system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in forward pass, and this is why we denote it as $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$. Finally, we have the following important result.

Lemma 3.6.1. *Let* $\{X_{0:T}^{\boldsymbol{\theta}}, U_{0:T-1}^{\boldsymbol{\theta}}\}$ *be a stationary solution to the auxiliary control system* $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ *in (3.16). Then,* $\{X_{0:T}^{\boldsymbol{\theta}}, U_{0:T-1}^{\boldsymbol{\theta}}\}$ *satisfies Pontryagin's Maximum Principle of* $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$*, which is (3.13), and*

$$\{X_{0:T}^{\boldsymbol{\theta}}, U_{0:T-1}^{\boldsymbol{\theta}}\} = \left\{ \frac{\partial \boldsymbol{x}_{0:T}^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}, \frac{\partial \boldsymbol{u}_{0:T-1}^{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} \right\} = \frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}. \tag{3.17}$$

Proof. To prove Lemma 3.6.1, we just need to show that the Pontryagin's Maximum Principle for the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ in (3.16) is exactly the differential PMP in (3.13). To this end, we define the following Hamiltonian for the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$:

$$\bar{H}_t = \operatorname{Tr}\left(\frac{1}{2}\begin{bmatrix} X_t \\ U_t \end{bmatrix}' \begin{bmatrix} H_t^{xx} & H_t^{xu} \\ H_t^{ux} & H_t^{uu} \end{bmatrix} \begin{bmatrix} X_t \\ U_t \end{bmatrix} + \begin{bmatrix} H_t^{xe} \\ H_t^{ue} \end{bmatrix}' \begin{bmatrix} X_t \\ U_t \end{bmatrix}\right) + \operatorname{Tr}\left(\Lambda_{t+1}'(F_t X_t + G_t U_t + E_t)\right), \quad (3.18)$$

with $t = 0, 1, \cdots, T-1$. Here $\Lambda_{t+1} \in \mathbb{R}^{n \times r}$ denotes the costate (matrix) variables for the auxiliary control system. Based on Section 3 in [127], there exists a sequence of costates $\Lambda_{1:T}^{\boldsymbol{\theta}}$, which together the stationary solution $\{X_{0:T}^{\boldsymbol{\theta}}, U_{0:T-1}^{\boldsymbol{\theta}}\}$ to the auxiliary control system must satisfy the following the matrix version of PMP (we here follow the notation style used in (3.11)).

The dynamics equation:

$$\begin{aligned} \frac{\partial \bar{H}_t}{\partial \Lambda_{t+1}^{\boldsymbol{\theta}}} &= \frac{\partial \operatorname{Tr}\left(\Lambda_{t+1}'(F_t X_t + G_t U_t + E_t)\right)}{\partial \Lambda_{t+1}}\Bigg|_{\substack{\Lambda_{t+1}=\Lambda_{t+1}^{\boldsymbol{\theta}} \\ X_t = X_t^{\boldsymbol{\theta}} \\ U_t = U_t^{\boldsymbol{\theta}}}} \\ &= F_t X_t^{\boldsymbol{\theta}} + G_t U_t^{\boldsymbol{\theta}} + E_t = \mathbf{0}. \end{aligned} \quad (3.19a)$$

The costate equation:

$$\begin{aligned} \frac{\partial \bar{H}_t}{\partial X_t^{\boldsymbol{\theta}}} &= \frac{\partial \operatorname{Tr}\left(\frac{1}{2}X_t' H_t^{xx} X_t\right) + \partial \operatorname{Tr}\left(U_t' H_t^{ux} X_t\right) + \partial \operatorname{Tr}\left(H_t^{ex} X_t\right) + \partial \operatorname{Tr}\left(\Lambda_{t+1}' F_t X_t\right)}{\partial X_t}\Bigg|_{\substack{\Lambda_{t+1}=\Lambda_{t+1}^{\boldsymbol{\theta}} \\ X_t = X_t^{\boldsymbol{\theta}} \\ U_t = U_t^{\boldsymbol{\theta}}}} \\ &= H_t^{xx} X_t^{\boldsymbol{\theta}} + H_t^{xu} U_t^{\boldsymbol{\theta}} + H_t^{xe} + F_t' \Lambda_{t+1}^{\boldsymbol{\theta}} = \Lambda_t^{\boldsymbol{\theta}}. \end{aligned} \quad (3.19b)$$

Input equation:

$$\begin{aligned} \frac{\partial \bar{H}_t}{\partial U_t^{\boldsymbol{\theta}}} &= \frac{\partial \operatorname{Tr}\left(\frac{1}{2}U_t' H_t^{uu} U_t\right) + \partial \operatorname{Tr}\left(U_t' H_t^{ux} X_t\right) + \partial \operatorname{Tr}\left(H_t^{eu} U_t\right) + \partial \operatorname{Tr}\left(\Lambda_{t+1}' G_t U_t\right)}{\partial U_t}\Bigg|_{\substack{\Lambda_{t+1}=\Lambda_{t+1}^{\boldsymbol{\theta}} \\ X_t = X_t^{\boldsymbol{\theta}} \\ U_t = U_t^{\boldsymbol{\theta}}}} \\ &= H_t^{uu} U_t^{\boldsymbol{\theta}} + H_t^{ux} X_t^{\boldsymbol{\theta}} + H_t^{ue} + G_t' \Lambda_{t+1}^{\boldsymbol{\theta}} = \mathbf{0}. \end{aligned} \quad (3.19c)$$

And boundary conditions:

$$\Lambda_T^{\boldsymbol{\theta}} = \frac{\partial \operatorname{Tr}(\frac{1}{2}X_T' H_T^{xx} X_T) + \partial \operatorname{Tr}((H_T^{xe})' X_T)}{\partial X_T}\Bigg|_{X_T = X_T^{\boldsymbol{\theta}}} = H_T^{xx} X_T^{\boldsymbol{\theta}} + H_T^{xe}, \quad (3.19d)$$

and $X_0^{\boldsymbol{\theta}} = \mathbf{0}$. Note that in the above derivations, we used the following matrix calculus [127]:

$$\frac{\partial \operatorname{Tr}(AB)}{\partial A} = B', \quad \frac{\partial f(A)}{\partial A'} = \left[\frac{\partial f(A)}{\partial A}\right]', \quad \frac{\partial \operatorname{Tr}(X'HX)}{\partial X} = HX + H'X, \tag{3.20}$$

and the following matrix trace properties:

$$\operatorname{Tr}(A) = \operatorname{Tr}(A'), \quad \operatorname{Tr}(ABC) = \operatorname{Tr}(BCA) = \operatorname{Tr}(CAB), \quad \operatorname{Tr}(A+B) = \operatorname{Tr}(A) + \operatorname{Tr}(B). \tag{3.21}$$

Since the above obtained PMP equations (3.19) are the same with the differential PMP in (3.13), we thus can conclude that the Pontryagin's Maximum Principle of the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ in (3.16) is exactly the differential PMP equations (3.13), and thus (3.17) holds. This completes the proof. □

Lemma 3.6.1 states two assertions. First, the PMP condition for the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ is exactly the differential PMP in (3.13) for the original system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$; and second, importantly, the trajectory $\{X_{0:T}^{\boldsymbol{\theta}}, U_{0:T-1}^{\boldsymbol{\theta}}\}$ produced by the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ is exactly the derivative of trajectory of the original system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ with respect to the parameter $\boldsymbol{\theta}$. Based on Lemma 3.6.1, we can obtain $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ from $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ efficiently by the lemma below.

Lemma 3.6.2. *If H_t^{uu} in (3.16) is invertible for all $t = 0, 1 \cdots, T-1$, define the following recursions*

$$P_t = Q_t + A_t'(I + P_{t+1}R_t)^{-1}P_{t+1}A_t, \tag{3.22a}$$

$$W_t = A_t'(I + P_{t+1}R_t)^{-1}(W_{t+1} + P_{t+1}M_t) + N_t, \tag{3.22b}$$

with $P_T = H_T^{xx}$ and $W_T = H_T^{xe}$. Here, I is identity matrix, $A_t = F_t - G_t(H_t^{uu})^{-1}H_t^{ux}$, $R_t = G_t(H_t^{uu})^{-1}G_t'$, $M_t = E_t - G_t(H_t^{uu})^{-1}H_t^{ue}$, $Q_t = H_t^{xx} - H_t^{xu}(H_t^{uu})^{-1}H_t^{ux}$, $N_t = H_t^{xe} - H_t^{xu}(H_t^{uu})^{-1}H_t^{ue}$ are

all known given (3.14). Then, the stationary solution $\{X^{\boldsymbol{\theta}}_{0:T}, U^{\boldsymbol{\theta}}_{0:T-1}\}$ *in (3.17) can be obtained by iteratively solving the following equations from* $t = 0$ *to* $T-1$ *with* $X^{\boldsymbol{\theta}}_0 = X_0 = \mathbf{0}$*:*

$$U^{\boldsymbol{\theta}}_t = -(H^{uu}_t)^{-1}\left(H^{ux}_t X^{\boldsymbol{\theta}}_t + H^{ue}_t + G_t{}'(I + P_{t+1}R_t)^{-1}\Big(P_{t+1}A_t X^{\boldsymbol{\theta}}_t + P_{t+1}M_t + W_{t+1}\Big)\right), \quad (3.23a)$$

$$X^{\boldsymbol{\theta}}_{t+1} = F_t X^{\boldsymbol{\theta}}_t + G_t U^{\boldsymbol{\theta}}_t + E_t. \quad (3.23b)$$

Proof. Based on Lemma 3.6.1 and its proof, we known that the PMP of the auxiliary control system, (3.19), is exactly the differential PMP equations (3.13). Thus below, we only look at the differential PMP equations in (3.19). From (3.19c), we solve for $U^{\boldsymbol{\theta}}_t$ (if H^{uu}_t invertible):

$$U^{\boldsymbol{\theta}}_t = -(H^{uu}_t)^{-1}\Big(H^{ux}_t X^{\boldsymbol{\theta}}_t + G'_t\Lambda^{\boldsymbol{\theta}}_{t+1} + H^{ue}_t\Big). \quad (3.24)$$

By substituting (3.24) into (3.19a) and (3.19b), respectively, and considering the definitions of matrices A_t, R_t, M_t, Q_t and N_t in (3.22), we have

$$X^{\boldsymbol{\theta}}_{t+1} = A_t X^{\boldsymbol{\theta}}_t - R_t\Lambda^{\boldsymbol{\theta}}_{t+1} + M_t, \quad (3.25)$$

$$\Lambda^{\boldsymbol{\theta}}_t = Q_t X^{\boldsymbol{\theta}}_t + A'_t\Lambda^{\boldsymbol{\theta}}_{t+1} + N_t, \quad (3.26)$$

for $t = 0, 1, \ldots, T-1$, and also the boundary condition in (3.19d)

$$\Lambda^{\boldsymbol{\theta}}_T = H^{xx}_T X^{\boldsymbol{\theta}}_T + H^{xe}_T,$$

for $t = T$. Next, we prove that there exist matrices P_t and W_t such that

$$\Lambda^{\boldsymbol{\theta}}_t = P_t X^{\boldsymbol{\theta}}_t + W_t. \quad (3.27)$$

Proof by induction: (3.19d) shows that (3.27) holds for $t = T$ if $P_T = H^{xx}_T$ and $W_T = H^{xe}_T$. Assume (3.27) holds for $t+1$, then by manipulating (3.25) and (3.26), we have

$$\Lambda^{\boldsymbol{\theta}}_t = \underbrace{\Big(Q_t + A'_t(I + P_{t+1}R_t)^{-1}P_{t+1}A_t\Big)}_{P_t} X^{\boldsymbol{\theta}}_t + \underbrace{A'_t(I + P_{t+1}R_t)^{-1}(W_{t+1} + P_{t+1}M_t) + N_t}_{W_t}, \quad (3.28)$$

which indicates (3.27) holds for t, if P_t and W_t satisfy (3.22a) and (3.22b), respectively. Substituting (3.27) to (3.26) and also considering (3.24) will lead to (3.23a). (3.23b) directly results from (3.19a). We complete the proof. □

Lemma 3.6.2 states that the trajectory of the above auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ can be obtained by two steps: first, iteratively solve (3.22) backward in time to obtain matrices P_t and W_t (all other coefficient matrices are known given $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$); second, calculate $\{X^{\boldsymbol{\theta}}_{0:T}, U^{\boldsymbol{\theta}}_{0:T-1}\}$ by iteratively integrating a feedback-control system (3.23) forward in time. In fact, these two steps constitute the standard procedure to solve general finite-time LQR problems [128].

As a conclusion to the techniques developed in Section 3.6, in Algorithm 3 we summarize the procedure of computing $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ via the introduced auxiliary control system. Algorithm 3 serves as a key component in the backward pass of the PDP learning framework, as shown in Fig. 3.1.

Algorithm 3: Solving $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ using Auxiliary Control System

Input: the trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ generated by the system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$

Compute the coefficient matrices (3.14) to obtain the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ in (3.16);

def Auxiliary_Control_System_Solver ($\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$): ▷ `Lemma 3.6.2`

Set $P_T = H^{xx}_T$ and $W_T = H^{xe}_T$;

for $t \leftarrow T$ **to** 0 **by** -1 **do**

Update P_t and W_t using equations (3.22); ▷ `backward in time`

end

Set $X^{\boldsymbol{\theta}}_0 = \mathbf{0}$;

for $t \leftarrow 0$ **to** T **by** 1 **do**

Update $X^{\boldsymbol{\theta}}_t$ and $U^{\boldsymbol{\theta}}_t$ using equations (3.23); ▷ `forward in time`

end

Return: $\{X^{\boldsymbol{\theta}}_{0:T}, U^{\boldsymbol{\theta}}_{0:T-1}\}$

Return: $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = \{X^{\boldsymbol{\theta}}_{0:T}, U^{\boldsymbol{\theta}}_{0:T-1}\}$

3.7 Applications

We investigate three learning modes of PDP, as described in Section 3.4. For each mode, we demonstrate its capability in four environments listed in Table 3.2. The environments are described in Appendix A. For each application mode of PDP, a baseline and a state-of-the-art learning/control methods are compared.

Table 3.2. Experimental environments

Systems	Dynamics parameter $\boldsymbol{\theta}_{\text{dyn}}$	Control objective $\boldsymbol{\theta}_{\text{obj}}$
Cartpole	cart mass, pole mass and length	$c{=}\|\boldsymbol{\theta}'_{\text{obj}}(\boldsymbol{x}-\boldsymbol{x}_{\text{g}})\|^2{+}\|\boldsymbol{u}\|^2$ $h=\|\boldsymbol{\theta}'_{\text{obj}}(\boldsymbol{x}-\boldsymbol{x}_{\text{g}})\|^2$
Two-link robot arm	length and mass for each link	
6-DoF quadrotor maneuvering	mass, wing length, inertia matrix	
6-DoF rocket powered landing	mass, rocket length, inertia matrix	

We fix the unit weight to $\|\boldsymbol{u}\|^2$, because estimating all weights will incur ambiguity [40]; $\boldsymbol{x}_{\text{g}}$ is the goal state. Results for 6-DoF rocket landing is in Section 3.10)

3.7.1 IRL/IOC Mode

Algorithm 4: Algorithm of PDP in IRL/IOC Mode

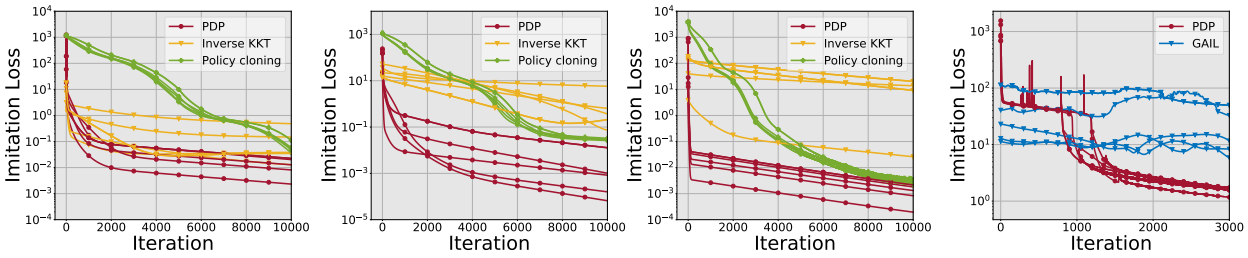
Data: Expert demonstrations $\{\boldsymbol{\xi}^{\text{d}}\}$
Parameterization: The parameterized optimal control system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.1)
Loss: $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ in (3.4)
Initialization: $\boldsymbol{\theta}_0$, learning rate $\{\eta_k\}_{k=0,1,\cdots}$
for $k = 0, 1, 2, \cdots$ **do**
  Solve $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ from the optiaml control system $\boldsymbol{\Sigma}(\boldsymbol{\theta}_k)$; ▷ `using any OC solver`
  Obtain $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$ using Algorithm 3 given $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$; ▷ `using Algorithm 3`
  Obtain $\frac{\partial L}{\partial \boldsymbol{\xi}}\Big|_{\boldsymbol{\xi}_{\boldsymbol{\theta}_k}}$ from the given loss function $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$;
  Apply the chain rule (3.9) to obtain $\frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$;
  Update $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k \frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$;
end

As formulated in Section 3.4, in the IRL/IOC mode of PDP, the parameterized $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ is in (3.1) and the loss in (3.4). In the forward pass of PDP, $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ is solved from $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ by any OC solver. In the backward pass, $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ is computed from the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ in (3.16) using Algorithm 3. The full algorithm is depicted in Algorithm 4.

Experiment: Imitation Learning. We use IRL/IOC Mode to solve imitation learning in environments in Table 3.2. The true dynamics is parameterized, and control objective is parameterized as a weighted distance to the goal, $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\text{dyn}}, \boldsymbol{\theta}_{\text{obj}}\}$. The dataset of expert demonstrations $\{\boldsymbol{\xi}^{\text{d}}\}$ is generated by solving an expert optimal control system with the expert's dynamics and control objective parameter $\boldsymbol{\theta}^* = \{\boldsymbol{\theta}^*_{\text{dyn}}, \boldsymbol{\theta}^*_{\text{dyn}}\}$ given. We generate a number of five trajectories, where different trajectories $\boldsymbol{\xi}^{\text{d}} = \{\boldsymbol{x}^{\text{d}}_{0:T}, \boldsymbol{u}^{\text{d}}_{0:T-1}\}$ have different initial conditions $\boldsymbol{x}_0$ and time horizons T (T ranges from 40 to 50). Set imitation loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \|\boldsymbol{\xi}^{\text{d}} - \boldsymbol{\xi}_{\boldsymbol{\theta}}\|^2$. Two other methods are compared: (i) neural policy cloning, and (ii) inverse KKT [34]. We set learning rate $\eta = 10^{-4}$ and run five trials given random initial $\boldsymbol{\theta}_0$. The results in Fig. 3.2a-3.2c show that PDP significantly outperforms the policy cloning and inverse-KKT for a much lower training loss and faster convergence. In Fig. 3.2d, we apply the PDP to learn a neural control objective function for the robot arm using the same demonstration data in Fig. 3.2b, and we also compare with the GAIL [129]. Results in Fig. 3.2d show that the PDP successfully learns a neural objective function and the imitation loss of PDP is much lower than that of GAIL. It should note that because the demonstrations are not strictly realizable (optimal) under the parameterized neural objective function, the final loss for the PDP is small but not zero. This indicates that given sub-optimal demonstrations, PDP can still find the 'best' control objective function within the function set $J(\boldsymbol{\theta})$ such that its reproduced $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ has the *minimal distance* to the demonstrations. Please refer to Section 3.10 for more experiment details and additional validations.

3.7.2 SysID Mode

In the SysID mode, $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ is (3.5) and loss is (3.6). PDP is greatly simplified: in forward pass, $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ is solved by integrating the difference equation (3.5). In the backward pass, $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ is reduced to



(a) Cart-pole (b) Robot arm (c) Quadrotor (d) Comparison

Figure 3.2. (a-c) imitation loss v.s. iteration, (d) PDP learns a neural objective function and comparison.

$$\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}}): \qquad \text{dynamics:} \quad X^{\boldsymbol{\theta}}_{t+1} = F_t X^{\boldsymbol{\theta}}_t + E_t \quad \text{with} \quad X_0 = \mathbf{0}. \tag{3.29}$$

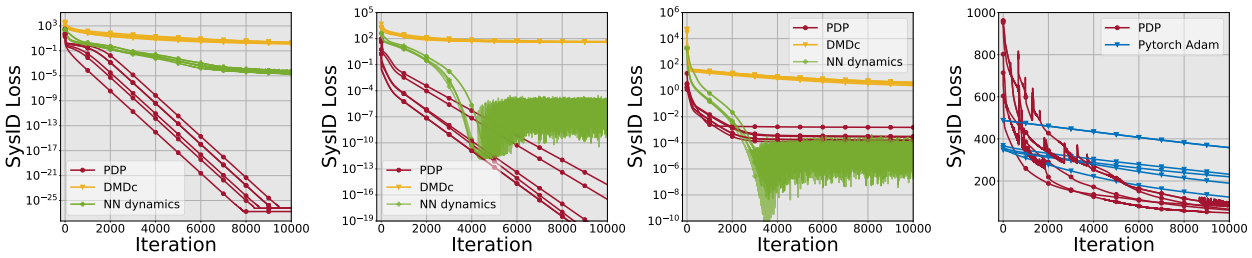
This is because $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.5) results from letting $J(\boldsymbol{\theta}) = 0$, (3.13b-3.13d) and $\bar{J}$ in (3.16) are then trivialized, and due to $\boldsymbol{u}_{0:T-1}$ given, $U^{\boldsymbol{\theta}}_t = \mathbf{0}$ in (3.13a). The algorithm is in Algorithm 5.

Algorithm 5: Algorithm of PDP in SysID Mode

Data: Input-state data $\{\boldsymbol{\xi}^{\text{o}}\}$
Parameterization: The parameterized dynamics model $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.5)
Loss: $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ in (3.6)
Initialization: $\boldsymbol{\theta}_0$, learning rate $\{\eta_k\}_{k=0,1,\cdots}$
for $k = 0, 1, 2, \cdots$ **do**
  Obtain $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by iteratively integrating $\boldsymbol{\Sigma}(\boldsymbol{\theta}_k)$ in (3.5) for $t = 0, ..., T-1$;
  Compute the coefficient matrices (3.14) to obtain the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ in (3.29);
  Obtain $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$ by iteratively integrating $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}_k})$ in (3.29) for $t = 0, ..., T-1$;
  Obtain $\frac{\partial L}{\partial \boldsymbol{\xi}}\Big|_{\boldsymbol{\xi}_{\boldsymbol{\theta}_k}}$ from the given loss function in (3.6);
  Apply the chain rule (3.9) to obtain $\frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$;
  Update $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k \frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$;
end

Experiment: System Identification. We use the SysID Mode to identify the dynamics parameter $\boldsymbol{\theta}_{\text{dyn}}$ for the systems in Table 3.2. We collect a total number of five trajectories from systems (in Table 3.2) with dynamics known, wherein different trajectories

$\boldsymbol{\xi}^{\mathrm{o}} = \{\boldsymbol{x}^{\mathrm{o}}_{0:T}, \boldsymbol{u}_{0:T-1}\}$ have different initial conditions $\boldsymbol{x}_0$ and horizons T (T ranges from 10 to 20), with random inputs $\boldsymbol{u}_{0:T-1}$ drawn from uniform distribution. Set the SysID loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \|\boldsymbol{\xi}^{\mathrm{o}} - \boldsymbol{\xi}_{\boldsymbol{\theta}}\|^2$. Two other methods are compared: (i) learning a neural network (NN) dynamics model, and (ii) DMDc [130]. For all methods, we set learning rate $\eta = 10^{-4}$, and run five trials with random $\boldsymbol{\theta}_0$. The results are in Fig. 3.3. Fig. 3.3a-3.3c show an obvious advantage of PDP over the NN baseline and DMDc in terms of lower training loss and faster convergence speed. In Fig. 3.3d, we compare PDP and Adam [131] (here both with $\eta = 10^{-5}$) for training the same neural dynamics model for the robot arm. The results again show that PDP outperforms Adam for faster learning speed and lower training loss. Such advantages are due to that PDP has injected an inductive bias of optimal control into learning, making it more efficient for handling dynamical systems. More experiments and validations are in Section 3.10.



(a) Cart-pole (b) Robot arm (c) Quadrotor (d) Neural dynamics

Figure 3.3. (a-c) SysID loss v.s. iteration, (d) PDP learns a neural dynamics model.

3.7.3 Control/Planning Mode

The parameterized system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ is (3.7) and loss is (3.8). PDP for this mode is also simplified. In forward pass, $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ is solved by integrating a (controlled) difference equation (3.7). In backward pass, $\bar{J}$ in the auxiliary control system (3.16) is trivialized because we have considered $J(\boldsymbol{\theta}) = 0$ in (3.7). Since the control is now given by $\boldsymbol{u}_t = \boldsymbol{u}(t, \boldsymbol{x}_t, \boldsymbol{\theta})$, $U_t^{\boldsymbol{\theta}}$ is obtained by differentiating the policy on both side with respect to $\boldsymbol{\theta}$, that is, $U_t^{\boldsymbol{\theta}} = U_t^x X_t^{\boldsymbol{\theta}} + U_t^e$ with $U_t^x = \frac{\partial \boldsymbol{u}_t}{\partial \boldsymbol{x}_t}$ and $U_t^e = \frac{\partial \boldsymbol{u}_t}{\partial \boldsymbol{\theta}}$. Thus,

$$\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}}): \quad \begin{aligned} \text{dynamics:} \quad & X^{\boldsymbol{\theta}}_{t+1} = F_t X^{\boldsymbol{\theta}}_t + G_t U^{\boldsymbol{\theta}}_t \quad \text{with} \quad X_0 = \mathbf{0}, \\ \text{control policy:} \quad & U^{\boldsymbol{\theta}}_t = U^x_t X^{\boldsymbol{\theta}}_t + U^e_t. \end{aligned} \tag{3.30}$$

Integrating the above auxiliary control system in (3.30) from $t = 0$ to T leads to $\{X^{\boldsymbol{\theta}}_{0:T}, U^{\boldsymbol{\theta}}_{0:T-1}\} = \frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$. The whole algorithm is in Algorithm 6.

Algorithm 6: Algorithm of PDP in Control/Planning Mode

Parameterization: The parameterized-policy system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.7)

Loss: $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ in (3.8)

Initialization: $\boldsymbol{\theta}_0$, learning rate $\{\eta_k\}_{k=0,1,\cdots}$

for $k = 0, 1, 2, \cdots$ **do**

Obtain $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by iteratively integrating $\boldsymbol{\Sigma}(\boldsymbol{\theta}_k)$ in (3.7) for $t = 0, ..., T-1$;

Compute the coefficient matrices (3.14) to obtain the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ in (3.30);

Obtain $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$ by iteratively integrating $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}_k})$ in (3.30) for $t = 0, ..., T-1$;

Obtain $\frac{\partial L}{\partial \boldsymbol{\xi}}\Big|_{\boldsymbol{\xi}_{\boldsymbol{\theta}_k}}$ from the given loss function $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ in (3.8);

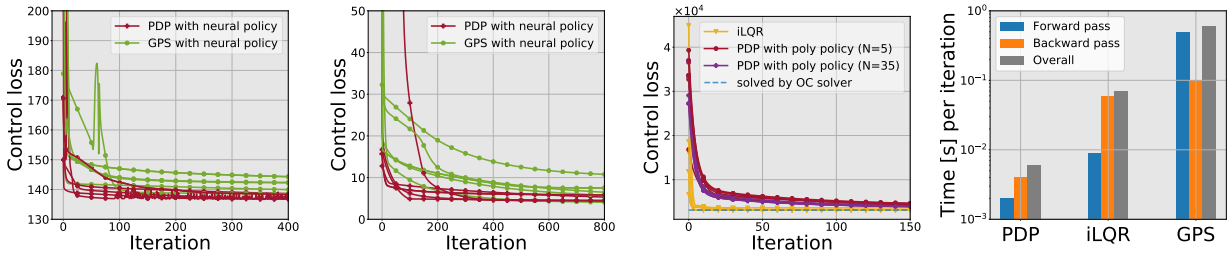
Apply the chain rule (3.9) to obtain $\frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$;

Update $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k \frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$;

end

Experiment: Control and Planning. Based on identified dynamics, we learn policies of each system to optimize a control objective with given $\boldsymbol{\theta}_{\text{obj}}$. We set loss (3.8) as the control objective (below called control loss). To parameterize policy (3.7), we use a Lagrange polynomial of degree N (for planning) or neural network (for feedback control). iLQR [7] and guided policy search (GPS) [98] are compared. We set learning rate η=10^{-4} or 10^{-6} and run five trials for each system. Fig. 3.4a-3.4b are learning neural network feedback policies for the cart-pole and robot arm, respectively. The results show that PDP outperforms GPS for having lower control loss. Fig. 3.4c is motion planning for quadrotor using a polynomial policy. It shows that PDP achieves a competitive performance with iLQR.

Compared to iLQR, PDP minimizes over polynomial policies instead of input sequences, and thus has a higher final loss which depends on the expressiveness of the polynomial: e.g., the polynomial of degree N=35 has a lower loss than that of N=5. Since iLQR can be viewed as '1.5-order' method (discussed in Section 3.1), it has faster converging speed than PDP which is only first-order, as shown in Fig. 3.4c. But iLQR is computationally extensive, PDP, instead, has a huge advantage of running time, as illustrated in Fig. 3.4d. Due to space constraint, we put detailed analysis between GPS and PDP in Section 3.10.



(a) Cart-pole control (b) Robot arm control (c) Quadrotor planning (d) Timing results

Figure 3.4. (a-c) control loss v.s. iteration, (d) comparison for running time per iteration.

3.8 Discussion

3.8.1 Complexity of PDP

We consider the algorithm complexity of different learning modes of PDP, and suppose that the time horizon of the parameterized system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ is T.

IRL/IOC Mode (Algorithm 4): in forward pass, PDP needs to obtain and store the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ of the optimal control system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.1), and this optimal trajectory can be solved by any (external) optimal control solver. In backward pass, PDP first uses $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ to build the auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ in (3.16) and then computes $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ by Lemma 3.6.2, which takes $2T$ steps.

SysID Mode (Algorithm 5): in forward pass, PDP needs to obtain and store the trajectory $\boldsymbol{\xi}_{\theta}$ of the original dynamics system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.5). Such trajectory is simply a result of iterative integration of (3.5), which takes T steps. In backward pass, PDP first uses $\boldsymbol{\xi}_{\theta}$ to build the

auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ in (3.29) and then computes $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ by iterative integration of (3.29), which takes T steps.

Control/Planning Mode (Algorithm 6): in forward pass, PDP needs to obtain and store the trajectory $\boldsymbol{\xi}_{\theta}$ of the controlled system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ in (3.7). Such trajectory is simply a result of iterative integration of (3.7), which takes T steps. In backward pass, PDP first uses $\boldsymbol{\xi}_{\theta}$ to build an auxiliary control system $\overline{\boldsymbol{\Sigma}}(\boldsymbol{\xi}_{\boldsymbol{\theta}})$ in (3.30) and then computes $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ by integration of (3.30), which takes T steps.

Therefore, we can summarize that the memory- and computational- complexity for the PDP framework is only linear to the time horizon T of the parameterized system $\boldsymbol{\Sigma}(\boldsymbol{\theta})$. This is significantly advantageous over existing end-to-end learning frameworks, as summarized in Table 3.3 (discussed in Section 3.3.1).

Table 3.3. Complexity comparison for different end-to-end learning frameworks

Learning frameworks	Forward pass		Backward pass	
	Method and accuracy	Complexity (linear to)	Method	Complexity (linear to)
PI-Net [112]	N-step unrolled graph using gradient descent; accuracy depends on N	computation: NT memory: NT	Back-propagation over the unrolled graph	computation: NT memory: NT
UPN [115]	N-step unrolled graph using gradient descent; accuracy depends on N	computation: NT memory: NT	Back-propagation over the unrolled graph	computation: NT memory: NT
Diff-MPC [114]	iLQR finds fixed points; can achieve any accuracy	computation: — memory: T	Differentiate the LQR approximation and solve linear equations	computation: T^2 memory: T^2
PDP	Accept any OC solver; can achieve any accuracy	computation: —, memory: T	Auxiliary control system	computation: T, memory: T

*Here T denotes the time horizon of the system; N (usually large) is the number of steps of applying gradient descent until the convergence of the trajectory solution to the optimal control problem.

In Fig. 3.5, we have also compared the running time of PDP with that of differentiable MPC [114]. Compared to differentiable MPC, the first advantage of the PDP framework is that the differentiation in the backward pass is directly performed on the parameterized optimal control system (by differentiating through PMP). Second, we develop the auxiliary control system in the backward pass of PDP, whose trajectory is exactly the gradient of the system trajectory in the forward pass. The gradient then is iteratively solved using the

auxiliary control system by Lemma 3.6.2 (Algorithm 3). Those proposed techniques enables the PDP to have significant advantage in computational efficiency over differentiable MPC. To illustrate this, we have compare the algorithm complexity for both PDP and differentiable MPC in Table 3.3 and provide an experiment in Fig. 3.5.

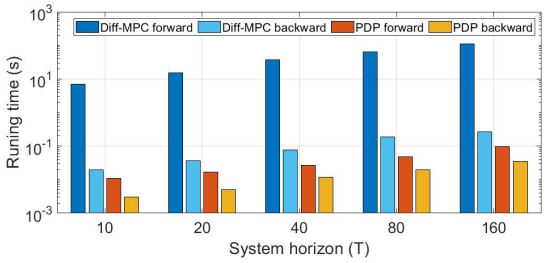


Figure 3.5. Runtime (per iteration) comparison between the PDP and differentiable MPC [114] for different time horizons of a pendulum system. Note that y-axis is log-scale, and the runtime is averaged over 100 iterations. Both methods are implemented in Python and run on the same machine using CPUs. The results show that the PDP runs 1000x faster than differentiable MPC.

3.8.2 Convergence of PDP

PDP is a First-Order Method. We observe that (i) all gradient quantities in PDP are analytical and exact; (ii) the development of PDP does not involve any second-order derivative/approximation of functions or models (note that PMP is a first-order optimality condition for optimal control); and (iii) PDP minimizes a loss function directly with respect to unknown parameters in a system using gradient descent. Thus, we conclude that PDP is a first-order gradient-descent based optimization framework. Specifically for the SysID and Control/Planning modes of PDP, they are also first-order algorithms. When using these modes to solve optimal control problems, this first-order nature may bring disadvantages of PDP compared to high-order methods, such as iLQR which can be considered as 1.5-order because it uses second-order derivative of a value function and first-order derivative of dynamics, or DDP which is a second-order method as it uses the second-order derivatives of both value function and dynamics. The disadvantages of PDP have already been empirically shown in Fig. 3.4c, where the converging speed of PDP in its planning mode is slower than

that of iLQR. For empirical comparisons between first- and second-order techniques, we refer the reader to [132].

Convergence to Local Optima. *Since PDP is a first-order gradient-descent based algorithm, PDP can only achieve local minima for general non-convex optimization problems in (3.3).* Furthermore, we observe that the general problem in (3.3) belongs to a bi-level optimization framework. As explored in [133], under certain assumptions such as convexity and smoothness on models (e.g., dynamics model, policy, loss function and control objective function), global convergence of the bi-level optimization can be established. But we think such conditions are too restrictive in the context of dynamical control systems. As a future direction, we will investigate mild conditions for good convergence by resorting to dynamical system and control theory, such as Lyapunov theory.

Parameterization Matters for Convergence. Although PDP only achieves local convergence, these still exists a question of how likely PDP can obtain the global convergence. In our empirical experiments, we find that how models are parameterized matters for good convergence performance. For example, in IOC/IRL mode, we observe that using a neural network control objective function (in Fig. 3.2d) is more likely to get trapped in local minima than using the parameterization of weighted distance objective functions (in Fig. 3.2a-3.2c). In control/planning mode, using a deeper neural network policy (in Fig. 3.4a-3.4b) is more like to result in local minima than using a simpler one. Also in the motion planning experiment, we use the Lagrange polynomial to parameterize a policy instead of using standard polynomials, because the latter can lead to poor conditioning and sensitivity issues (a small change of polynomial parameter results in large change in performance) and thus more easily get stuck in local minima. One high-level explanation is that more complex parameterization will bring extreme non-convexity to the optimization problem, making the algorithm more easily trapped in local minima. Again, how to theoretically justify those empirical experience and find the mild conditions for global convergence guarantee still needs to be investigated in future research.

3.9 Conclusions

This chapter presents a Pontryagin differentiable programming (PDP) methodology to establish an end-to-end learning framework for solving a range of learning and control tasks. The key contribution in PDP is that we incorporate the knowledge of optimal control theory as an inductive bias into the learning framework. Such combination enables PDP to achieve higher efficiency and capability than existing learning and control methods in solving many tasks including inverse reinforcement learning, system identification, and control/planning. We envision the proposed PDP could benefit to both learning and control fields for solving many high-dimensional continuous-space problems.

3.10 Supplementary: Experiments Details

3.10.1 System/Environment Setup

Dynamics Discretization. All the experimental environments/systems involved are described in Appendix A. The continuous-time dynamics of all experimental systems in Table 3.2 are discretized using the Euler method: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \Delta \cdot \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ with the discretization interval $\Delta = 0.05$s or $\Delta = 0.1$s.

Simulation Environment Source Codes. We have made different simulation environments/systems in Table 3.2 as a standalone Python package, which is available at https://github.com/wanxinjin/Pontryagin-Differentiable-Programming. This environment package is easy to use and has user-friendly interfaces for customization.

3.10.2 Experiment of Imitation Learning

Data Acquisition. The dataset of expert demonstrations $\{\boldsymbol{\xi}^{\mathrm{d}}\}$ is generated by solving an expert optimal control system with the expert's dynamics and control objective parameter $\boldsymbol{\theta}^* = \{\boldsymbol{\theta}^*_{\mathrm{dyn}}, \boldsymbol{\theta}^*_{\mathrm{dyn}}\}$ given. We generate a number of five trajectories, where different trajectories $\boldsymbol{\xi}^{\mathrm{d}} = \{\boldsymbol{x}^{\mathrm{d}}_{0:T}, \boldsymbol{u}^{\mathrm{d}}_{0:T-1}\}$ have different initial conditions $\boldsymbol{x}_0$ and time horizons T (T ranges from 40 to 50).

Inverse KKT Method. We choose the inverse KKT method [34] for comparison because it is suitable for learning objective functions for high-dimensional continuous-space systems. We adapt the inverse KKT method, and define the KKT loss as the norm-2 violation of the KKT conditions [41] by the demonstration data $\boldsymbol{\xi}^{\mathrm{d}}$, that is,

$$\min_{\boldsymbol{\theta},\boldsymbol{\lambda}_{1:T}} \left(\left\| \frac{\partial L}{\partial \boldsymbol{x}_{0:T}}(\boldsymbol{x}^{\mathrm{d}}_{0:T}, \boldsymbol{u}^{\mathrm{d}}_{0:T-1}) \right\|^2 + \left\| \frac{\partial L}{\partial \boldsymbol{u}_{0:T\text{-}1}}(\boldsymbol{x}^{\mathrm{d}}_{0:T}, \boldsymbol{u}^{\mathrm{d}}_{0:T-1}) \right\|^2 \right), \tag{3.31}$$

where $\frac{\partial L}{\partial \boldsymbol{x}_{0:T}}(\cdot)$ and $\frac{\partial L}{\partial \boldsymbol{u}_{0:T-1}}(\cdot)$ are the derivatives of Lagrangian L with respect to state and control sequences, respectively, and $\boldsymbol{\theta}$=$\{\boldsymbol{\theta}_{\mathrm{dyn}}, \boldsymbol{\theta}_{\mathrm{dyn}}\}$. We minimize the above KKT-loss with respect to the unknown $\boldsymbol{\theta}$ and the costate variables $\boldsymbol{\lambda}_{1:T}$.

Note that to illustrate the inverse-KKT learning results in Fig. 3.2, we plot the imitation loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \|\boldsymbol{\xi}^{\mathrm{d}} - \boldsymbol{\xi}_{\boldsymbol{\theta}}\|^2$ instead of the KKT loss (3.31), because we want to guarantee that the comparison criterion is the same across different methods. Thus for each iteration k in minimizing the KKT loss (3.31), we use the parameter $\boldsymbol{\theta}_k$ to compute the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ and obtain the imitation loss.

Neural Policy Cloning. For the neural policy cloning (similar to [134]), we directly learn a neural-network policy $\boldsymbol{u} = \boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x})$ from the dataset using supervised learning, that is

$$\min_{\boldsymbol{\theta}} \textstyle\sum_{t=0}^{T-1} \|\boldsymbol{u}^{\mathrm{d}}_t - \boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x}^{\mathrm{d}}_t)\|^2. \tag{3.32}$$

Learning Neural Control Objective Functions. In Fig. 3.2d, we apply PDP to learn a neural objective function of the robot arm. The neural objective function is constructed as

$$J(\boldsymbol{\theta}) = V_{\boldsymbol{\theta}}(\boldsymbol{x}) + 0.0001\|\boldsymbol{u}\|^2, \tag{3.33}$$

with $V_{\boldsymbol{\theta}}(\boldsymbol{x})$ a fully-connected feed-forward network with `n-n-1` layers and tanh activation functions, i.e., an input layer with `n` neurons equal to the dimension of state, n, one hidden layer with `n` neurons and one output layer with `1` neuron. $\boldsymbol{\theta}$ is the neural network parameter. We separate the input cost from the neural network because otherwise it will cause instability when solving OC problems in the forward pass. Also, in learning the above neural objective function, we fix the dynamics because otherwise it will also lead to instability of solving OC.

In the comparing GAIL method [129], we use the following hyper-parameters: the policy network is a fully-connected feed-forward network with `n-400-300-m` layers and `relu` activation functions; the discriminator network is a `(n+m)-400-300-1` fully-connected feed-forward network with `tanh` and `sigmoid` activation functions; and the policy regularizer λ is set to zero.

Results and Validation. In Fig. 3.6, we show more detailed results of imitation loss versus iteration for three systems (cart-pole, robot arm, and quadrotor). On each system, we run five trials for all methods with random initial guess, and the learning rate for all methods is set as $\eta = 10^{-4}$. In Fig. 3.9, we validate the learned models (i.e., learned dynamics and learned control objective) by performing motion planning of each system in unseen settings. Specifically, we set each system with new initial state $\boldsymbol{x}_0$ and horizon T and plan the control trajectory using the learned models, and we also show the corresponding true trajectory of the expert.

3.10.3 Experiment of System Identification

Data Acquisition. In the system identification experiment, we collect a total number of five trajectories from systems (in Table 3.2) with dynamics known, wherein different trajectories $\boldsymbol{\xi}^{\mathrm{o}} = \{\boldsymbol{x}^{\mathrm{o}}_{0:T}, \boldsymbol{u}_{0:T-1}\}$ have different initial conditions $\boldsymbol{x}_0$ and horizons T (T ranges from 10 to 20), with random inputs $\boldsymbol{u}_{0:T-1}$ drawn from uniform distribution.

DMDc Method. The DMDc method [130], which can be viewed as a variant of Koopman theory [70], estimates a linear dynamics model $\boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t$, using the following least square regression

$$\min_{A,B} \sum_{t=0}^{T-1} \|\boldsymbol{x}^{\mathrm{o}}_{t+1} - A\boldsymbol{x}^{\mathrm{o}}_t - B\boldsymbol{u}_t\|^2. \tag{3.34}$$

Neural Network Baseline. For the neural network baseline, we use a neural network $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{u})$ to represent the system dynamics, where the input of the network is state and

control vectors, and output is the state of next step. We train the neural network by minimizing the following residual

$$\min_{\boldsymbol{\theta}} \sum_{t=0}^{T-1} \|\boldsymbol{x}_{t+1}^{\mathrm{o}} - \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_t^{\mathrm{o}}, \boldsymbol{u}_t)\|^2. \tag{3.35}$$

Learning Neural Dynamics Model. In Fig. 3.3d, we compare the performance of PDP with Adam [131] for learning the same neural dynamics model for the robot arm system. Here, the neural dynamics model is a fully-connected feed-forward neural network with `(m+n)-(2m+2n)-n` layers and tanh activation functions, that is, an input layer with `(m+n)` neurons equal to the dimension of state, n, plus the dimension of control m, one hidden layer with `(2m+2n)` neurons and one output layer with `(n)` neurons. The learning rate for the PDP and the PyTorch Adam is both set as $\eta = 10^{-5}$.

Results and Validation. In Fig. 3.7, we show more detailed results of SysID loss versus iteration for the three systems (cart-pole, robot arm, and quadrotor). On each system, we run five trials with random initial guess, and we set the learning rate as $\eta = 10^{-4}$ for all methods. In Fig. 3.10, we use the learned dynamics model to perform motion prediction of each system in unactuated conditions (i.e., $\boldsymbol{u}_t = \mathbf{0}$), in order to validate the effectiveness/correctness of the learned dynamics models.

3.10.4 Experiment of Control/Planning

We use the dynamics identified in the system ID part, and the specified control objective function is set as weighted distance to the goal, as given in Table 3.2 ($\boldsymbol{\theta}_{\mathrm{obj}}$ is given). Throughout the optimal control/planning experiments, we use the time horizons T ranging from 20 to 40.

Learning Neural Network Policies. On the cart-pole and robot-arm systems (in Fig. 3.4a and Fig. 3.4b), we learn a feedback policy by minimizing given control objective functions. For both systems, we parameterize the policy using a neural network. Specifically, we use a fully-connected feed-forward neural network which has a layer structure of `n-n-m` with tanh activation functions, i.e., there is an input layer with `n` neurons equal to the

dimension of state, one hidden layer with `n` neurons and one output layer with `m` neurons. The policy parameter $\boldsymbol{\theta}$ is the neural network parameter. We apply the PDP Control/Planning mode in Algorithm 6 and set the learning rate $\eta = 10^{-4}$. For comparison, we apply the guided policy search (GPS) method [98] (its deterministic version) to learn the same neural policy with the learning rate $\eta = 10^{-6}$ (η in GPS is used to update the Lagrange multipliers for the policy constraint and we choose $\eta = 10^{-6}$ because it achieves the most stable results).

Motion Planning with Lagrange Polynomial Policies. On the 6-DoF quadrotor, we use PDP to perform motion planning, that is, to find a control sequence to minimize the given control cost (loss) function. Here, we parameterize the policy $\boldsymbol{u}_t = \boldsymbol{u}(t, \boldsymbol{\theta})$ as N-degree Lagrange polynomial [135] with $N+1$ pivot points evenly populated over the time horizon, that is, $\{(t_0, \boldsymbol{u}_0), (t_1, \boldsymbol{u}_1), \cdots, (t_N, \boldsymbol{u}_N)\}$ with $t_i = iT/N$, $i = 0, \cdots, N$. The analytical form of the parameterized policy is

$$\boldsymbol{u}(t, \boldsymbol{\theta}) = \sum_{i=0}^{N} \boldsymbol{u}_i b_i(t) \qquad \text{with} \qquad b_i(t) = \prod_{0 \le j \le N, j \neq i} \frac{t - t_j}{t_i - t_j}. \tag{3.36}$$

Here, $b_i(t)$ is called Lagrange basis, and the policy parameter $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta} = [\boldsymbol{u}_0, \cdots, \boldsymbol{u}_N]' \in \mathbb{R}^{m(N+1)}. \tag{3.37}$$

The above Lagrange polynomial parameterization has been normally used in some trajectory optimization method such as [51, 136]. In this planning experiment, we have used different degrees of Lagrange polynomials, i.e., $N = 5$ and $N = 35$, respectively, to show how policy expressiveness can influence the final control loss (cost). The learning rate in PDP is set as $\eta = 10^{-4}$. For comparison, we also apply iLQR [7] to solve for the optimal control sequence.

Results. In Fig. 3.8, we show the detailed results of control loss (i.e. the value of control objective function) versus iteration for three systems (cart-pole, robot arm, and quadrotor). For each system, we run five trials with random initial parameter $\boldsymbol{\theta}_0$. In Fig. 3.11, we apply the learned neural network policies (for cart-pole and robot arm systems) and the Lagrange polynomial policy (for quadrotor system) to simulate the corresponding

system. For reference, we also plot the optimal trajectory solved by an OC solver [50] (which corresponds to the minimal control cost).

Comments on the Results Between GPS [98] and PDP. In learning feedback policies, comparing the results obtained by the guided policy search (GPS) [98] and PDP in Fig. 3.8 and in Fig. 3.11, we have the following remarks.

(1) PDP outperforms GPS in terms of having lower control loss (cost). This can be seen in Fig. 3.8 and Fig. 3.11 (in Fig. 3.11, PDP results in a simulated trajectory which is closer to the optimal one than that of GPS). This can be understood from the fact that GPS considers the policy as constraint and updates it in a supervised learning step during the learning process. Although GPS aims to *simultaneously* minimize the control cost and the degree to which the policy is violated, it does not necessarily mean that before the learning researches convergence, when *strictly following* a pre-convergence control policy, the system will have a cost as minimal as it can possibly achieve.

(2) Instead, PDP adopts a different way to synchronize the fulfillment of policy constraints and the minimization of the control cost. In fact, throughout the entire learning process, PDP always guarantees that the policy constraint is perfectly respected (as the forward pass strictly follows the policy). Therefore, the core difference between PDP and GPS is that PDP does not simultaneously minimize two aspects—the policy violation and control cost, instead, it enforces that one aspect—policy—is always respected and only focuses on minimizing the other—control cost. The benefit of doing so is that at each learning step, the control cost for PDP is always as minimal as it can possibly achieve. This explains why PDP outperforms GPS in terms of having lower control cost (loss).

3.10.5 Experiment of Rocket Powered Landing Problems

As a final part in experiments, we will demonstrate the capability of PDP to solve the more challenging 6-DoF rocket powered landing problems.

We here omit the description of mechanics modeling for the 6-DoF powered rocket system, and refer the reader to Page 5 in [137] for the rigid body dynamics model of a rocket system

(the notations and coordinates used below follows the ones in [137]). The state vector of the rocket system is defined as

$$\boldsymbol{x} = \begin{bmatrix} m & \boldsymbol{r}_{\mathcal{I}}' & \boldsymbol{v}_{\mathcal{I}}' & \boldsymbol{q}_{\mathcal{B}/\mathcal{I}}' & \boldsymbol{\omega}_{\mathcal{B}}' \end{bmatrix}' \in \mathbb{R}^{14}, \tag{3.38}$$

where $m \in \mathbb{R}$ is the mass of the rocket; $\boldsymbol{r}_{\mathcal{I}} \in \mathbb{R}^3$ and $\boldsymbol{v}_{\mathcal{I}} \in \mathbb{R}^3$ are the position and velocity of the rocket (center of mass) in the inertially-fixed Up-East-North coordinate frame; $\boldsymbol{q}_{\mathcal{B}/\mathcal{I}} \in \mathbb{R}^4$ is the unit quaternion denoting the attitude of rocket body frame with respect to the inertial frame (also see the description in the quadrotor dynamics in Appendix 3.10.1); and $\boldsymbol{\omega}_{\mathcal{B}} \in \mathbb{R}^3$ is the angular velocity of the rocket expressed in the rocket body frame. In our simulation, we only focus on the final descending phase before landing, and thus assume the mass depletion during such a short phase is very slow and thus $\dot{m} \approx 0$. We define the control input vector of the rocket, which is the thrust force vector

$$\boldsymbol{u} = \boldsymbol{T}_{\mathcal{B}} = [T_x, T_y, T_z]' \in \mathbb{R}^3, \tag{3.39}$$

acting on the gimbal point of the engine (situated at the tail of the rocket) and is expressed in the body frame. Note that the relationship between the total torque $\boldsymbol{M}_{\mathcal{B}}$ applied to the rocket and the thrust force vector $\boldsymbol{T}_{\mathcal{B}}$ is $\boldsymbol{M}_{\mathcal{B}} = \boldsymbol{r}_{\mathcal{I},\mathcal{B}} \times \boldsymbol{T}_{\mathcal{B}}$, with $\boldsymbol{r}_{\mathcal{I},\mathcal{B}} \in \mathbb{R}^3$ being constant position vector from the center-of-mass of the rocket to the gimbal point of the engine. The continuous dynamics is discretized using the Euler method: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \Delta \cdot \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ with the discretization interval $\Delta = 0.1$s.

For the rocket system, the unknown dynamics parameter, $\boldsymbol{\theta}_{\text{dyn}}$, includes the rocket's initial mass m_0, and the moment of inertia $\boldsymbol{J}_{\mathcal{B}} \in \mathbb{R}^{3\times3}$, and the rocket length ℓ, thus, $\boldsymbol{\theta}_{\text{dyn}} = \{m_0, \boldsymbol{J}_{\mathcal{B}}, \ell\} \in \mathbb{R}^8$.

For the control objective (cost) function, we consider a weighted combination of the following aspects:

- distance of the rocket position from the target position, with weight w_1;
- distance of the rocket velocity from the target velocity, with weight w_2;

- penalty of the excessive title angle of the rocket, with weight w_3;
- penalty of the side effects of the thrust vector, with weight w_4;
- penalty of the total fuel cost, with weighted w_5.

So the parameter of the control objective function, $\boldsymbol{\theta}_{\text{obj}} = \begin{bmatrix} w_1, & w_2, & w_3, & w_4, & w_5 \end{bmatrix}' \in \mathbb{R}^5$. In sum, the overall parameter for the 6-DoF rocket powered landing control system is

$$\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\text{dyn}}, \ \boldsymbol{\theta}_{\text{obj}}\} \in \mathbb{R}^{13}. \tag{3.40}$$

Imitation Learning. We apply the IRL/IOC mode of PDP to perform imitation learning of the 6-DoF rocket powered landing. The experiment process is similar to the experiments in Appendix 3.10.2, where we collect five trajectories from an expert system with dynamics and control objective function both known (different trajectories have different time horizons T ranging from 40 to 50 and different initial state conditions). Here we minimize imitation loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \|\boldsymbol{\xi}^{\text{d}} - \boldsymbol{\xi}_{\boldsymbol{\theta}}\|^2$ over the parameter of dynamics and control objective, $\boldsymbol{\theta}$ in (3.40). The learning rate is set to $\eta = 10^{-4}$, and we run five trials with random initial parameter guess $\boldsymbol{\theta}_0$. The imitation loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ versus iteration is plotted in Fig. 3.12a. To validate the learned models (the learned dynamics and the learned objective function), we use the learned models to perform motion planing of rocket powered landing in unseen settings (here we use new initial condition and new time horizon). The planing results are plotted in Fig. 3.12b, where we also plot the ground truth for comparison.

System Identification. We apply the SysID mode of PDP to identify the dynamics parameter $\boldsymbol{\theta}_{\text{dyn}}$ of the rocket. The experiment process is similar to the experiments in Appendix 3.10.3, where we collect five trajectories with different initial state conditions, time horizons (T ranges from 10 to 20), and random control inputs. We minimize the SysID loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \|\boldsymbol{\xi}^{\text{o}} - \boldsymbol{\xi}_{\boldsymbol{\theta}}\|^2$ over $\boldsymbol{\theta}_{\text{dyn}}$ in (3.40). The learning rate is set to $\eta = 10^{-4}$, and we run five trials with random initial parameter guess for $\boldsymbol{\theta}_{\text{dyn}}$. The SysID loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ versus iteration is plotted in Fig. 3.13a. To validate the learned dynamics, we use it to predict the motion of rocket given a new sequence of control inputs. The prediction results are in Fig. 3.13b, where we also plot the ground truth for reference.

Optimal Powered Landing Control. We apply the Control/Planning mode of PDP to find an optimal control sequence for the rocket to perform a successful powered landing. The experiment process is similar to the experiments performed for the quadrotor system in Appendix 3.10.4. We set the time horizon as $T = 50$, and randomly choose an initial state condition $\boldsymbol{x}_0$ for the rocket. We minimize the control loss function, which is now a given control objective function with $\boldsymbol{\theta}_{\text{obj}}$ known. The control policy we use here is parameterized as the Lagrangian polynomial, as described in (3.36) in Appendix 3.10.4, here with degree $N = 25$. The control loss is set as the control objective function learned in the previous imitation learning experiment. The learning rate is set to $\eta = 10^{-4}$, and we run five trials with random initial guess of the policy parameter. The the control loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ versus iteration is plotted in Fig. 3.14a. To validate the learned optimal control policy, we use it to simulate the motion (control trajectory) of the rocket landing, and compare with the ground truth optimal trajectory obtained by an OC solver. The validation results are in Fig. 3.14b.

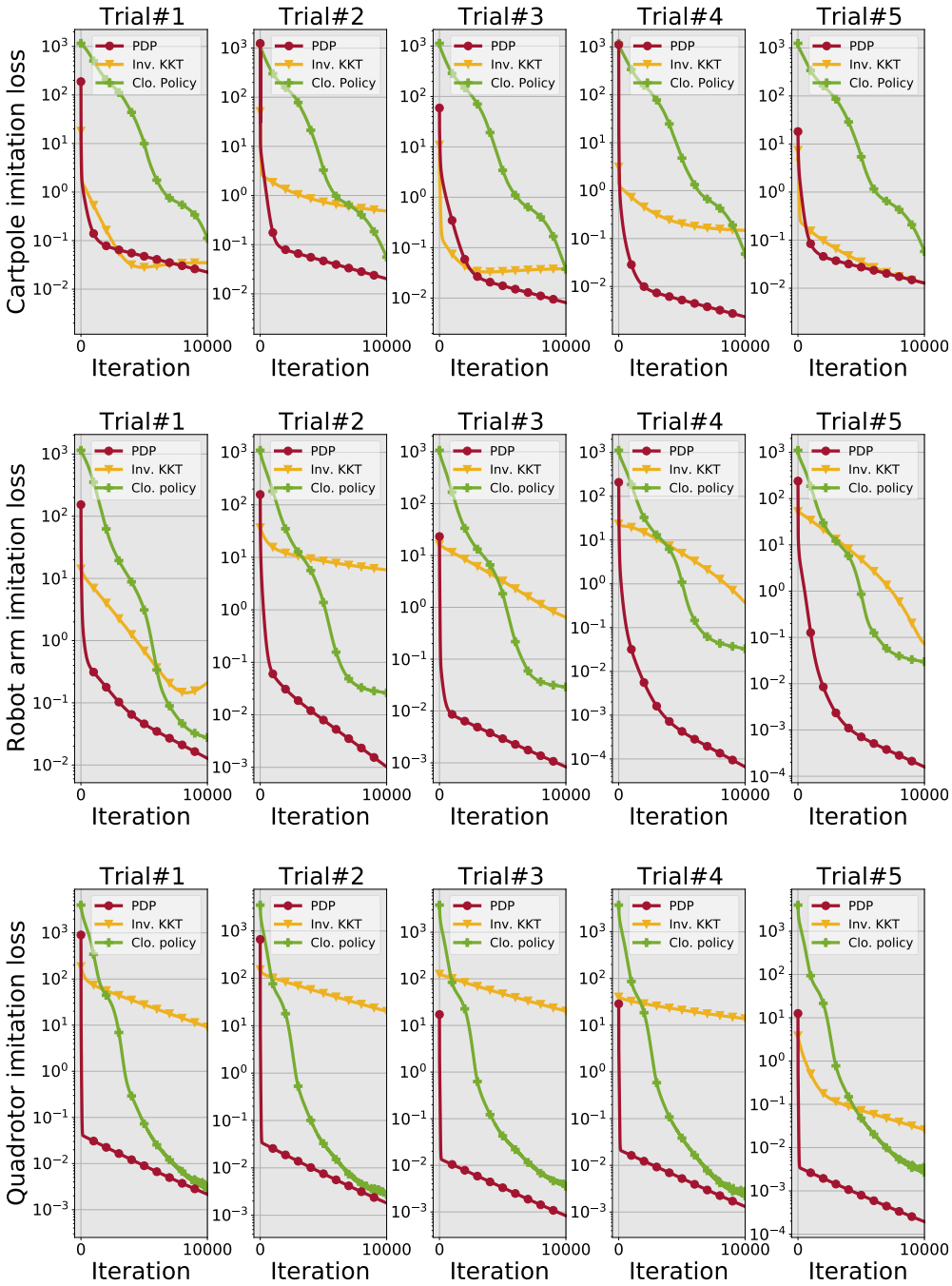


Figure 3.6. Experiments for PDP IRL/IOC Mode: imitation loss versus iteration. For each system, we run five trials starting with random initial guess $\boldsymbol{\theta}_0$, and the learning rate is $\eta = 10^{-4}$ for all methods. The results show a significant advantage of the PDP over the neural policy cloning and inverse-KKT [34] in terms of lower training loss and faster convergence speed. Please see Appendix Fig. 3.9 for validation. Please find the video demo at https://youtu.be/awVNiCIJCfs.

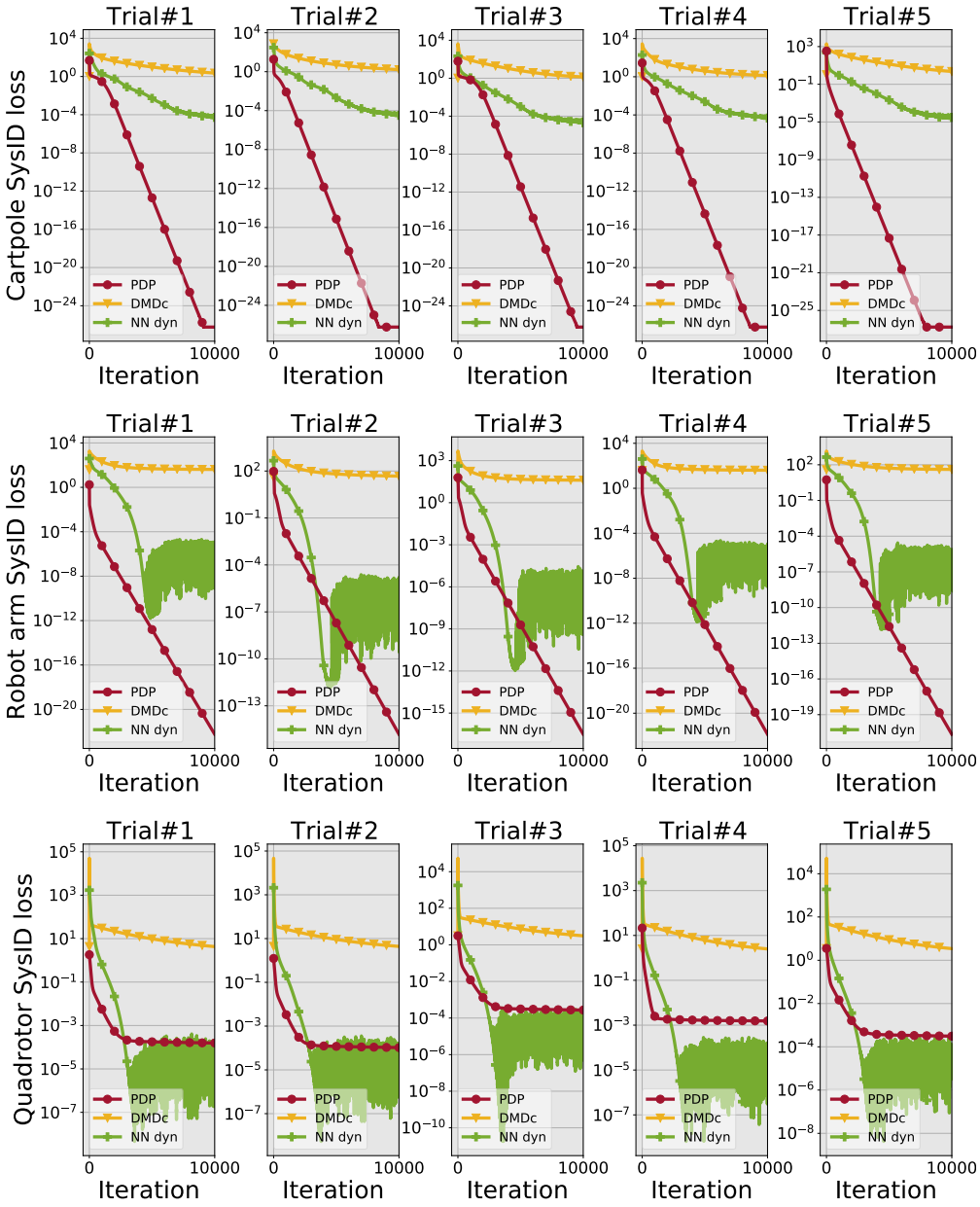


Figure 3.7. Experiments for PDP SysID Mode: SysID loss versus iteration. For each system, we run five trials with random initial guess $\boldsymbol{\theta}_0$, and set the learning rate $\eta = 10^{-4}$ for all methods. The results show a significant advantage of the PDP over neural-network dynamics and DMDc in terms of lower training loss and faster convergence speed. Please see Fig. 3.10 for validation. Please find the video demo at https://youtu.be/PAyBZjDD6OY.

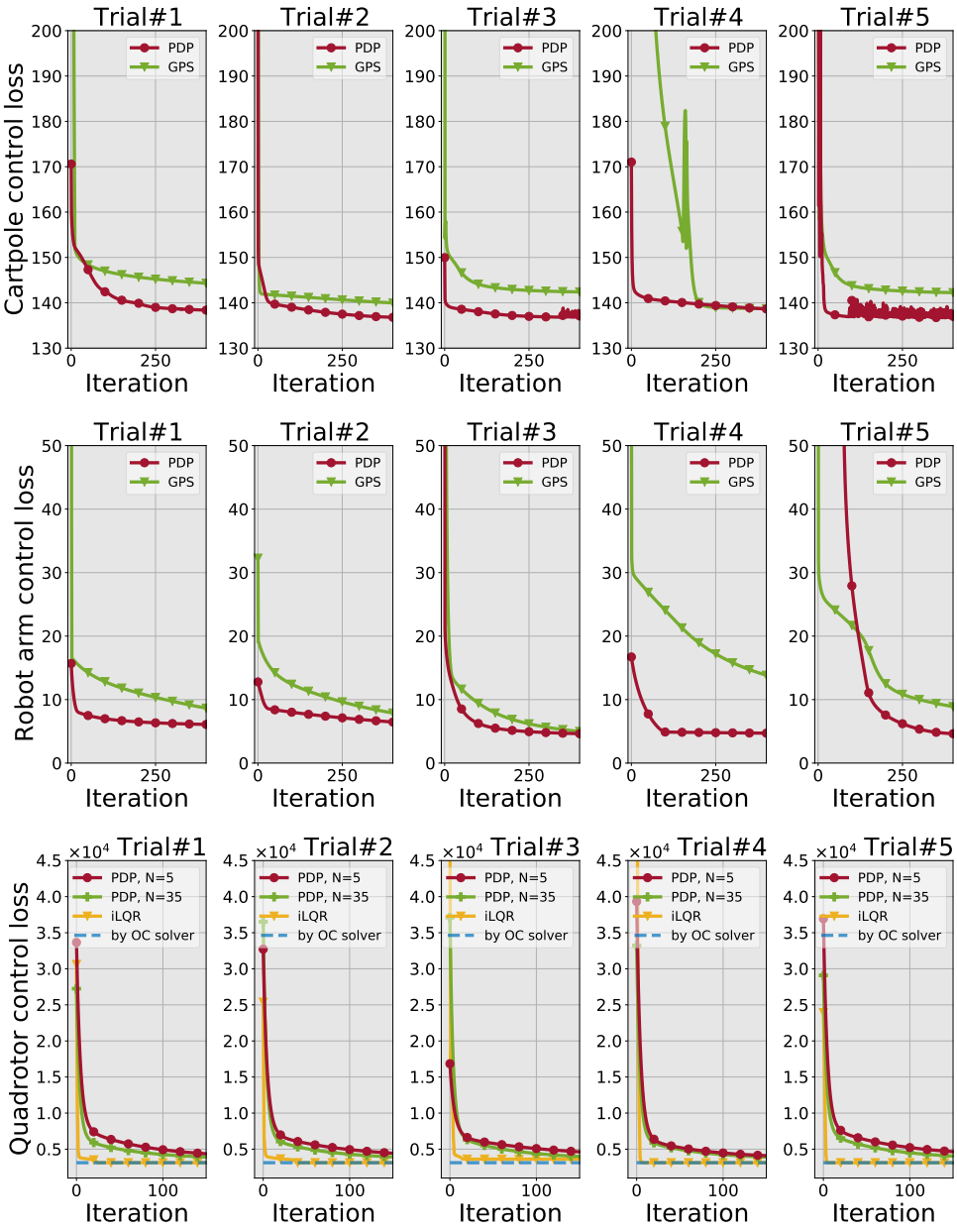
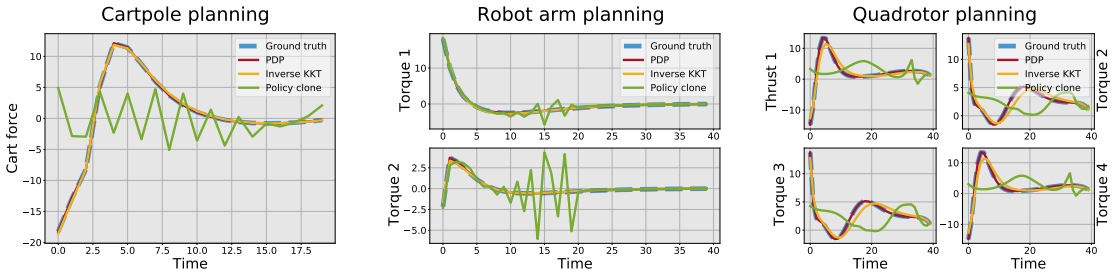
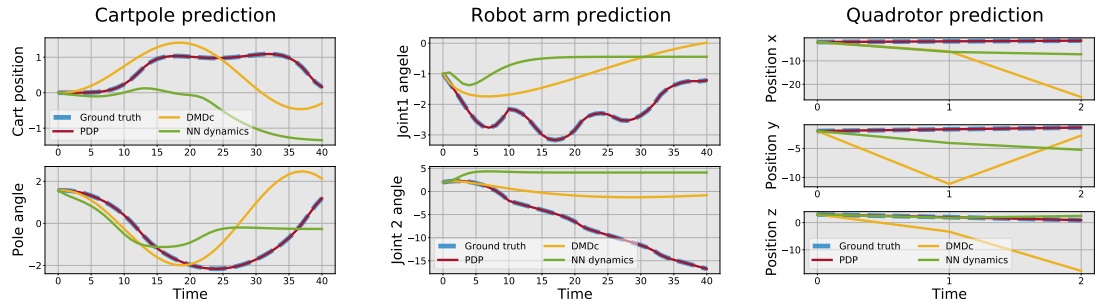


Figure 3.8. Experiments for PDP Control/Planning Mode: control loss (i.e., objective function value) versus iteration. For the cart-pole (top panel) and robot arm (middle panel) systems, we learn neural feedback policies, and compare with the GPS method [98]. For the quadrotor system, we perform motion planning with a Lagrange polynomial policy (we use different degree N), and compare with iLQR and an OC solver [50]. The results show that for learning feedback control policies, PDP outperforms GPS in terms of having lower control loss (cost); and for motion planning, iLQR has faster convergence speed than PDP. Please find the video demo at https://youtu.be/KTw6TAigfPY.



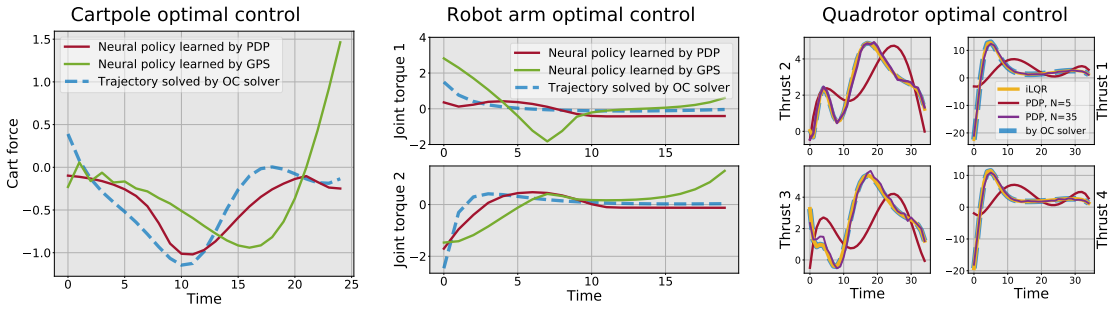
(a) Cart-pole (b) Robot arm (c) Quadrotor

Figure 3.9. Validation for the imitation learning experiment in Fig. 3.6. We preform motion planing for each system in unseen conditions (new initial condition and new time horizon) using the learned models. Results show that compared to the neural policy cloning and inverse KKT [34], PDP result can accurately plan the expert's trajectory in unseen settings. This indicates PDP can accurately learn the dynamics and control objective, and has the better generality than the other two. Although policy imitation has lower imitation loss than inverse KKT, it has the poorer performance in planing. This is because with limited data, the cloned policy can be over-fitting, while the inverse KKT learns a cost function, a high-level representation of policies, thus has better generality to unseen conditions.



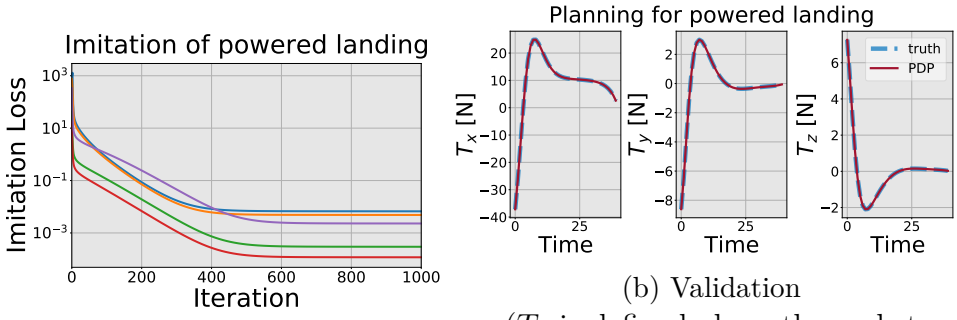
(a) Cart-pole (b) Robot arm (c) Quadrotor

Figure 3.10. Validation for the system identification experiment in Fig. 3.7. We perform motion prediction in unactuated conditions ($\boldsymbol{u} = 0$) using the learned dynamics. Results show that compared to neural-network dynamics training and DMDc, PDP can accurately predict the motion trajectory of each systems. This indicates the effectiveness of the PDP in identifying dynamics models.



(a) Cart-pole (b) Robot arm (c) Quadrotor

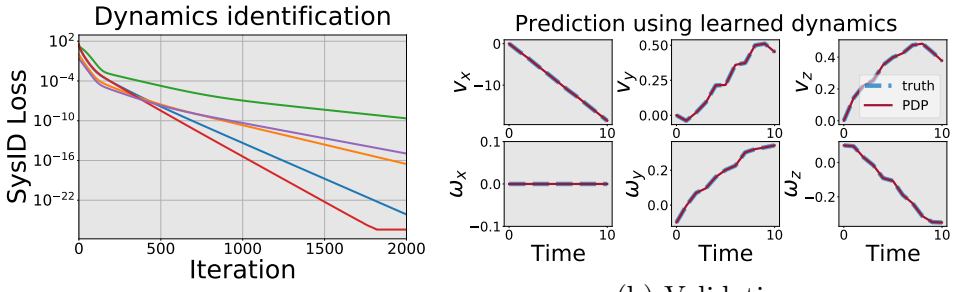
Figure 3.11. Simulation of the learned policies in the control and planning experiment in Fig. 3.8. Fig. 3.11a-3.11b are the simulations of the learned neural feedback policies on the cart-pole and robot arm systems, respectively, where we also plot the optimal trajectory solved by an OC solver [50] for reference. From Fig. 3.11a-3.11b, we observe that PDP results in a trajectory that is much closer to the optimal one than that of GPS; this implies that PDP has lower control loss (please check our analysis on this in Appendix 3.10.4) than GPS. Fig. 3.11c is the planning results for the quadrotor system using PDP, iLQR, and an OC solver [50], where we have used different degrees of Lagrange polynomial policies in PDP. The results show that PDP can successfully plan a trajectory very close to the ground truth optimal trajectory. We also observe that the accuracy of the resulting trajectory depends on choice of the policy parameterization (i.e., expressive power): for example, the use of polynomial policy of a higher degree N results in a trajectory closer to the optimal one (the one using the OC solver) than the use of a lower degree. iLQR is generally able to achieve high-accuracy solutions because it directly optimizes the loss function with respect to individual control inputs (instead of a parameterized policy), but this comes at the cost of high computation expense, as shown in Fig. 3.4d.



(a) Training

(b) Validation (T_x is defined along the rocket direction)

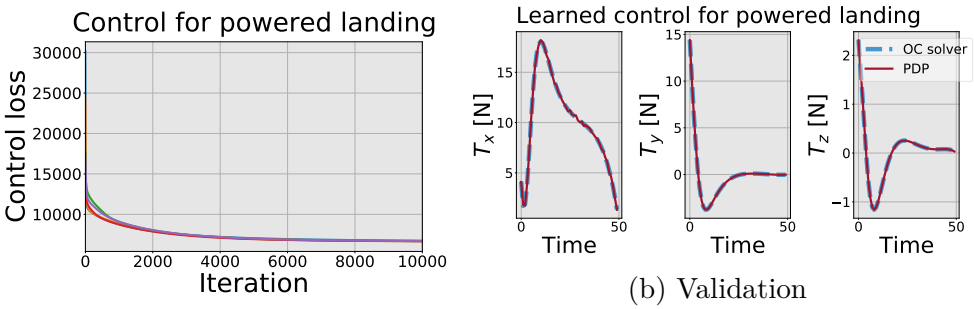
Figure 3.12. (a) Training process for imitation learning of 6-DoF rocket powered landing: the imitation loss versus iteration; here we have performed five trials (labeled by different colors) with random initial parameter guess. (b) Validation: we use the learned models (dynamics and control objective function) to perform motion planning of the rocket powered landing in unseen settings (i.e. given new initial state condition and new time horizon requirement); here we also plot the ground-truth motion planning of the expert for reference. The results in (a) and (b) show that the PDP can accurately learn the dynamics and control objective function from demonstrations, and have good generalizability to novel situations. Please find the video demo at https://youtu.be/4RxDLxUcMp4.



(a) Training

(b) Validation

Figure 3.13. (a) Training process for identification of rocket dynamics: SysID loss versus iteration; here we have performed five trials (labeled by different colors) with random initial parameter guess. (b) Validation: we use the learned dynamics model to perform motion prediction of the rocket given a new control sequence; here we also plot the ground-truth motion (where we know the exact dynamics). The results in (a) and (b) show that the PDP can accurately identify the dynamics model of the rocket.



(a) Training

(b) Validation (T_x is defined along the rocket)

Figure 3.14. (a) Training process of learning the optimal control policy for rocket powered landing: the control loss versus iteration; here we have performed five trials (labeled by different colors) with random initial guess of the policy parameter. (b) Validation: we use the learned policy to simulate the rocket control trajectory; here we also plot the ground-truth optimal control solved by an OC solver. The results in (a) and (b) show that the PDP can successfully find the optimal control policy (or optimal control sequence) to successfully perform the rocket powered landing. Please find the video demo at https://youtu.be/5Jsu772Sqcg.

PART II

LEARNING WITH HUMAN-ON-THE-LOOP

4. LEARNING FROM SPARSE DEMONSTRATIONS

Starting from this chapter, we focus on the robot learning with human guidance. To boost efficiency of robot learning while maintaining high-level autonomy, we aim to develop the innovative robot learning paradigms that make the most use of human guidance while maintaining the human burden as low as possible in providing such guidance.

In this chapter, we focus on human guidance in the form of behavioral demonstrations. We will develop a new method which allows a robot to learn an objective function from human's sparse demonstrations. The sparse demonstrations are given by a small number of sparse waypoints, which are desired outputs of the robot's trajectory at certain time instances, sparsely located within a demonstration time horizon. The proposed method learns an objective function by directly minimizing a trajectory loss, which quantifies the discrepancy between a robot's reproduced trajectory and the observed sparse demonstrations. The content of this chapter appears in [138], and the code and experiments for this chapter can be accessed at https://github.com/wanxinjin/Learning-from-Sparse-Demonstrations.

4.1 Introduction

The appeal of learning from demonstrations (LfD) lies in its capability to facilitate robot programming by simply providing demonstrations from an expert. It circumvents the need for expertise in controller design and coding, which is required by traditional robot programming, and empowers non-experts to program a robot as needed [139]. LfD has been successfully applied to various scenarios such as manufacturing [140], assistive robots [141], and autonomous vehicles [26].

LfD techniques can be broadly categorized based on what to learn from the observed demonstrations. A branch of LfD focuses on learning a policy [142, 143, 144, 145, 146], which directly maps from the robot's states, environment, or raw observation information to the robot's actions, based on supervised machine learning techniques. While effective in many situations, policy learning typically requires a considerable amount of demonstration data, and the learned policy may generalize poorly to unseen or long horizon tasks [139]. To alleviate this, another direction of LfD research focuses on learning control objective (e.g.,

cost or reward) functions from demonstrations [32], based on which the optimal policies or trajectories are derived. These methods assume the optimality of demonstrations and use inverse reinforcement learning (IRL) [3] or inverse optimal control (IOC) [147] to estimate the control objective function. Since an objective function is a more compact and high-level representation of a task, LfD via learning objective functions has demonstrated advantage over policy learning in terms of better generalization to unseen situations [148] and relatively lower sample complexity [32]. Despite significant progress along this direction, LfD based on objective learning still inherits some limitations from the core IRL and IOC techniques, which are summarized below.

(i) Most IOC/IRL techniques require entire demonstrations of a complete task [22, 29, 33, 34, 40, 42]. Such requirements make it challenging to collect demonstration data, especially obtaining demonstration of high degree-of-freedom systems such as humanoid robots.

(ii) The majority of existing IOC and IRL methods [22, 29, 33, 34, 40, 42] assume an objective function as a *linear combination* of selected features, and their algorithms are designed in the *feature space* by taking advantage of the *linearity* of the feature weights [12]. Those approaches typically do not directly minimize the discrepancy between the robot's reproduced trajectory and the demonstrations in *trajectory space*, and cannot be readily extended to non-linear parameterization of an objective function.

(iii) There might exist time-scale discrepancy between expert's demonstrations and the actual actuation of the robot [149]. For instance, consider a robot that learns from human motion. The duration of the human demonstration may not reflect the dynamics constraint of a robot, as the robot may be actuated by a weak servo motor and cannot move as fast as the human.

In recognition of these limitations, in this chapter we present a new method to learn from sparse demonstrations, which has the following advantages over existing methods:

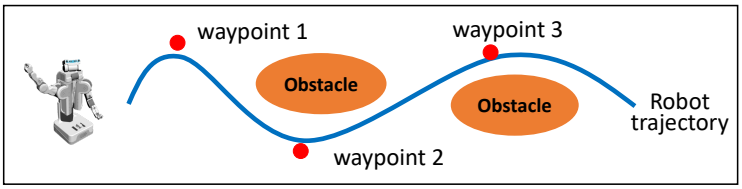


Figure 4.1. Illustration of learning from sparse demonstrations. The red dots are the expert's sparse demonstration waypoints, from which the robot learns a control objective function such that its reproduced trajectory (blue line) is closest to these waypoints. At first sight, the depicted robot's reproduced trajectory (blue line) may seem a result of using 'curve fitting' method (which inherently belongs to policy learning methods); however, a key difference from 'curve fitting' is that the robot here learns a control objective function instead of imitating a trajectory, and the learned control objective function is generalizable to unseen situations, such as new initial conditions or longer time horizons. Please find video demos at https://wanxinjin.github.io/posts/lfsd.

- First, the proposed method learns an objective function using only sparse demonstration data, which consists of a small number of desired outputs of the robot's trajectory at some sparse time instances within a time horizon, as shown in Fig. 4.1. The given sparse demonstrations do not necessarily contain control input information.

- Second, the proposed method learns an objective function over a parameterized function set by directly minimizing the distance between the robot's reproduced trajectory and the sparse demonstrations. Even though the demonstrations may not correspond to an exact objective function within the parameterized function set, e.g., demonstrations are not optimal or even randomly given, the method can still find a 'best' objective function within the parameterized function set such that the reproduced trajectory is closest to the given demonstrations in Euclidean distance, as shown in Fig. 4.1.

- Third, since the time requirement associated with the sparse waypoints may not be achievable with the robot actuation, in addition to learning an objective function, the proposed method jointly learns a time-warping function, which maps from

the demonstration time axis to the robot execution time axis. This addresses the potential issue of time misalignment for existing IOC/IRL methods.

4.1.1 Background and Related Work

Since the theme of the method devised in this chapter belongs to the category of LfD based on learning objective functions, here we mainly focus on the related work of IRL/IOC methods, which share the same goal of learning objective functions from demonstrations. For the other types of LfD methods, e.g., policy learning, or the comparison between them, we refer the reader to recent surveys [139] and [150] for more details.

Over the past decades, various IRL and IOC techniques have been proposed, with different work emphasizing different formulations to infer an objective function. The early IRL/IOC techniques include feature matching [32], maximum margin [33], and maximum entropy [29]. Recently, a new line of IOC/IRL research [11, 12, 34, 40, 42] directly solves for the objective function parameters by establishing optimality conditions, such as Karush-Kuhn-Tucker conditions [151] or Pontryagin's maximum principle [44]. The key idea is that the demonstration data is assumed to be optimal and thus must satisfy the optimality conditions. By directly minimizing the violation of the optimality conditions by the demonstration data over objective function parameters, one can obtain an estimate of the objective function. The benefits of doing so is that these methods can avoid repetitive solving of the direct optimal control or reinforcement learning problems in each iteration.

All the above IOC and IRL techniques assume a linear combination of selected features as their parameterized objective functions with unknown feature weights. Learning objective functions is not formulated on a *trajectory space*, that is, they do not directly minimize discrepancy between the reproduced trajectory and demonstrations. Instead, they design algorithms in the selected feature space by taking advantage of the *linearity* of the feature weights. For example, the maximum margin IRL [33] and feature matching IRL [32] focus on maximizing and equaling the feature values between the given demonstrations and the reproduced trajectories, respectively. While the recent work in [152] formulates the IOC problem as minimization of direct loss, the algorithm is still similar to the maximum margin

approach in a selected-feature space. In [22], the authors use a double-layer optimization to solve the IOC problem and directly minimize a trajectory loss function. In the upper layer of updating the objective function, a derivative-free method [153] is used by approximating the loss function with a quadratic function. This involves multiple evaluations of the loss and thus requires solving the optimal control problem repetitively in each update, which is computationally expensive. Also, the derivative-free method has inherent limitations for handling problems of large size [154]. For the second line of IOC methods which solve the feature weights by minimizing the violation of the optimality conditions, they still indirectly consider trajectory error.

Learning objective functions in a linear feature space may facilitate the design of learning algorithms (such as by taking advantage of linearity of feature weights), but their performance relies on the choice of features. While many IOC approaches [34, 40, 42] assume the optimality of demonstration data; that is, the observed demonstrations are a result of optimizing the parameterized objective functions, this assumption is subject to observation noise and good feature selection.

The other challenges of existing IOC and IRL techniques are listed below. First, existing methods require as input the continuous demonstration data of an entire task; in other words, a given demonstration needs to be a complete trajectory over the entire course of execution time. Thus, demonstration data needs to be carefully collected from an expert, which can be burdensome especially for high-dimensional systems. Instead, it is relatively easier to provide only sparse demonstrations. Although [12] proposes a method to solve IOC from incomplete trajectory data, it still requires a trajectory segment to be long enough to satisfy a recovery condition and thus cannot handle very sparse demonstrations as shown in Fig. 4.1. In [155], the authors develop a method for learning from keyframe demonstrations. This method is a policy learning technique: it learns a kinematic trajectory model (Gaussian mixture models) instead of learning an objective function. The unseen motion between keyframes is handled by interpolation. Such a process leads to poor generalization and high sample complexity (we will show this later in experiments). Another limitation of existing IOC and IRL methods is that they rarely account for the time misalignment between the demonstrations and the feasible actuation capabilities of a robot. This is critical in practical implementation. For

example, consider a humanoid robot that learns to imitate a human demonstrator. The robot may be actuated by a weak servo motor which may not move as fast as human. The demonstrations thus cannot be directly used for objective function learning. To address this, [149] learns a time-warping function between a robot and a demonstrator, but this method is used to align the time of a demonstrated trajectory for optimal tracking instead of learning objective functions.

4.1.2 Contributions

We propose a new approach to learn objective functions from sparse demonstrations. The contributions of the method relative to existing IRL/IOC methods are listed below.

(1) The proposed method learns an objective function by directly minimizing a trajectory loss, which quantifies the discrepancy between a robot's reproduced trajectory and the observed demonstrations. Different from [22] using derivative-free techniques [153], the proposed approach is a *gradient based* optimization method, which can handle high-dimensional systems.

(2) The proposed method accepts a general parameterization of objective functions (e.g., nonlinear in function parameters such as neural networks), which is not necessarily a linear combination of features. The algorithm finds an objective function within the given function set such that the reproduced trajectory has minimum Euclidean distance to demonstrations, even though the demonstrations may not be optimal and the exact corresponding objective function does not exist in the function set.

(3) The learning algorithm permits sparse demonstrations, which consists of a small number of desired outputs of the robot's trajectory at sparse time instances. The algorithm will find an objective function such that the reproduced trajectory gets closest to the given waypoints in Euclidean distance. In addition to learning the objective function, the method jointly learns a time-warping function to align the duration between the expert's demonstration and the feasible motion of the robot.

The organization of this chapter is as follows: Section 4.2 formulates the problem. Section 4.3 discusses the time-warping technique and reformulates the problem under a unified time axis. Section 4.4 proposes the learning algorithm. Experiments are provided in Sections 4.5 and 4.6. Section 4.7 gives discussion to the method, and finally Section 4.8 draws conclusions.

4.2 Problem Formulation

Consider a robot with the following continuous dynamics:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \quad \text{with} \quad \boldsymbol{x}(0), \tag{4.1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the robot state; $\boldsymbol{u}(t) \in \mathbb{R}^m$ is the control input; vector function $\boldsymbol{f} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ is assumed to be twice-differentiable, and $t \in [0, \infty)$ is time. Suppose the robot motion over a time horizon $t_f > 0$ is controlled by optimizing the following parameterized objective function:

$$J(\boldsymbol{p}) = \int_0^{t_f} c(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}) dt + h(\boldsymbol{x}(t_f), \boldsymbol{p}), \tag{4.2}$$

where $c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p})$ and $h(\boldsymbol{x}, \boldsymbol{p})$ are the running and final costs, respectively, both of which are assumed twice-differentiable; and $\boldsymbol{p} \in \mathbb{R}^r$ is a tunable parameter vector. For a fixed choice of $\boldsymbol{p}$, the robot produces a trajectory of states and inputs

$$\boldsymbol{\xi}_{\boldsymbol{p}} = \{\boldsymbol{\xi}_{\boldsymbol{p}}(t) \,|\, 0 \leq t \leq t_f\} \text{ with } \boldsymbol{\xi}_{\boldsymbol{p}}(t) = \{\boldsymbol{x}_{\boldsymbol{p}}(t), \boldsymbol{u}_{\boldsymbol{p}}(t)\}. \tag{4.3}$$

which optimizes the objective function (4.2). Here the subscript in $\boldsymbol{\xi}_{\boldsymbol{p}}$ indicates that the trajectory implicitly depends on $\boldsymbol{p}$.

The goal of learning from demonstrations is to estimate the objective function parameter $\boldsymbol{p}$ based on the observed demonstrations of an expert (usually a human operator). Here, we suppose that an expert provides demonstrations through a known output function

$$\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}), \tag{4.4}$$

where $\boldsymbol{g} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^o$ defines a map from the robot's state and input to an output $\boldsymbol{y} \in \mathbb{R}^o$. The expert's demonstrations include (i) an *expected* time horizon T, and (ii) a number of N waypoints, each of which is a desired output for the robot to reach at an *expected* time instance, denoted as

$$\mathcal{D} = \{\boldsymbol{y}^*(\tau_i) \,|\, \tau_i \in [0, T],\, i = 1, 2, \cdots, N\}. \tag{4.5}$$

Here, $\boldsymbol{y}^*(\tau_i)$ is the ith waypoint demonstrated by the expert, and τ_i is the expected time instance at which the expert wants the robot to reach the waypoint $\boldsymbol{y}^*(\tau_i)$. As the expert can freely provide the number of N waypoints and choose the positions of expected time instances τ_i relative to the expected horizon T, we refer to $\mathcal{D}$ as *sparse demonstrations*. As will be shown later in simulations, N here can be small.

Note that both the expected time horizon T and the expected time instances τ_i are in the time axis of the expert's demonstrations. This demonstration time axis may not be identical to the actual time axis of execution of the robot; in other words, the given times T and τ_i may not be achievable by the robot. For example, when the robot is actuated by a weak servo motor, its motion inherently cannot meet the time step τ_i required by a human demonstrator. To accommodate the misalignment of duration between the robot and expert's demonstrations, we introduce a *time warping function*

$$t = w(\tau), \tag{4.6}$$

which defines a map from the expert's demonstration time axis τ to the robot time axis t. We make the following reasonable assumption: w is strictly increasing for the range of $[0, T]$ and continuously differentiable function with $w(0) = 0$.

Given the sparse demonstrations $\mathcal{D}$, **the problem of interest** is to find an objective function parameter $\boldsymbol{p}$ and a time-warping function w such that the following trajectory loss is minimized:

$$\min_{\boldsymbol{p}, w} \sum_{i=1}^{N} l\Big(\boldsymbol{y}^*(\tau_i), \boldsymbol{g}\big(\boldsymbol{\xi}_{\boldsymbol{p}}(w(\tau_i))\big)\Big), \tag{4.7}$$

where $l(\boldsymbol{a}, \boldsymbol{b})$ is a given differentiable scalar function to quantify a point distance metric between vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, e.g., $l(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|^2$. Minimizing the loss in (4.7) means that we want the robot to find the 'best' objective function within the parameterized objective function set (4.2), together with a time-warping function, such that its reproduced trajectory is as close to the given sparse demonstrations as possible.

4.3 Problem Reformulation by Time Warping

In this section, we present the parameterization of the time-warping function, and then re-formulate the problem of interest presented in the previous section under a unified time axis.

4.3.1 Parametric Time Warping Function

To facilitate learning of an unknown time-warping function, we parameterize the time-warping function. Suppose that a differentiable time-warping function $w(\tau)$ satisfies $w(0) = 0$ and is strictly increasing in the range $[0, T]$. Then the derivative

$$v(\tau) = \frac{dw(\tau)}{d\tau} > 0 \tag{4.8}$$

for all $\tau \in [0, T]$. We use a polynomial time-warping function:

$$t = w_{\boldsymbol{\beta}}(\tau) = \sum_{i=1}^{s} \beta_i \tau^i, \tag{4.9}$$

where $\boldsymbol{\beta} = [\beta_1, \beta_2, \cdots, \beta_s]' \in \mathbb{R}^s$ is the coefficient vector of the polynomial. Since $w_{\boldsymbol{\beta}}(0) = 0$, there is no constant (zero-order) term in (4.9) (i.e., $\beta_0 = 0$). Due to the requirement $dw_{\boldsymbol{\beta}}/d\tau = v_{\boldsymbol{\beta}}(\tau) > 0$ for all $\tau \in [0, T]$ in (4.8), one can always obtain a feasible (e.g. compact) set for $\boldsymbol{\beta}$, denoted as $\Omega_{\boldsymbol{\beta}}$, such that $\frac{dw_{\boldsymbol{\beta}}(\tau)}{d\tau} > 0$ for all $\tau \in [0, T]$ if $\boldsymbol{\beta} \in \Omega_{\boldsymbol{\beta}}$.

4.3.2 Equivalent Formulation under a Unified Time Axis

Substituting the parametric time-warping function $w_{\boldsymbol{\beta}}$ in (4.9) into both the robot's dynamics (4.1) and the control objective function (4.2), we obtain the following time-warped dynamics

$$\frac{d\boldsymbol{x}}{d\tau} = \frac{dw_{\boldsymbol{\beta}}}{d\tau}\boldsymbol{f}\Big(\boldsymbol{x}(w_{\boldsymbol{\beta}}(\tau)), \boldsymbol{u}(w_{\boldsymbol{\beta}}(\tau))\Big) \quad \text{with } \boldsymbol{x}(0), \tag{4.10}$$

and the time-warped objective function

$$J(\boldsymbol{p}, \boldsymbol{\beta}) = \int_0^T \frac{dw_{\boldsymbol{\beta}}}{d\tau} c_{\boldsymbol{p}}(\boldsymbol{x}(w_{\boldsymbol{\beta}}(\tau)), \boldsymbol{u}(w_{\boldsymbol{\beta}}(\tau)))d\tau + h_{\boldsymbol{p}}(\boldsymbol{x}(w_{\boldsymbol{\beta}}(T))). \tag{4.11}$$

Here, the left side of (4.10) is due to chain rule: $\frac{d\boldsymbol{x}}{d\tau} = \dot{\boldsymbol{x}}\frac{dt}{d\tau}$, and the time horizon satisfies $t_f = w_{\boldsymbol{\beta}}(T)$ (note that T is specified by the expert). For notation simplicity, we write $\frac{dw_{\boldsymbol{\beta}}}{d\tau} = v_{\boldsymbol{\beta}}(\tau)$, $\boldsymbol{x}(w(\tau)) = \boldsymbol{x}(\tau)$, $\boldsymbol{u}(w(\tau)) = \boldsymbol{u}(\tau)$, and $\frac{d\boldsymbol{x}}{d\tau} = \dot{\boldsymbol{x}}(\tau)$. Then, the above time-warped dynamics (4.10) and time-warped objective function (4.11) are rewritten as:

$$\dot{\boldsymbol{x}}(\tau) = v_{\boldsymbol{\beta}}(\tau)\boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau)) \quad \text{with } \boldsymbol{x}(0) \tag{4.12a}$$

and

$$J(\boldsymbol{p}, \boldsymbol{\beta}) = \int_0^T v_{\boldsymbol{\beta}}(\tau)c(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{p})d\tau + h(\boldsymbol{x}(T), \boldsymbol{p}), \tag{4.12b}$$

respectively. We concatenate the unknown objective function parameter vector $\boldsymbol{p}$ and unknown time-warping function parameter vector $\boldsymbol{\beta}$ as

$$\boldsymbol{\theta} = [\boldsymbol{p}, \boldsymbol{\beta}]' \in \mathbb{R}^{r+s}. \tag{4.13}$$

For a choice of $\boldsymbol{\theta}$, the time-warped optimal trajectory resulting from solving the above time-warped optimal control system (4.12) is rewritten as

$$\boldsymbol{\xi}_{\boldsymbol{\theta}} = \{\boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau) \mid 0 \leq \tau \leq T\}, \tag{4.14}$$

with $\boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau) = \{\boldsymbol{x}_{\boldsymbol{\theta}}(\tau), \boldsymbol{u}_{\boldsymbol{\theta}}(\tau)\}$. The trajectory distance loss in (4.7) to be minimized can now be defined as

$$L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D}) = \sum_{i=1}^{N} l\Big(\boldsymbol{y}^*(\tau_i),\ \boldsymbol{g}\big(\boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)\big)\Big). \tag{4.15}$$

Minimizing the above loss function in (4.15) over the unknown parameter vector $\boldsymbol{\theta}$ is a process of simultaneously learning the control objective function $J(\boldsymbol{p})$ in (4.2) and the time-warping function $t = w_{\beta}(\tau)$ in (4.9).

In summary, the problem of interest is reformulated as an optimization problem of jointly learning the objective function $J(\boldsymbol{p})$ in (4.2) and time-warping function $t = w_{\beta}(\tau)$ in (4.9):

$$\begin{aligned} &\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \ \ L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D}) \\ &\text{s.t.} \quad \boldsymbol{\xi}_{\boldsymbol{\theta}} \text{ is produced by optimal control system (4.12).} \end{aligned} \tag{4.16}$$

Here $\boldsymbol{\Theta}$ defines a feasible domain of variable $\boldsymbol{\theta}$, $\boldsymbol{\Theta} = \mathbb{R}^r \times \Omega_{\beta}$; the constraint in optimization (4.16) says that $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ is an optimal trajectory generated by the optimal control system (4.12) with the control objective function (4.12b) and dynamics (4.12a). In the next section, we will focus on developing a new learning algorithm to efficiently solve the above optimization problem.

4.4 Proposed Learning Algorithm

4.4.1 Algorithm Overview

To solve the optimization (4.16), we start with an arbitrary initial guess $\boldsymbol{\theta}_0 \in \boldsymbol{\Theta}$, and apply the gradient descent

$$\boldsymbol{\theta}_{k+1} = \text{Proj}_{\boldsymbol{\Theta}}\left(\boldsymbol{\theta}_k - \eta_k \frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}\right), \tag{4.17}$$

where k is the iteration index; η_k is the step size (or learning rate); $\text{Proj}_{\boldsymbol{\Theta}}$ is a projection operator to guarantee the feasibility of $\boldsymbol{\theta}_k$ in each update, e.g., $\text{Proj}_{\boldsymbol{\Theta}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{z} \in \boldsymbol{\Theta}} \|\boldsymbol{\theta} -$

$\boldsymbol{z}\|$; and $\frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$ denotes the gradient of the given loss function (4.15) directly with respect to $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_k$. Applying the chain rule to the gradient term, we have

$$\frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k} = \sum_{i=1}^{N} \frac{\partial l}{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)}\Big|_{\boldsymbol{\xi}_{\boldsymbol{\theta}_k}(\tau_i)} \frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}, \tag{4.18}$$

where $\frac{\partial l}{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)}\Big|_{\boldsymbol{\xi}_{\boldsymbol{\theta}_k}(\tau_i)}$ is the gradient of the single point distance loss defined in (4.15) with respect to the τ_i-time trajectory point, $\boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)$, evaluated at point $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}(\tau_i)$, and $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$ is the gradient of the τ_i-time trajectory point, $\boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)$, with respect to the parameter vector $\boldsymbol{\theta}$ evaluated at value $\boldsymbol{\theta}_k$. From (4.17) and (4.18), we can note that at each iteration k, the update of the parameter $\boldsymbol{\theta}_k$ includes the following three steps:

Step 1: *With the current parameter estimate $\boldsymbol{\theta}_k$, generate the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ in (4.14) by solving the optimal control problem in (4.12);*

Step 2: *Compute the gradients $\frac{\partial l}{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)}\Big|_{\boldsymbol{\xi}_{\boldsymbol{\theta}_k}(\tau_i)}$ and $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$; apply the chain rule (4.18) to compute $\frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$;*

Step 3: *Update $\boldsymbol{\theta}_k$ using (4.17) for the next iteration.*

The interpretation of the above procedure is straightforward: In each update k, first, with the current parameter estimate $\boldsymbol{\theta}_k$, the optimal control system (4.12) produces an optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, and the corresponding trajectory loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}_k}, \mathcal{D})$ (that is, the distance to the given sparse demonstrations) is computed; second, the current gradient of the trajectory loss with respect to $\boldsymbol{\theta}$, $\frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$, is solved; finally, this gradient $\frac{dL}{d\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$ is used to update the current estimate $\boldsymbol{\theta}_k$ for the next iteration $k+1$.

In Step 1 of the learning procedure, the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ for the current parameter estimate $\boldsymbol{\theta}_k$ is solved using any available optimal control solvers such as Casadi [50]. In Step 2, the gradient quantities $\frac{\partial L}{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)}$ can be readily computed by directly differentiating the given trajectory loss function (4.15). The main challenge, however, lies in how to obtain the gradient $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_k}$, that is, the gradient of the system optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ with respect to the parameter $\boldsymbol{\theta}$ for the optimal control system (4.12). In what follows, we will show how

to efficiently compute it by proposing the technique of differential Pontryagin's Maximum Principle. In the following, we suppress the iteration index k for notation simplicity.

4.4.2 Differential Pontryagin's Maximum Principle

Consider the system optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ in (4.14) produced by the optimal control system (4.12) under a fixed choice of $\boldsymbol{\theta}$. The Pontryagin's Maximum Principle [44] states an optimality condition that the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ must satisfy. To present Pontryagin's Maximum Principle, we define the Hamiltonian:

$$H(\tau)=v_\beta(\tau)c_{\boldsymbol{p}}(\boldsymbol{x}(\tau),\boldsymbol{u}(\tau))+\boldsymbol{\lambda}(\tau)'v_\beta(\tau)\boldsymbol{f}(\boldsymbol{x}(\tau),\boldsymbol{u}(\tau)), \tag{4.19}$$

where $\boldsymbol{\lambda}(\tau) \in \mathbb{R}^n$ is called the costate or adjoint variable for $0 \leq \tau \leq T$. According to Pontryagin's Maximum Principle, there exists a costate trajectory

$$\{\boldsymbol{\lambda}_{\boldsymbol{\theta}}(\tau) \,|\, 0 \leq \tau \leq T\}, \tag{4.20}$$

which is associated with the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ in (4.14), such that the following conditions hold:

$$\dot{\boldsymbol{x}}_{\boldsymbol{\theta}}(\tau) = \frac{\partial H}{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}(\boldsymbol{x}_{\boldsymbol{\theta}}(\tau), \boldsymbol{u}_{\boldsymbol{\theta}}(\tau), \boldsymbol{\lambda}_{\boldsymbol{\theta}}(\tau)), \tag{4.21a}$$

$$-\dot{\boldsymbol{\lambda}}_{\boldsymbol{\theta}}(\tau) = \frac{\partial H}{\partial \boldsymbol{x}}(\boldsymbol{x}_{\boldsymbol{\theta}}(\tau), \boldsymbol{u}_{\boldsymbol{\theta}}(\tau), \boldsymbol{\lambda}_{\boldsymbol{\theta}}(\tau)), \tag{4.21b}$$

$$\mathbf{0} = \frac{\partial H}{\partial \boldsymbol{u}}(\boldsymbol{x}_{\boldsymbol{\theta}}(\tau), \boldsymbol{u}_{\boldsymbol{\theta}}(\tau), \boldsymbol{\lambda}_{\boldsymbol{\theta}}(\tau)), \tag{4.21c}$$

$$\boldsymbol{\lambda}_{\boldsymbol{\theta}}(T) = \frac{\partial h_{\boldsymbol{p}}}{\partial \boldsymbol{x}}(\boldsymbol{x}_{\boldsymbol{\theta}}(T)). \tag{4.21d}$$

In fact, given $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ one can always solve the corresponding costate trajectory $\{\boldsymbol{\lambda}_{\boldsymbol{\theta}}(\tau)\}$ by integrating the ODE equation (4.21b) backward in time with the end condition given by (4.21d).

Recall that our technical challenge in the previous part is to obtain the gradient $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$. Towards this goal, we differentiate the above Pontryagin's Maximum Principle equations in

(4.21) on both sides with respect to the parameter $\boldsymbol{\theta}$, which yields the following differential Pontryagin's Maximum Principle

$$\frac{d}{d\tau}(\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}) = F(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + G(\tau)\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + E(\tau), \tag{4.22a}$$

$$-\frac{d}{d\tau}(\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}) = H_{xx}(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{xu}(\tau)\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + F(\tau)'\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{xe}(\tau), \tag{4.22b}$$

$$\mathbf{0} = H_{ux}(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{uu}(\tau)\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + G(\tau)'\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{ue}(\tau), \tag{4.22c}$$

$$\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(T) = H_{xx}(T)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{xe}(T). \tag{4.22d}$$

Here the coefficient matrices in (4.22) are defined as

$$F(\tau)=\frac{\partial^2 H}{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}\partial \boldsymbol{x}_{\boldsymbol{\theta}}}, \quad G(\tau)=\frac{\partial^2 H}{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}\partial \boldsymbol{u}_{\boldsymbol{\theta}}}, \quad E(\tau)=\frac{\partial^2 H}{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}\partial \boldsymbol{\theta}}, \tag{4.23a}$$

$$H_{xx}(\tau)=\frac{\partial^2 H}{(\partial \boldsymbol{x}_{\boldsymbol{\theta}})^2}, \; H_{xu}(\tau)=\frac{\partial^2 H}{\partial \boldsymbol{x}_{\boldsymbol{\theta}}\partial \boldsymbol{u}_{\boldsymbol{\theta}}}, \; H_{xe}(\tau)=\frac{\partial^2 H}{\partial \boldsymbol{x}_{\boldsymbol{\theta}}\partial \boldsymbol{\theta}}, \tag{4.23b}$$

$$H_{ux}(\tau)=H'_{xu}(\tau), \quad H_{uu}(\tau)=\frac{\partial^2 H}{(\partial \boldsymbol{u}_{\boldsymbol{\theta}})^2}, \quad H_{ue}(\tau)=\frac{\partial^2 H}{\partial \boldsymbol{u}_{\boldsymbol{\theta}}\partial \boldsymbol{\theta}}, \tag{4.23c}$$

$$H_{xx}(T)=\frac{\partial^2 h_{\boldsymbol{p}}}{\partial \boldsymbol{x}_{\boldsymbol{\theta}}\partial \boldsymbol{x}_{\boldsymbol{\theta}}}, \; H_{xe}(T)=\frac{\partial^2 h_{\boldsymbol{p}}}{\partial \boldsymbol{x}_{\boldsymbol{\theta}}\partial \boldsymbol{\theta}}. \tag{4.23d}$$

Once we obtain the optimal trajectory $\{\boldsymbol{\xi}_{\boldsymbol{\theta}}\}$ and the associated costate trajectory $\{\boldsymbol{\lambda}_{\boldsymbol{\theta}}(\tau)\}$ in (4.20), all the above coefficient matrices in (4.23) are known and their computation is straightforward. Using these matrices (4.23) and (4.22), the lemma below presents an iterative method to solve the gradient $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau)}{\partial \boldsymbol{\theta}}$.

Lemma 4.4.1. *If $H_{uu}(\tau)$ in (4.23c) is invertible for all $0 \le \tau \le T$, define the following differential equations for matrix variables $P(\tau) \in \mathbb{R}^{n\times n}$ and $W(\tau) \in \mathbb{R}^{n\times(r+s)}$:*

$$-\dot{P} = Q(\tau) + A(\tau)'P + PA(\tau) - PR(\tau)P, \tag{4.24a}$$

$$\dot{W} = PR(\tau)W - A(\tau)'W - PM(\tau) - N(\tau), \tag{4.24b}$$

with $P(T) = H_{xx}(T)$ *and* $W(T) = H_T^{xe}$. *Here,* I *is identity,*

$$A(\tau) = F - G(H_{uu})^{-1}H_{ux}, \tag{4.25a}$$

$$R(\tau) = G(H_{uu})^{-1}G', \tag{4.25b}$$

$$M(\tau) = E - G(H_{uu})^{-1}H_{ue}, \tag{4.25c}$$

$$Q(\tau) = H_{xx} - H_{xu}(H_{uu})^{-1}H_{ux}, \tag{4.25d}$$

$$N(\tau) = H_{xe} - H_{xu}(H_{uu})^{-1}H_{ue}, \tag{4.25e}$$

are all known given (4.23). Then, the gradient of the optimal trajectory at any time instance $0 \leq \tau \leq T$, *denoted as*

$$\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau)}{\partial \boldsymbol{\theta}} = \left(\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(\tau), \frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(\tau)\right) \tag{4.26}$$

is obtained by integrating the following equations up to τ*:*

$$\frac{d}{d\tau}\left(\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\right) = -(H_{uu}(\tau))^{\text{-}1}\Big(H_{ux}(\tau) + G(\tau)'W(\tau) + H_{ue}(\tau) + G(\tau)'P(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(\tau)\Big), \tag{4.27a}$$

$$\frac{d}{d\tau}\left(\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\right) = F(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(\tau) + G(\tau)\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(\tau) + E(\tau), \tag{4.27b}$$

with $\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(0) = \mathbf{0}$ *(because* $\boldsymbol{x}(0)$ *is given), where the matrices* $\{P(\tau)\}$ *and* $\{W(\tau)\}$ *are the solutions to the differential equations in (4.24a) and (4.24b), respectively.*

Proof. Consider the differential equations of the Pontryagin's Maximum Principle in (4.22), which we rewrite as below:

$$\frac{d}{d\tau}(\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}) = F(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + G(\tau)\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + E(\tau), \tag{4.28a}$$

$$-\frac{d}{d\tau}(\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}) = H_{xx}(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{xu}(\tau)\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + F(\tau)'\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{xe}(\tau), \tag{4.28b}$$

$$\mathbf{0} = H_{ux}(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{uu}(\tau)\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + G(\tau)'\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{ue}(\tau), \tag{4.28c}$$

$$\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(T) = H_{xx}(T)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{xe}(T). \tag{4.28d}$$

Consider that $H_{uu}(\tau)$ in (4.23c) is invertible for all $0 \leq \tau \leq T$, we can solve the $\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ from (4.28c):

$$\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = -H_{uu}^{-1}(\tau)\Big(H_{ux}(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + G(\tau)'\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + H_{ue}(\tau)\Big). \tag{4.29}$$

Substituting (4.29) into both (4.28a) and (4.28b) and combining the definition of matrices in (4.25), we have

$$\frac{d}{d\tau}\Big(\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big) = A(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} - R(\tau)\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + M(\tau), \tag{4.30a}$$

$$-\frac{d}{d\tau}\Big(\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big) = Q(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + A(\tau)'\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + N(\tau). \tag{4.30b}$$

Motivated by (4.28d), we assume

$$\frac{\partial \boldsymbol{\lambda}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = P(\tau)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + W(\tau), \tag{4.31}$$

with introduced $P(\tau) \in \mathbb{R}^{n\times n}$ and $W(\tau) \in \mathbb{R}^{n\times(s+r)}$ are two time-varying matrices for $0 \leq \tau \leq T$. Of course, the above (4.31) holds for $\tau = T$, if

$$P(\tau) = H_{xx}(T) \quad \text{and} \quad W(\tau) = H_{xe}(T). \tag{4.32}$$

Substituting (4.31) to (4.30b) and (4.30b), respectively, to eliminate $\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$, we obtain the following

$$\frac{d}{d\tau}\Big(\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big) = (A - RP)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + (-RW + M), \tag{4.33a}$$

$$-\dot{P}\frac{d}{d\tau}\Big(\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}\Big) = (Q + \dot{P} + A'P)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + (A'W + N + \dot{W}), \tag{4.33b}$$

where $\dot{P} = \frac{dP(\tau)}{d\tau}$, $\dot{W} = \frac{dW(\tau)}{d\tau}$, and we here have suppressed the dependence on time τ for all time-varying matrices. By multiplying $(-\dot{P})$ on both sides of (4.33a), and equaling the left sides of (4.33a) and (4.33b), we have

$$\begin{aligned}&(-PA + PRP)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + (PRW - PM)\\ =&(Q + \dot{P} + A'P)\frac{\partial \boldsymbol{x}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + (A'W + N + \dot{W}).\end{aligned} \tag{4.34}$$

The above equation holds if

$$-PA + PRP = Q + \dot{P} + A'P, \tag{4.35a}$$

$$PRW - PM = A'W + N + \dot{W}, \tag{4.35b}$$

which directly are (4.24). Substituting (4.31) into (4.29) yields (4.27a), and (4.27b) directly results from (4.28a). This completes the proof. □

Lemma 4.4.1 states that for the optimal control system (4.12), the gradient of its optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ (the trajectory satisfying Pontryagin's Maximum Principle) with respect to parameter $\boldsymbol{\theta}$ can be obtained in two steps: first, integrate (4.24) backward in time to obtain matrices $\{P(\tau)\}$ and $\{W(\tau)\}$ for $0 \leq \tau \leq T$; and second, obtain $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(\tau)$ by integrating (4.27). With the differential Pontryagin's maximum principle, Lemma 4.4.1 states an efficient way to obtain the gradient of the optimal trajectory with respect the unknown parameters in an optimal control system. By Lemma 4.4.1, one can obtain the derivative of any trajectory point $\boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau)$, for any $0 \leq \tau \leq T$, along the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$, with respect to the parameter $\boldsymbol{\theta}$, $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(\tau)$.

Based on Lemma 4.4.1, we summarize the overall algorithm to solve the optimization problem (4.16) in Algorithm 7.

4.5 Numerical Examples

We demonstrate the proposed approach using two systems: (i) an inverted pendulum, and (ii) 6-DoF maneuvering quadrotor. We compare the proposed method with related work.

Algorithm 7: Learning from Sparse Demonstrations

Input: Sparse demonstrations $\mathcal{D}$ in (4.5) and learning rate $\{\eta_k\}$.
Initialization: initial parameter guess $\boldsymbol{\theta}_0$,
for $k = 0, 1, 2, \cdots$ **do**
  Obtain the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by solving the optimal control problem in (4.12) with currnet parameter $\boldsymbol{\theta}_k$;
  Obtain the costate trajectory $\{\boldsymbol{\lambda}_{\boldsymbol{\theta}_k}(\tau)\}$ using by integrating (4.21b) given (4.21d);
  Compute $\frac{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)}{\partial \boldsymbol{\theta}}\big|_{\boldsymbol{\theta}_k}$ using Lemma 4.4.1 for $i = 1, 2, \cdots N$;
  Comopute $\frac{\partial l}{\partial \boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)}\big|_{\boldsymbol{\xi}_{\boldsymbol{\theta}_k}(\tau_i)}$ from (4.15);
  Compute $\frac{dL}{d\boldsymbol{\theta}}|_{\boldsymbol{\theta}_k}$ using the chain rule (4.18);
  Update $\boldsymbol{\theta}_{k+1} \leftarrow \text{Proj}_{\boldsymbol{\Theta}}\left(\boldsymbol{\theta}_k - \eta_k \frac{dL}{d\boldsymbol{\theta}}\big|_{\boldsymbol{\theta}_k}\right)$;
end

4.5.1 Inverted Pendulum

The dynamics of an inverted pendulum is given in Appendix A.1. We define the state and control variables of the pendulum system as $\boldsymbol{x} = [\alpha, \dot{\alpha}]'$ and $\boldsymbol{u} = u$, respectively, and set the initial state $\boldsymbol{x}(0) = [0, 0]'$. For the inverted pendulum control, we set the parameterized cost function in (4.2) as

$$\begin{aligned} c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) &= p_1(\alpha - \pi)^2 + p_2\dot{\alpha}^2 + 0.1u^2, \\ h(\boldsymbol{x}, \boldsymbol{p}) &= p_1(\alpha - \pi)^2 + p_2\dot{\alpha}^2, \end{aligned} \tag{4.36}$$

with the parameter vector $\boldsymbol{p} = [p_1, p_2]'$ to be determined. For the parametric time-warping function (4.9), we simply use a linear function:

$$t = w_{\boldsymbol{\beta}}(\tau) = \beta\tau, \tag{4.37}$$

with $\beta \in \Omega_{\boldsymbol{\beta}} = \{\beta : \beta > 0\}$ (we will discuss the use of more complex time-warping functions later). The overall parameter vector to be determined is $\boldsymbol{\theta} = [\boldsymbol{p}', \beta]' \in \mathbb{R}^3$.

The output function (4.4) is set as $\alpha = \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$, which means that the expert only provides the position information, not including the velocity information. For the trajectory loss function in (4.15), we use the l_2 norm to quantify the distance measure:

$$L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D}) = \sum_{i=1}^{N} \|\boldsymbol{y}^*(\tau_i) - \boldsymbol{g}\Big(\boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)\Big)\|^2. \tag{4.38}$$

Known Ground Truth

First, we generate sparse demonstrations $\mathcal{D}$ to test the proposed method when the true objective function and time-warping function are both known. Specifically, we set the true parameter $\boldsymbol{\theta}^* = [1, 1, 2]'$, based on which we generate the trajectory by solving the optimal control problem (4.12). Then, we pick some points as the sparse demonstrations $\mathcal{D}$, listed in Table 4.1. We want to see if the proposed method can correctly learn $\boldsymbol{\theta}^*$ from these sparse points. Given the sparse waypoints in Table 4.1, we apply Algorithm 7 to learn the parameter $\boldsymbol{\theta}$ by solving (4.16). In Algorithm 7, we set the learning rate $\eta = 10^{-2}$, and initialize the parameter $\boldsymbol{\theta}$ randomly.

Table 4.1. Sparse demonstrations $\mathcal{D}$ for inverted pendulum.

Demonstration time instance τ_i	waypoints $\boldsymbol{y}^*(\tau_i)$
$\tau_1 = 0.1$s	$\alpha^*(\tau_1) = 0.371$
$\tau_2 = 0.3$s	$\alpha^*(\tau_2) = 1.372$
$\tau_3 = 0.6$s	$\alpha^*(\tau_3) = 2.286$
$\tau_4 = 0.7$s	$\alpha^*(\tau_4) = 2.475$
$\tau_5 = 1.0$s	$\alpha^*(\tau_5) = 2.785$
Time horizon $T = 1$s	

We plot the loss value $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$ in (4.38) versus the number of iterations in Fig. 4.2. The result shows that as the iteration number increases, the loss diminishes fast and finally converges to zero. This indicates that the trajectory gradually gets close to the sparse demonstrations and finally passes through them. This convergence is also illustrated by the right panel of Fig. 4.2, where we plot the pendulum's (time-warped) trajectory in each

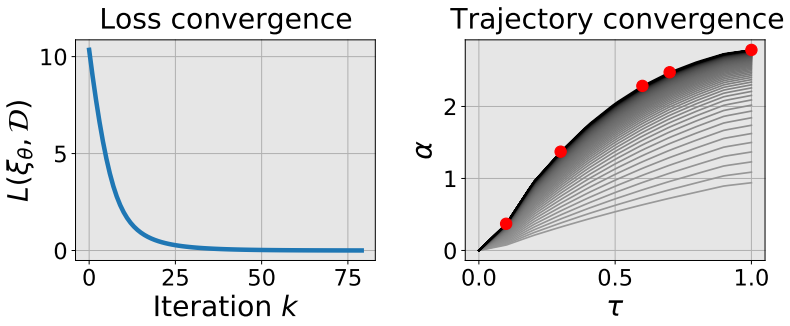


Figure 4.2. Learning from sparse demonstrations for inverted pendulum using data in Table 4.1. Left: the loss value (4.38) versus the number of iterations. Right: the convergence of the pendulum's (time-warped) trajectory as iteration increases, where the color from light to dark gray corresponds to increasing iteration number, and the red dots are waypoints in Table 4.1.

iteration, where the color going from light to dark gray corresponds to increasing iteration number, and the red dots indicate the sparse demonstrations. As shown by the results, the initial trajectory (lightest gray) is far away from the sparse demonstrations, and as $\boldsymbol{\theta}$ updates, the trajectory (with increasingly dark colors) approaches and finally passes through the waypoints (i.e., the converged loss is zero). To illustrate whether the parameters converge to the ground truth $\boldsymbol{\theta}^* = [1, 1, 2]'$, we define the following parameter error: $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$, and plot the parameter error versus the number of iterations in Fig. 4.3, from which we note that as the number of iterations increases, $e_{\boldsymbol{\theta}}$ converges to zero, indicating that the true parameter $\boldsymbol{\theta}^*$ of the objective and time-warping functions is successfully learned.

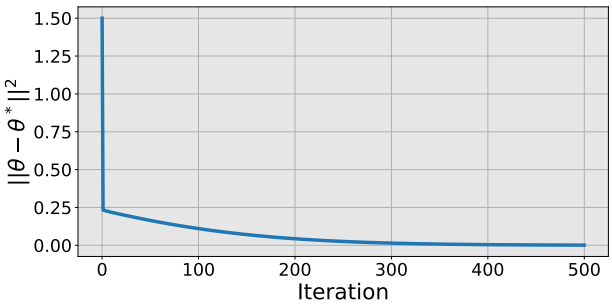


Figure 4.3. Parameter error $\|\boldsymbol{\theta}_k - \boldsymbol{\theta}^*\|^2$ versus iteration number.

Non-realizable Case

In this case, we use random sparse demonstrations, where the waypoints here are sampled from a uniform distribution with the centers being the ones in Table 4.1. The randomness of the given sparse demonstrations means that an exact objective function (whose optimal trajectory exactly passes through the sparse demonstrations) may not exist within the given parameterized function set in (4.36) because of limited expressive power. The random sparse demonstrations are listed in Table 4.2, and the other settings are the same as the previous case. The learning results are shown in Fig. 4.4. The results show that as the number of iterations increases, the loss value (4.38) is decreasing and converging to a value of 0.391 but not zero. This is because the waypoints are randomly given, thus there does not exist $\boldsymbol{\theta}^*$ such that the corresponding system trajectory *exactly* passes through these given waypoints. It shows that the proposed method can always find the 'best' objective function and the 'best' time-warping function within the parametric function sets, which finally leads the reproduced trajectory to be closest to the waypoints in a sense of having the minimal distance loss (4.7), as shown in the right panel of Fig. 4.4.

Table 4.2. Sparse demonstrations $\mathcal{D}$ for pendulum system.

Demonstration time instance τ_i	waypoints $\boldsymbol{y}^*(\tau_i)$
$\tau_1 = 0.1$s	$\alpha^*(\tau_1) = 0.5$
$\tau_2 = 0.3$s	$\alpha^*(\tau_2) = 1.8$
$\tau_3 = 0.6$s	$\alpha^*(\tau_3) = 2.0$
$\tau_4 = 0.7$s	$\alpha^*(\tau_4) = 2.9$
$\tau_5 = 0.9$s	$\alpha^*(\tau_5) = 3.1$
Time horizon $T = 1$s	

Different Parametric Time-Warping Functions

In this case, we test the performance of the method using different parametric time-warping functions. The sparse demonstrations $\mathcal{D}$ are in Table 4.3, where the demonstration time labels τ_k are infeasible for the pendulum actuation. The other experimental settings are

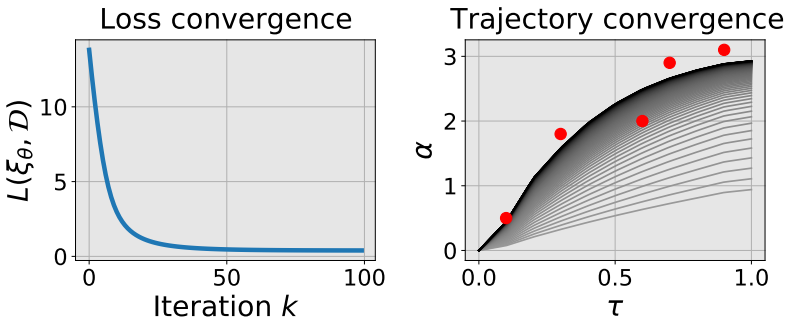


Figure 4.4. Learning from sparse demonstrations for inverted pendulum from data in Table 4.2. Left: the loss value (4.38) versus the number of iterations. Right: the convergence of the pendulum's (time-warped) trajectory as the number of iterations increases, where the color from light to gray dark corresponds to increasing iteration number, and the red dots are waypoints in Table 4.2.

the same as the previous cases, except that we use the parametric polynomial time-warping function (4.9) with different degrees s. We summarize in Table 4.4 the learned time-warping function and the obtained minimal loss value of (4.38), i.e., $\min L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$.

Table 4.3. Sparse demonstrations $\mathcal{D}$ for pendulum system.

Demonstration time instance τ_i	waypoints $\boldsymbol{y}^*(\tau_i)$
$\tau_1 = 0.02$s	$\alpha^*(\tau_1) = 0.5$
$\tau_2 = 0.06$s	$\alpha^*(\tau_2) = 1.8$
$\tau_3 = 0.12$s	$\alpha^*(\tau_3) = 2.0$
$\tau_4 = 0.14$s	$\alpha^*(\tau_4) = 2.9$
$\tau_5 = 0.18$s	$\alpha^*(\tau_5) = 3.1$
Time horizon $T = 0.2$s	

As shown in Table 4.4, more complex time-warping functions lead to a lower minimal loss value of $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$. This is understandable because using a higher-degree polynomial will introduce additional degrees of freedom, which contribute to further decreasing the loss $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$ in terms of generating a 'more-deformed' time axis. Also from a system perspective, if we look at the entire parameterized optimal control system (4.12), use of a

Table 4.4. Different polynomial time-warping functions

Learned time-warping function $t = w(\tau)$	$\min L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$
t=4.656τ	0.423
t=4.826$\tau + 0.167\tau^2$	0.413
t=5.094τ+0.171$\tau^2 + 0.016\tau^3$	0.400
t=5.200τ+0.177τ^2+0.018τ^3+0.004τ^4	0.395

higher-degree polynomial time-warping function will make the parameterized system more expressive, achieving a lower loss on the same training data.

From Table 4.4, we further observe that the first-order terms in all learned time-warping polynomials are approximately the same, and the higher-order terms are relatively small compared to the first-order term and they do not significantly contribute to lowering the final training loss. This indicates that the first-order term dominates the time scale difference between the demonstration and robot's execution, because T here is small and the higher-order terms thus are not significant compared to the first-order term. In the following experiments, we therefore only use the first-order polynomial time-warping functions.

Neural Objective Functions

Instead of using parameterization (4.36), we here represent the objective function using a neural network and aim to learn a neural objective function. We test this still using the inverted pendulum system. Specifically, the parameterized objective function is represented as

$$\begin{aligned} c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) &= V(\boldsymbol{x}, \boldsymbol{p}) + 0.0001\|\boldsymbol{u}\|^2, \\ h(\boldsymbol{x}, \boldsymbol{p}) &= V(\boldsymbol{x}, \boldsymbol{p}), \end{aligned} \tag{4.39}$$

where $V(\boldsymbol{x}, \boldsymbol{p})$ is a *2-2-1* fully-connected neural network with tanh activation functions [156] (i.e., 2-neuron input layer, 2-neuron hidden layer, and 1-neuron output layer), and $\boldsymbol{p} \in \mathbb{R}^9$ is the parameter vector of the neural network, that is, the weight matrices and bias vectors. The time-warping polynomial is first-order as in (4.37) and the loss function is (4.38). We use the sparse demonstration data in Table 4.3, and the learning rate is set as $\eta = 10^{-2}$. We plot the

learning results in Fig. 4.5, which shows that the proposed approach can successfully learn a neural objective function from sparse demonstrations, such that the pendulum's reproduced trajectory is close to the given waypoint in Euclidean distance.

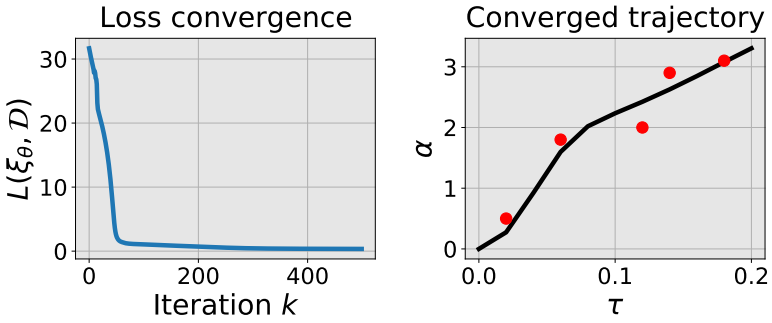


Figure 4.5. Learning from sparse waypoints with the objective function represented by a neural network. Left: the loss value (4.38) versus the number of iterations, and the loss finally converges to 0.346. Right: the learned time-warped trajectory, where the red dots are waypoints in Table 4.3.

In the left panel of Fig. 4.5, the converged loss is 0.346, which is lower than the loss of 0.423 in Table 4.4 for the weighted distance parameterization (4.36). This difference can be also seen by comparing the right panel of Fig. 4.5 with the one in Fig. 4.4. The lower loss here is because neural network representation is more expressive than weighted distance parameterization. The results in Fig. 4.5 demonstrate the capability of the proposed method to learn complex parametric objective functions, and it shows the utility of the method when the knowledge-based parametric objective function is not readily available.

However, despite the convenience of using universal neural network objective functions, how to choose appropriate structure and hyper-parameters for a neural network (such as the number of layers/neurons and the type of activation functions) still needs to be specified. Our empirical experience also finds the other drawbacks of neural objective functions, including a lack of physical interpretability for the learned results, more iterations needed to reach convergence as empirically shown in left panel of Fig. 4.5, and a tendency of getting trapped in locally optimal solutions. In Section 4.7, we will provide a further analysis for the choice of parametric objective functions.

4.5.2 Comparison with other Methods

Comparison with Learning From KeyFrames [155]

We first compare the proposed method with the method of learning from keyframe demonstrations developed in [155]. As discussed in the related work, this is a policy-learning based method: a Gaussian mixture model (GMM) is first learned from keyframe demonstrations, based on which a trajectory is then reproduced using Gaussian mixture regression (GMR). In this comparison experiment, we use the inverted pendulum system with the same setting as in Section 4.5.1. Here, we provide 20 waypoints (with the time instances evenly populated over $[0, 1]$; we find that a smaller number of waypoints leads to failure of the GMM method). During trajectory reproduction, we set a new time duration $T = 2$ (note that the training data uses $T = 1$) to test the generalization performance of each method. Comparison results are plotted in Fig. 4.6, where we also plot the ground-truth for reference.

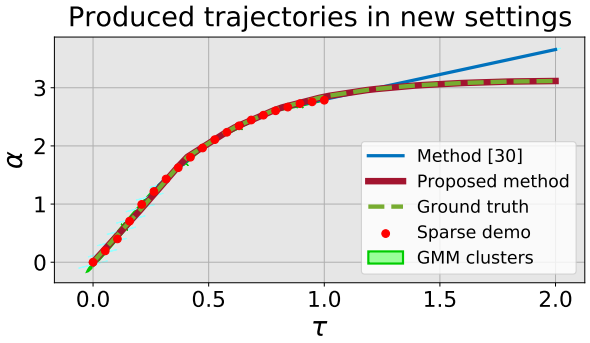


Figure 4.6. Reproduced trajectories with a new time duration $T = 2$ (note that the demonstration data is with the duration $T = 1$).

From Fig. 4.6, we observe that under unseen information (here with a longer time horizon), our method produces a trajectory much closer to the ground truth than [155]. This indicates better generalization of the proposed method to unseen settings (or long horizon tasks). In fact, better generalization is generally one of the advantages of objective function learning over policy learning, as discussed in [148].

Comparison with Numerical Gradient Descent

Here, we compare the proposed method with direct gradient descent, where the gradient is *estimated numerically*. Specifically, in each update we use the numerical differentiation to approximate the gradient $\frac{dL}{d\boldsymbol{\theta}}$. The experiment uses the pendulum system with the same settings as Section 4.5.1. Here we have tried two cases: the first case uses the sparse demonstration data in Table 4.1, and the second case uses the sparse demonstration data in Table 4.2. The comparison results are shown in Fig. 4.7.

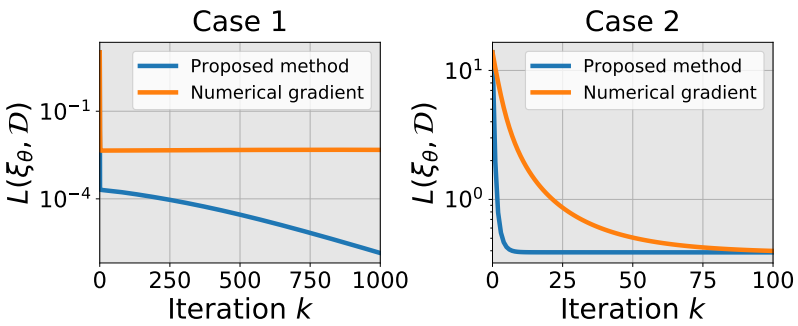


Figure 4.7. Comparison between the proposed method and numerical gradient descent. Left: using the sparse demonstrations in Table 4.1; and right: using the sparse data in Table 4.2. Both methods use the same learning rate $\eta = 10^{-2}$.

From Fig. 4.7, we can observe that the proposed method has an obvious advantage in terms of lower training loss and faster convergence speed. The numerical gradient descent is effective for this case but has a lower accuracy due to the error induced during gradient approximation. Because of this approximation error, the loss does not descend along the 'steepest' direction, thus leading to a slower convergence. Here, the optimization variable $\boldsymbol{\theta} \in \mathbb{R}^3$ is low-dimensional, the numerical gradient is thus relatively easier to compute, and the numerical gradient descent works. For high dimensional tasks, as we will show below, we found that the numerical gradient descent is prone to fail due to inaccuracy of gradient estimation.

4.5.3 Experiment on 6-DoF Maneuvering Quadrotor

We here show the effectiveness of the proposed method on a more complex 6-DoF maneuvering quadrotor. The equation of motion of a quadrotor flying in SE(3) (full position and attitude) space is given in Appendix A.3. We define the state variable

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{r}_I & \boldsymbol{v}_I & \boldsymbol{q}_{B/I} & \boldsymbol{\omega}_B \end{bmatrix} \in \mathbb{R}^{13}. \tag{4.40}$$

and define the control variable

$$\boldsymbol{u} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 \end{bmatrix}' \in \mathbb{R}^4. \tag{4.41}$$

To achieve SE(3) maneuvering control, we need to carefully design the attitude error. As in [157], we define the attitude error between the quadrotor's current attitude $\boldsymbol{q}$ and goal attitude $\boldsymbol{q}^{\mathrm{g}}$ as

$$e(\boldsymbol{q}, \boldsymbol{q}_{\mathrm{g}}) = \frac{1}{2}\mathrm{trace}(I - R'(\boldsymbol{q}^{\mathrm{g}})R(\boldsymbol{q})), \tag{4.42}$$

where $R(\boldsymbol{q}) \in \mathbb{R}^{3\times 3}$ is the direction cosine matrix corresponding to the quaternion $\boldsymbol{q}$ (see [158] for more details).

The parameterized cost function in (4.2) is set as

$$c(\boldsymbol{p}) = \|\boldsymbol{p}_r'(\boldsymbol{r}_I - \boldsymbol{r}_I^{\mathrm{g}})\|^2 + \|\boldsymbol{p}_v'(\boldsymbol{v}_I - \boldsymbol{v}_I^{\mathrm{g}})\|^2 + p_q e(\boldsymbol{q}_{B/I}, \boldsymbol{q}_{B/I}^{\mathrm{g}}) + \|\boldsymbol{p}_\omega'(\boldsymbol{\omega}_B - \boldsymbol{\omega}_B^{\mathrm{g}})\|^2 + 0.1\|\boldsymbol{u}\|^2, \tag{4.43a}$$

$$h(\boldsymbol{p}) = \|\boldsymbol{p}_r'(\boldsymbol{r}_I - \boldsymbol{r}_I^{\mathrm{g}})\|^2 + \|\boldsymbol{p}_v'(\boldsymbol{v}_I - \boldsymbol{v}_I^{\mathrm{g}})\|^2 + p_q \cdot e(\boldsymbol{q}_{B/I}, \boldsymbol{q}_{B/I}^{\mathrm{g}}) + \|\boldsymbol{p}_\omega'(\boldsymbol{\omega}_B - \boldsymbol{\omega}_B^{\mathrm{g}})\|^2. \tag{4.43b}$$

Here, $\boldsymbol{r}_I^{\mathrm{g}} = \boldsymbol{0}$, $\boldsymbol{v}_I^{\mathrm{g}} = \boldsymbol{0}$, $\boldsymbol{q}_{B/I}^{\mathrm{g}} = [1, 0, 0, 0]'$, and $\boldsymbol{w}_B^{\mathrm{g}} = \boldsymbol{0}$ are the goal position, velocity, orientation, and angular velocity, respectively; the objective function parameter vector here is

$$\boldsymbol{p} = [\boldsymbol{p}_r,\ \boldsymbol{p}_v,\ p_q,\ \boldsymbol{p}_\omega]' \in \mathbb{R}^{10}. \tag{4.44}$$

For the parametric time-warping function, we use the first-degree polynomial as in (4.37). The total parameter vector to be determined is

$$\boldsymbol{\theta} = [\boldsymbol{p}, \beta]' \in \mathbb{R}^{11}. \tag{4.45}$$

We set the output function in (4.4) as

$$\boldsymbol{y} = [\boldsymbol{r}_I, \boldsymbol{q}_{B/I}] = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}), \tag{4.46}$$

which means that the expert can only provide the position and attitude demonstrations for quadrotor maneuvering (not including velocity information).

Table 4.5. Sparse demonstrations $\mathcal{D}$ for quadrotor maneuvering.

time instance τ_i	waypoints $\boldsymbol{y}^*(\tau_i)$	
$\tau_1 = 0.13$s	$\boldsymbol{r}_I^*(\tau_1) = [$-8.20, -2.47, 8.42$]$	$\boldsymbol{q}_{B/I}^*(\tau_1) = [0.97, $-0.16, -0.12, 0.04$]$
$\tau_2 = 0.40$s	$\boldsymbol{r}_I^*(\tau_2) = [$-7.35, -4.90, 5.10$]$	$\boldsymbol{q}_{B/I}^*(\tau_2) = [0.91, $-0.38, 0.14, -0.12$]$
$\tau_3 = 0.80$s	$\boldsymbol{r}^*(\tau_3) = [$-3.85, -2.85, 2.35$]$	$\boldsymbol{q}_{B/I}^*(\tau_3) = [0.99, 0.05, $-0.09, -0.10$]$
$\tau_4 = 1.33$s	$\boldsymbol{r}_I^*(\tau_4) = [$-1.09-0.71, 0.82$]$	$\boldsymbol{q}_{B/I}^*(\tau_4) = [0.99, 0.07, $-0.07, -0.09$]$
$\tau_5 = 1.73$s	$\boldsymbol{r}_I^*(\tau_5) = [$-0.48, -0.32, 0.37$]$	$\boldsymbol{q}_{B/I}^*(\tau_5) = [0.99, 0.02, $-0.03, -0.08$]$
Time horizon $T = 2$s		

The sparse demonstrations are in Table 4.5. The loss function $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$ is defined using Euclidean distance as in (4.38). In Algorithm 7, we set the learning rate $n = 10^{-2}$. We plot the learning results in Fig. 4.8. The results show that, as the parameter $\boldsymbol{\theta}$ is updated at each iteration, the loss value $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$ diminishes to zero quickly, meaning that the quadrotor's reproduced trajectory gets closest to the sparse demonstrations in Table 4.5. The right panel of Fig. 4.8 shows the final reproduced trajectory, which exactly passes through the given sparse demonstrations. This indicates the capability of the method in handling more complex systems.

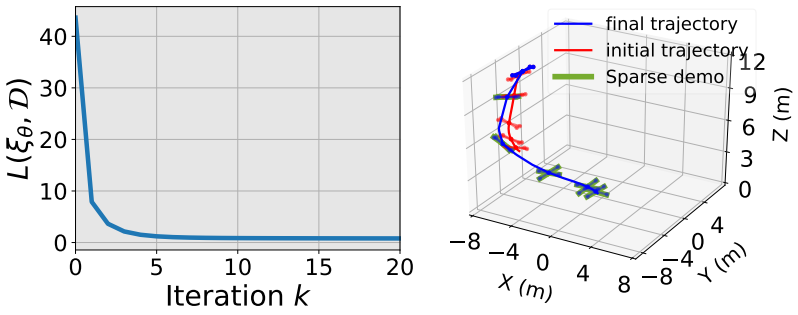


Figure 4.8. Learning from sparse demonstrations for 6-DoF quadrotor maneuvering. Left: the loss function value $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$ versus the number of iterations. Right: the quadrotor trajectory before learning (red) and the quadrotor trajectory after learning (blue), and green objects are the sparse demonstrations in Table 4.5.

4.6 Application: Learning for Obstacle Avoidance

In this section, we apply the proposed method to learning robot motion control in an environment with obstacles. Here, a human provides few waypoints in the vicinity of obstacles in an environment, and the robot learns a control objective function from those waypoints such that its resulting motion can get around the obstacles. We experiment on two systems: a 6-DoF maneuvering quadrotor and a two-link robot arm.

4.6.1 6-DoF Maneuvering Quadrotor

We still use the 6-DoF quadrotor system in Section 4.5.3. For the parameterized control objective function (4.2), instead of using the weighted distance to the goal state, we here use a general second-order polynomial parameterization as follows:

$$c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) = p_1 r_x^2 + p_2 r_x + p_3 r_y^2 + p_4 r_y + p_4 r_z^2 + p_5 r_z + 0.1\|\boldsymbol{u}\|^2, \tag{4.47a}$$

$$h(\boldsymbol{x}) = \|\boldsymbol{r}_I - \boldsymbol{r}_I^{\mathrm{g}}\|^2 + 10\|\boldsymbol{v}_I\|^2 + 100e(\boldsymbol{q}_{B/I}, \boldsymbol{q}_{B/I}^{\mathrm{g}}) + 10\|\boldsymbol{w}_B\|^2, \tag{4.47b}$$

where $\boldsymbol{r}_I = [r_x, r_y, r_z]'$ is the position of the quadrotor expressed in the world coordinate frame, and we have fixed the final cost $h(\boldsymbol{x})$ (i.e., no tunable parameters) since we always want the quadrotor to finally land on a target position given by $\boldsymbol{r}_I^{\mathrm{g}}$. Here the tunable objective

function parameter is $\boldsymbol{p} = [p_1, p_2, p_3, p_4, p_5]'$ in the running cost $c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p})$ as it determines how the quadrotor reaches the target (i.e., the specific path of the quadrotor).

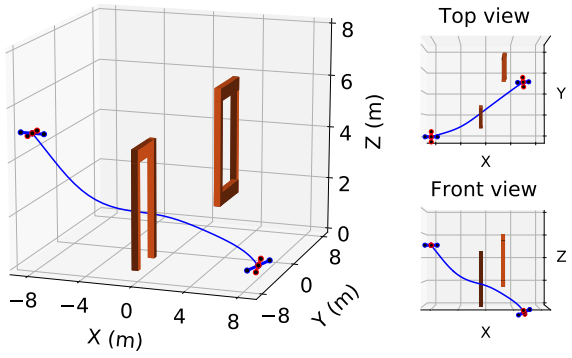


Figure 4.9. Quadrotor maneuvers in an environment with obstacles. The quadrotor's aim is to go through the two gates (from left to right) and finally land on the target position in the upper right corner. The plotted trajectory is a simulation with a random initial control objective function, which fails to achieve the goal (the quadrotor may crash into the first gate, as seen from the top view).

As shown in Fig. 4.9, we aim for the quadrotor to fly from the left position $\boldsymbol{r}_I(0) = [-8, -8, 5]'$, go through two gates (as depicted in Fig. 4.10), and finally land on the target position on the right $\boldsymbol{r}_I^{\mathrm{g}} = [8, 8, 0]'$. In Fig. 4.9, we draw the trajectory of the quadrotor for a random initial objective function parameter $\boldsymbol{p}$. It can be seen that here the quadrotor, although finally landing on the target position, does not meet the requirement of going through the two gates. We also note that the quadrotor may crash into the first gate (as seen from the top view). In the following, we will train the quadrotor by providing few sparse waypoints.

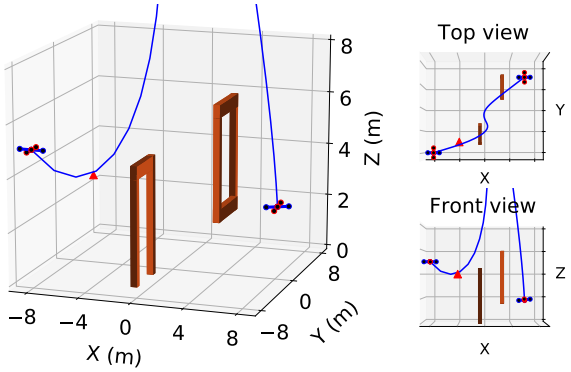
We provide the waypoints listed in Table 4.6. Note that we here only provide the position information for the quadrotor (The output function in (4.4) is now is $\boldsymbol{r}_I = \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$). Also note that we do not know whether these waypoints correspond to an exact objective function within the parameterized function set; we also do not know if the given time label for each waypoint and time horizon are achievable, i.e. if there exist an exact objective function and time-warping function such that the resulting trajectory exactly passes through the waypoints exactly at the given time instances.

Table 4.6. Sparse waypoints $\mathcal{D}$ for quadrotor maneuvering.

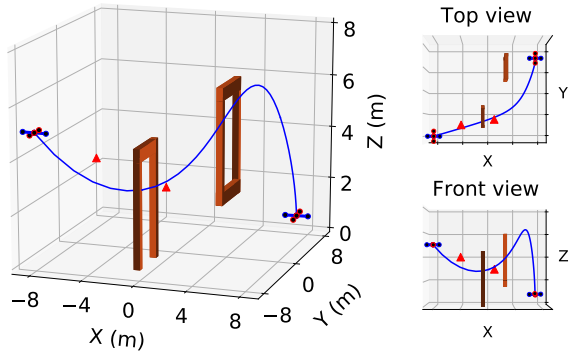
Demonstration time instance τ_i	waypoints $\boldsymbol{y}^*(\tau_i)$
$\tau_1 = 0.07$s	$\boldsymbol{r}_I(\tau_1) = [-4, -6, 4]$
$\tau_2 = 0.2$s	$\boldsymbol{r}_I(\tau_2) = [1, -5, 3]$
$\tau_3 = 0.4$s	$\boldsymbol{r}_I(\tau_3) = [1, -1, 4]$
$\tau_4 = 0.47$s	$\boldsymbol{r}_I(\tau_4) = [-1, 1, 4]$
$\tau_5 = 0.67$s	$\boldsymbol{r}_I(\tau_5) = [2, 3, 4]$
Time horizon $T = 1$s	

We divide the experiment into four cases, and for each case we use a different number of waypoints from Table 4.6 to learn an objective function and a time-warping function. The parametric time-warping function is first-order polynomial as in (4.37), and the loss function $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$ is set as (4.38). The learning rate is set as $\eta = 10^{-2}$.

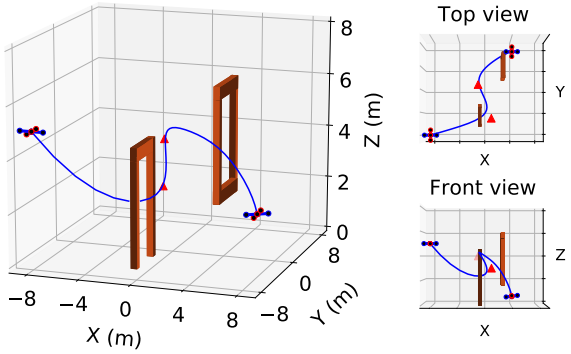
We plot the results in Fig. 4.10, where the quadrotor's trajectory reproduced by the learned objective function in different cases is shown. Specifically, in Fig. 4.10a, we only use one waypoint $\boldsymbol{r}_I^*(\tau_1)$ in Table 4.6 to learn the objective function (and time-warping function). The results in Fig. 4.10a illustrate that the learned objective function enables the quadrotor to reproduce a trajectory passing through the given waypoint and landing on the target position, but clearly the learned objective function fails to meet the requirement of going through the two gates. In Fig. 4.10b, we learn the objective function using two waypoints $\boldsymbol{r}_I^*(\tau_1)$ and $\boldsymbol{r}_I^*(\tau_2)$ in Table 4.6, where $\boldsymbol{r}_I^*(\tau_2)$ is placed because we want to guide the quadrotor to go through the first (left) gate. The results in Fig. 4.10b show that the quadrotor successfully learns an objective function to go through the first gate, and then land on the target position, but it fails to go through the other gate. In Fig. 4.10c, we only place two waypoints $\boldsymbol{r}_I^*(\tau_2)$ and $\boldsymbol{r}_I^*(\tau_3)$ between the two gates, where the waypoint $\boldsymbol{r}_I^*(\tau_2)$ accounts for the quadrotor to go through the first gate while the other waypoint $\boldsymbol{r}_I^*(\tau_3)$ is used to account for the navigation between two gates. The corresponding results show that the learned objective function successfully enables the quadrotor to go through the first gate, pass through the second waypoint, and finally land on the target position. However, as shown in top view in Fig. 4.10c, the quadrotor may crash into the frame of the second



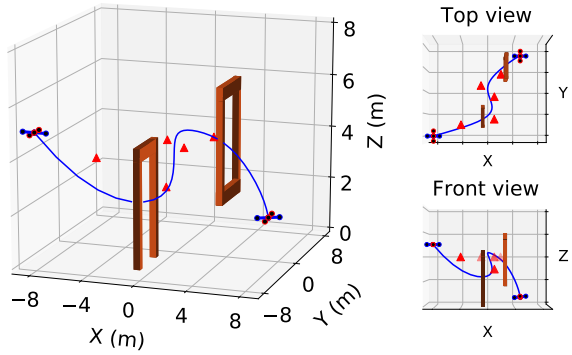
(a) Learning from only one waypoint $\boldsymbol{r}(\tau_1)$



(b) Learning from two waypoints $\boldsymbol{r}(\tau_1)$ and $\boldsymbol{r}(\tau_2)$



(c) Learning from two waypoints $\boldsymbol{r}(\tau_2)$ and $\boldsymbol{r}(\tau_3)$



(d) Learning from five waypoints from $\boldsymbol{r}(\tau_1)$ to $\boldsymbol{r}(\tau_5)$

Figure 4.10. 6-DoF quadrotor learns to maneuver control in an environment with obstacles: the quadrotor aims to start from the left position $(-8, -8, 5)$, then go though two gates, and finally land on a target position $(8, 8, 0)$ on the right. In different sub-figures, we use different number of waypoints from Table 4.6. The waypoints are labeled as red triangles. The motion trajectory reproduced by the learned objective function is shown in blue curve. Please find the video demo at https://wanxinjin.github.io/posts/lfsd.

gate. Compared to Fig. 4.10c, in Fig. 4.10d we provide two additional waypoints $\boldsymbol{r}_I^*(\tau_4)$ and $\boldsymbol{r}_I^*(\tau_5)$ in order to correctly guide the quadrotor to go through the second gate, and also one additional waypoint $\boldsymbol{r}_I^*(\tau_1)$ to the first gate. The results in Fig. 4.10d show that with these five waypoints, the quadrotor learns an objective function that successfully leads it to go through the two gates and finally land on the target position. In Fig. 4.11, we also plot the loss versus the number of iterations for each of the experiment cases.

The above experimental results demonstrate the effectiveness of the proposed method to learn objective functions from sparse demonstrations. It illustrates that the proposed method

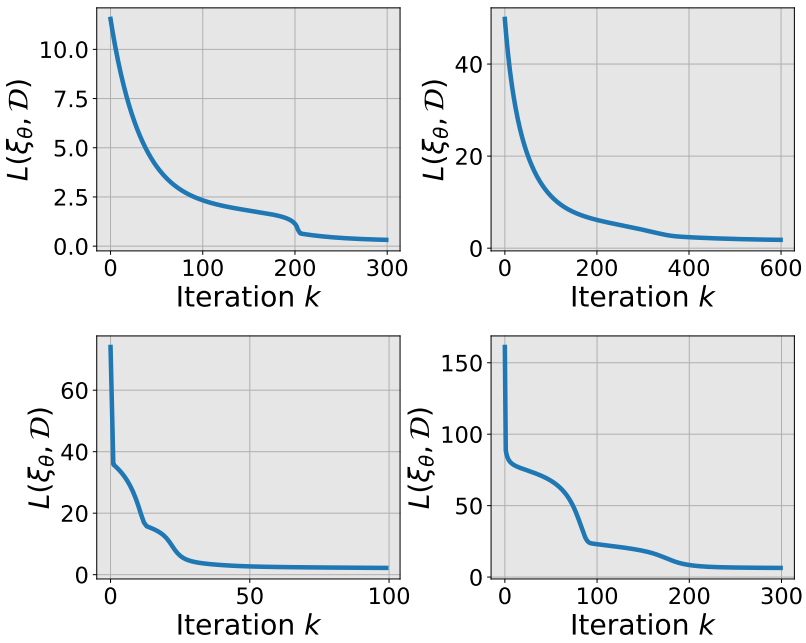


Figure 4.11. The loss versus number of iterations. The top-left panel is for experiment case (a) in Fig. 4.10, the top-right is for (b), the bottom-left is for (c), the bottom-right is for (d).

significantly simplifies the process of robot programming for motion planning/control tasks in environments with obstacles.

4.6.2 Two-link Robot Arm

In this part, we apply the proposed method to control a two-link robot arm in an environment with obstacles. The dynamics of the robot arm system are given in Appendix A.2. The state and control variables for the robot arm control system are $\boldsymbol{x} = [\boldsymbol{q}, \dot{\boldsymbol{q}}]'$ and $\boldsymbol{u} = \boldsymbol{\tau}$, respectively. Here all the parameters in the dynamics are set as units. We consider the cost function in (4.2) specifically as

$$c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) = p_1 q_1^2 + p_2 q_1 + p_3 q_2^2 + p_4 q_2 + 0.5\|\boldsymbol{u}\|^2, \tag{4.48a}$$

$$h(\boldsymbol{x}) = 10\|\boldsymbol{q} - \boldsymbol{q}_{\mathrm{g}}\|^2 + 100\|\dot{\boldsymbol{q}}\|^2, \tag{4.48b}$$

where the running cost $c(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p})$ is of a polynomial type with the tunable parameter $\boldsymbol{p} = [p_1, p_2, p_3, p_4]'$, and we also fix the final cost $h(\boldsymbol{x})$ because we aim the arm to finally reach the goal configuration given by $\boldsymbol{q}_{\mathrm{g}}$. Here $\boldsymbol{q}_{\mathrm{g}} = [\frac{\pi}{2}, 0]'$.

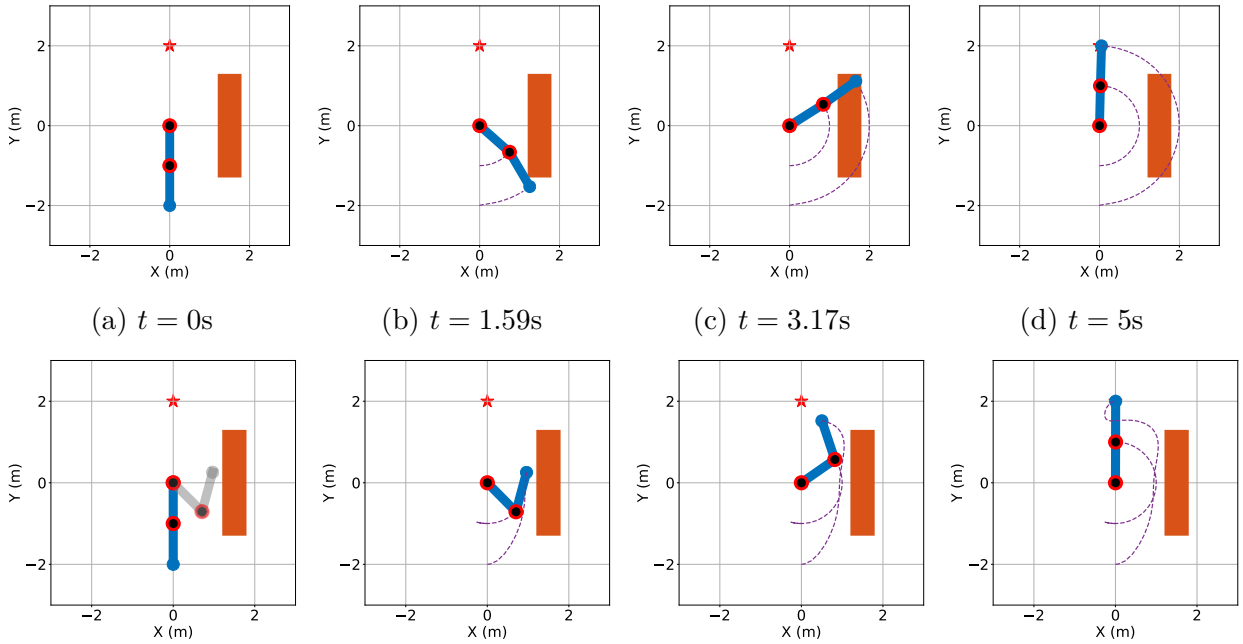
As shown in Fig. 4.12a, the robot arm is initially in state $\boldsymbol{x}(0) = [-\pi/2, 0, 0, 0, 0]'$ and we want the robot arm to reach and stop at the goal configuration $\boldsymbol{q}_{\mathrm{g}}$ while avoiding collision with an obstacle depicted by an orange block on its right side. Initially, we set the robot arm with an arbitrary initial running cost $c_{\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{u})$ and the resulting robot motion at different time instances is shown in Fig. 4.12a to 4.12d, respectively. Obviously, the robot arm has crashed into the obstacle during its motion (as seen from Fig. 4.12c).

Table 4.7. Sparse waypoints $\mathcal{D}$ for robot arm reaching.

Demonstration time instance τ_i	waypoints $\boldsymbol{y}^*(\tau_i)$
$\tau_1 = 0.3$s	$\boldsymbol{q}^*(\tau_1) = [-\frac{\pi}{4}, \frac{2\pi}{3}]$
Time horizon $T = 1$s	

Next, we give only one waypoint to the robot arm, which is in Table 4.7 (The output function (4.4) is $\boldsymbol{q} = \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$). Here the waypoint is away from the obstacle, as shown in gray in the second row of Fig. 4.12, since if the robot arm successfully follows the waypoint it could avoid crashing into the obstacle. Note that we do not know whether the given waypoint and the associated time are realizable or not (i.e. if there exist an exact objective function and time-warping function such that the resulting trajectory exactly pass through the waypoints at the given time instance). We apply the proposed method to learn both an objective function and time-warping function within the parameterized function set (4.48). The parametric time-warping function is the first-order polynomial given in (4.37), and the loss function $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$ is set as (4.38). The learning rate is set as $\eta = 10^{-2}$.

The learning results are in the second-row panels in Fig. 4.12, where we also show the demonstrated waypoint $\boldsymbol{q}^*(\tau_1)$ in Fig. 4.12e with gray color. The results show that with the learned objective function (and the learned time-warping function), the robot arm can successfully avoid the obstacle in its reaching motion. This demonstrates the effectiveness of the proposed method: even with only a single demonstration waypoint, the robot can



(a) $t = 0$s (b) $t = 1.59$s (c) $t = 3.17$s (d) $t = 5$s

(e) $\tau = 0$s (f) $t = 1.59$s (g) $t = 3.19$s (h) $t = 5.32$s

Figure 4.12. The upper panels (a)-(d): reaching motion of a two-link robot arm using an arbitrary initial objective function without accounting for the obstacles. Here the obstacle is labeled by an orange object and the reaching target by a red star. From the left to right, we plot the configuration of the robot arm at different time instances during its motion with a random initial control objective function. The second-row panels (e)-(h): reaching motion of the robot arm using the objective function learned from the given waypoint in Table 4.7. Here the waypoint $\boldsymbol{q}^*(\tau_1)$ is shown in (e) by gray color. From left to right, we plot the configuration of the arm at different time instances during its motion. Please also find the video demo at https://wanxinjin.github.io/posts/lfsd.

successfully learn a valid control objective function for its motion to avoid obstacles. In Fig. 4.13, we also plot the loss value $L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D})$ versus the number of iterations.

4.7 Discussion

In this section, we provide further discussion about the proposed learning method in terms of sparse demonstration data, objective function parameterization, and learning convergence.

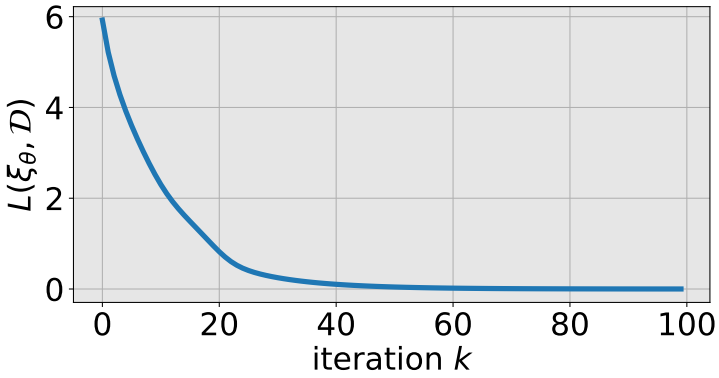


Figure 4.13. Loss versus iteration for robot arm learning.

4.7.1 Why do sparse demonstrations suffice?

As shown in both Sections 4.5 and 4.6, the proposed approach enables to learn a control objective function from only a few sparse demonstrations. We below provide one explanation of why use of sparse data can successfully recover an objective function.

Consider the problem in (4.16). For the optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}}$ produced by the time-warped optimal control system in (4.12), since we are only interested in the trajectory points $\boldsymbol{\xi}_{\boldsymbol{\theta}}(\tau_i)$ at the specified time instances τ_i $(1 \leq i \leq N)$, we discretize the time horizon of the optimal control system at these given time instances, and obtain the following discretized system [51]:

$$\text{dynamics: } \boldsymbol{x}_{i+1} = \bar{\boldsymbol{f}}(\boldsymbol{x}_i, \bar{\boldsymbol{u}}_i, \boldsymbol{\theta}), \quad \boldsymbol{x}_0 = \boldsymbol{x}(0), \tag{4.49a}$$

$$\text{objective: } J(\boldsymbol{\theta}) = \sum_{i=0}^{N-1} \bar{c}(\boldsymbol{x}_i, \bar{\boldsymbol{u}}_i, \boldsymbol{\theta}) + \bar{h}(\boldsymbol{x}_N, \bar{\boldsymbol{u}}_N, \boldsymbol{\theta}), \tag{4.49b}$$

where we denote $\boldsymbol{x}_i = \boldsymbol{x}(\tau_i)$, and discrete-time $\bar{\boldsymbol{f}}$ satisfies

$$\boldsymbol{x}_{i+1} = \bar{\boldsymbol{f}}(\boldsymbol{x}_i, \bar{\boldsymbol{u}}_i, \boldsymbol{\theta}) = \boldsymbol{x}_i + \int_{\tau_i}^{\tau_{i+1}} v_\beta(\tau) \boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau)) d\tau,$$

and the discrete-version of objective function satisfies

$$\bar{c}(\boldsymbol{x}_i, \bar{\boldsymbol{u}}_i, \boldsymbol{\theta}) = \int_{\tau_i}^{\tau_{i+1}} v_{\boldsymbol{\beta}}(\tau) c_{\boldsymbol{p}}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau)) d\tau,$$
$$\bar{h}(\boldsymbol{x}_N, \bar{\boldsymbol{u}}_N, \boldsymbol{\theta}) = \int_{\tau_N}^{T} v_{\boldsymbol{\beta}}(\tau) c_{\boldsymbol{p}}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau)) d\tau + h_{\boldsymbol{p}}(\boldsymbol{x}(T)).$$

Here the discrete input $\bar{\boldsymbol{u}}_i \in \mathbb{R}^d$ in $\bar{\boldsymbol{f}}$ may not necessarily has the same dimension as the control $\boldsymbol{u}(\tau) \in \mathbb{R}^n$ of the original system $\boldsymbol{f}$, e.g., $\bar{\boldsymbol{u}}_i$ contains all possible controls over the time range $[\tau_i, \tau_{i+1}]$, as [51]. The resulting optimal sequence $\{\boldsymbol{x}_{0:N}, \bar{\boldsymbol{u}}_{0:N}\}$ of the discrete-time optimal control system (4.49) satisfies the KKT conditions:

$$\boldsymbol{x}_{i+1} = \bar{\boldsymbol{f}}(\boldsymbol{x}_i, \bar{\boldsymbol{u}}_i, \boldsymbol{\theta}), \qquad i = 0, \cdots N-1, \tag{4.50a}$$
$$\boldsymbol{\lambda}_i = \frac{\partial \bar{c}}{\partial \boldsymbol{x}_i} + \frac{\partial \bar{\boldsymbol{f}}'}{\partial \boldsymbol{x}_i} \boldsymbol{\lambda}_{i+1}, \qquad i = 1, \cdots N-1, \tag{4.50b}$$
$$\boldsymbol{0} = \frac{\partial \bar{c}}{\partial \bar{\boldsymbol{u}}_i} + \frac{\partial \bar{\boldsymbol{f}}'}{\partial \bar{\boldsymbol{u}}_i} \boldsymbol{\lambda}_{i+1}, \qquad i = 0, \cdots N-1, \tag{4.50c}$$
$$\boldsymbol{\lambda}_N = \frac{\partial \bar{h}}{\partial \boldsymbol{x}_N}, \quad \frac{\partial \bar{h}}{\partial \bar{\boldsymbol{u}}_N} = \boldsymbol{0} \qquad i = N. \tag{4.50d}$$

The output of the discrete-time system (4.49) can be overloaded by $\boldsymbol{y}(\tau_i) = \boldsymbol{g}(\boldsymbol{x}_i, \bar{\boldsymbol{u}}_i)$. To simplify analysis, we further assume that the sparse demonstrations $\mathcal{D}$ in (4.5) correspond to an exact objective function (and time-warping function) with parameter $\boldsymbol{\theta}$, i.e., $\min L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D}) = 0$. Then,

$$\boldsymbol{y}^*(\tau_i) = \boldsymbol{g}(\boldsymbol{x}_i, \bar{\boldsymbol{u}}_i). \tag{4.51}$$

Given the sparse demonstrations $\mathcal{D}$ in (4.5), we can consider the recovery of an objective function to be a problem of solving a set of non-linear equations in (4.50) and (4.51), where the unknowns are $\{\boldsymbol{x}_{1:N}, \bar{\boldsymbol{u}}_{0:N}, \boldsymbol{\lambda}_{1:N}, \boldsymbol{\theta}\} \in \mathbb{R}^{2Nn+(N+1)d+(r+s)}$, and the total number of constraints are $2Nn + (N+1)d + No$. Here $(r+s)$ is the dimension of $\boldsymbol{\theta}$ and o is the dimension of $\boldsymbol{y}$. Thus, a necessary condition to compute $\{\boldsymbol{x}_{1:N}, \bar{\boldsymbol{u}}_{0:N}, \boldsymbol{\lambda}_{1:N}, \boldsymbol{\theta}\}$ uniquely requires the number of constraint equations to be no less than the number of unknowns, which leads to

$$N \geq \frac{r+s}{o}. \tag{4.52}$$

This has been empirically shown by experiments in Section 4.5.1, where the number of sparse demonstrations satisfy the above condition. On the other hand, if (4.52) is not fulfilled or the given sparse data $\mathcal{D}$ is of lower excitation (conceptually think of the persistent excitation in system identification), the unknowns then cannot be uniquely determined, which means that there might exist multiple $\boldsymbol{\theta}$ such that the trajectory passes through the sparse demonstrations. This has been shown in experiment in Sections 4.6.2 or 4.6.1, where we only provide one waypoint.

Note that the above discussion uses a perspective different from the technical development of this chapter to explain why sparse demonstrations can recover an objective function. This explanation however is limited as it fails to explain the case where sparse demonstrations are not realizable, i.e., $\min L(\boldsymbol{\xi}_{\boldsymbol{\theta}}, \mathcal{D}) > 0$, such as sub-optimal data, as demonstrated in Section 4.6.1 and 4.5.1. We leave this as a direction for future work, where we could formulate the problem in stochastic settings and explain data sparsity from the perspective of probability or information theory.

4.7.2 Choice of Parametric Objective Functions

We here discuss the choice of parametric objective functions based on different application scenarios.

Learning for Robot Motion Control

As shown in Section 4.5, when a robot learns from demonstrations for its motion control, a parameterized objective function can be selected as a weighted distance to the goal/target state together with the penalty for control efforts. This type of objective function is commonly used in tracking control problems [149] and model predictive control problems [103].

Learning for Obstacle Avoidance

As shown in Section 4.6.1, when a robot learns from sparse demonstrations in order to plan/control its motion for obstacle avoidance, the unknown objective function is set as a general parametric function that can represent global positions. For example, in (4.47a)

and (4.48a), we use general polynomial functions of states to represent the running cost function. The learned polynomial objective function will finally encode the information of the obstacles' global positions, based on which the robot then generates its motion to successfully avoid obstacles. Also, in these obstacle avoidance scenarios, the final cost (see (4.47b) and (4.48b)) is set to include the target/goal position that the human operator wants the robot to finally reach. However, this formulation will not allow a robot to generalize to dynamic environments, where obstacle locations may change over time. To handle dynamic environments, the cost function input must explicitly include information about obstacles, such as relative distance.

Using Universal Neural Network Objective Functions

When a human operator has little prior knowledge about the robot tasks and its dynamics constants, a universal method to represent a learnable objective function is to use a (deep) neural network, as we have demonstrated in Section 4.5.1. Despite its representation convenience, our experimental experience finds the following drawbacks of using neural network objective functions: (i) the great effort needed to specify the proper structure and hyper-parameters of a neural network, such as the number of layers or neural nodes and the type of activation functions, (ii) the lack of physical interpretability for the learned results, (iii) the relatively slower convergence in general, as empirically shown in Section 4.5.1, and (iv) the tendency of getting trapped in locally optimal solutions, which means that one has to carefully choose initial conditions for parameters of a neural network.

4.7.3 Choices of Sparse Demonstrations

Based on the applications in Sections 4.6.2 and 4.6.1, we have noted that few waypoints are sufficient to train a robot to accomplish the task of obstacle avoidance. However, it is also worth noting that a human demonstrator has to provide these few waypoints wisely in order to successfully teach the robot to learn to move around obstacles. For example, in the robot arm experiment in Section 4.6.2, if the single waypoint is placed too close to the initial or target configuration, the robot arm, even though it can still learn to pass through the

waypoint, will in other places crash into the obstacle. Thus, a wise choice of fewer waypoints requires the demonstrator's understanding of both the task and robot constraints in specific applications.

4.7.4 Convergence of the Proposed Learning Algorithm

The proposed learning algorithm is to solve the optimization problem in (4.16) using (projected) gradient descent. Generally, such a problem belongs to non-convex optimization, e.g., when one utilizes a deep neural network to represent the unknown objective function. For general non-convex optimization problems, it is known that finding the global minimum is generally difficult (if it is not impossible)[159], and (projected) gradient descent, with an appropriate step size (e.g., using the Armijo rule [159]), can provably converge to a stationary/critical point, i.e., a point at which the gradient of the loss function is zero, [159]. A stationary point could be global minima, local minima, or saddle points with worst-case initialization. Due to difficulty of finding global minima, the past research in non-convex optimization are mainly focused on how to overcome the convergence to saddle points. Very recently, new progress [160, 161, 162] in non-convex optimization shows that, under a very mild regularity, e.g., adding noise to data, convergence to saddle points is almost impossible, and gradient descent always converges to (local) minimizers for any random initialization.

When we pose further requirements, such as (strong) convexity and smoothness, on both the loss function (4.15) and the parametric optimal control system (4.12) with respect to both the system state-input trajectory and the parameter $\boldsymbol{\theta}$, convergence of the proposed learning algorithm to global minima could be guaranteed. This is because the proposed learning method is suited to the category of bi-level programming [163], where here the inner level is to solve an optimal control problem in (4.12) and the outer level to minimize the loss function (4.15); and [133] gives a proof of convergence to global minima for general bi-level programs. However, to prove global minima of our case, the convex and smooth requirements for the optimal control system (4.12) is too limited. As a future direction of this work, we will try to explore milder conditions that can ensure the global minima

convergence of the method, probably using the perspective of dynamical system or control theory.

4.8 Conclusions

In this chapter, we present an approach to learn an objective function from sparse demonstrations of an expert. The sparse demonstrations are given as few desired outputs of the robot's trajectory at some sparsely-located time instances specified by a human user. The proposed method enables the robot to jointly learn an objective function and a time-warping function such that its reproduced trajectory has minimal distance to the sparse demonstrations. The proposed technique of differential Pontryagin's Maximum Principle allows us to simultaneously learn a control objective function and a time-warping function by directly minimizing the Euclidean distance between the robot's reproduced trajectory and the given sparse demonstrations. The effectiveness and capability of the proposed method are demonstrated using multiple scenarios, including obstacle avoidance for a robot arm and a 6-DoF quadrotor maneuvering control. The results show that using only few sparse waypoints, a robot is able to learn a valid objective function to control its motion to successfully avoid obstacles.

5. LEARNING FROM DIRECTIONAL CORRECTIONS

Learning from demonstrations is an offline process: a human user first provides a robot with behavioral demonstrations in a onetime manner, then the robot learns a control policy or a control objective function off-line from the demonstrations. In this chapter, we develop a technique that enables a non-expert human user to teach a robot by incrementally improving the robot's motion. For instance, consider the example of a robot that plans motion under a (random) control objective function. While it is executing the motion, a human user who supervises the robot will find the robot's motion is not satisfactory; thus, the human user applies a correction to the robot during its motion execution. Then, the robot uses the correction to update its control objective function. This process of planning-correction-update repeats until the robot eventually achieves a control objective function such that its resulting trajectory agrees with the human user's expectation. In this learning procedure, the human's each correction does not necessarily move the robot to the optimal motion, but merely an incremental improvement of the robot's current motion towards the human's expectation, thus reducing the workload of a nonexpert user compared to learning from demonstrations

In this chapter, we present a new technique which enables a robot to learn a control objective function incrementally from human user's corrections. The human's corrections can be as simple as directional corrections—corrections that indicate the direction of a control change without indicating its magnitude—applied at some time instances during the robot's motion. We only assume that each of the human's corrections, regardless of its magnitude, points in a direction that improves the robot's current motion relative to an implicit objective function. The proposed method uses the direction of a correction to update the estimate of the robot control objective function. We establish the theoretical results to show that this process of incremental correction and update guarantees convergence of the learned objective function to the implicit one. The content of this chapter appears in [39], and the code and experiments developed for this chapter can be accessed at https://github.com/wanxinjin/Learning-from-Directional-Corrections.

5.1 Introduction

For tasks where robots work in proximity to human users, a robot is required to not only guarantee the accomplishment of a task but also complete it in a way that a human user prefers. Different users may have difference preferences about how the robot should perform the task. Such customized requirements usually lead to considerable workload of robot programming, which requires human users to have expertise to design and repeatedly tune robot's controllers until achieving satisfactory robot behaviors.

To circumvent the expertise requirement in traditional robot programming, learning from demonstrations (LfD) empowers a non-expert human user to program a robot by only providing demonstrations. In existing LfD techniques [139], a human user first provides a robot with behavioral demonstrations in a *one-time* manner, then the robot learns a control policy or a control objective function *off-line* from the demonstrations. Successful examples include autonomous driving [26], robot manipulation [34], and motion planning [138]. In some practical cases, the *one-time and offline nature* of LfD can introduce challenges. For example, when the demonstrations are insufficient to infer the objective function due to low data informativeness [12] or significant deviation from the optimal data [164], new demonstrations have to be re-collected and the robot has to be re-trained. Importantly, acquiring an optimal demonstration in a one-shot fashion for the systems of high degree-of-freedoms can be challenging [164], because the human demonstrator has to move the robot in all degrees-of-freedom in a spatially and temporally consistent manner.

In this work, we address the above challenges by developing a new programming scheme that enables a non-expert human user to program a robot by *incrementally improving* the robot's motion. For instance, consider the example of a robot that plans motion under a (random) control objective function. While it is executing the motion, a human user who supervises the robot will find the robot's motion is not satisfactory; thus, the human user applies a correction to the robot during its motion execution. Then, the robot uses the correction to update its control objective function. This process of planning-correction-update repeats until the robot eventually achieves a control objective function such that its resulting trajectory agrees with the human user's expectation. In this learning procedure,

the human's each correction does not necessarily move the robot to the optimal motion, but merely an *incremental improvement* of the robot's current motion towards the human's expectation, thus reducing the workload of a non-expert user when programming a robot compared to LfD. In addition to the incremental learning capability, the proposed learning from directional corrections technique in this work also has the following highlights.

1) The proposed method only requires human's *directional corrections*. A directional correction is a correction that only contains directional information and does not necessarily need to be magnitude-specific. For instance, for teaching a mobile robot, the directional corrections are simply as 'left' or 'right' without dictating how far the robot should move.

2) The human's directional corrections to the robot's motion can be *sparse*. That means that the corrections can be applied only at *sparse time instances* within the time horizon of the robot's motion. The learning is performed directly based on the sparse corrections, without attaining/retaining any intermediate corrected trajectory that may introduce inaccuracy.

3) Both theoretical results and experiments are established to show the convergence of the proposed learning algorithm. Specifically, we validate the method on two human-robot games and the results show that the proposed method enables a robot to efficiently learn a control objective function for the desired motion with few human's directional corrections.

5.1.1 Related Work

Offline Learning from Demonstrations

To learn a control objective function from demonstrations, the available approaches include inverse optimal control [11, 42, 147] and inverse reinforcement learning [3, 29, 33], where given optimal demonstrations, an objective function that explains such demonstrations is inferred and used for motion control and planning. Despite the significant progress achieved in theory and applications [13, 26, 34, 35, 141], LfD approaches could be inconvenient in some practical situations. First, demonstrations in LfD are usually given in a *one-time* manner and the learning process is usually performed *offline* after the demonstrations are obtained. In the case when the given demonstration data is insufficient to learn

the objective function from, such as low data informativeness as discussed in [12], or the demonstrations significantly deviates from the optimal ones, the data has to be re-collected and the whole learning process has to be re-run. Second, existing LfD techniques [3, 11, 29, 33, 42] normally assume optimality of the demonstration data, which is challenging to obtain for robots with high degree-of-freedoms. For example, when producing demonstrations for a humanoid robot, a human demonstrator has to account for the motion in all degrees in a spatially and temporally consistent manner. [164].

Online Learning from Feedback or Physical Corrections

Compared to offline LfD, learning from corrections or feedback enables a human user to incrementally correct the robot's current motion, making it more accessible for the non-expert users who cannot provide optimal demonstrations in a one-time manner [165]. The key assumption for learning from corrections or feedback is that the corrected robot's motion is better than that before the correction. Under this assumption, [164] proposes a co-active learning method, in which a robot receives human's feedback to update its objective function. The human's feedback includes the passive selection of a top-ranked robot trajectory or the active physical interference for providing a preferred robot trajectory. By defining a learning regret, which quantifies the *average* misalignment of the score values between the human's intended trajectory and robot's trajectory under the human's *implicit* objective function, the authors show the convergence of the regret. Since the regret is an average indicator over the entire learning process, one still cannot explicitly tell if the learned objective function is actually converging towards the human's implicit one.

Very recently, the authors in [38, 166, 167] approach learning from corrections from the perspective of a partially observable Markov decision process (POMDP), where human's corrections are viewed as the observations about the unknown objective function parameters. By approximating the observation model and applying maximum a posteriori estimation, they obtain a learning update that is similar to the co-active learning [164]. To handle the sparse corrections that a human user applies only at sparse time instances during the robot's motion, these methods apply the trajectory deformation technique [168] to interpret

each single-time-step correction through a human indented trajectory, i.e., a deformed robot trajectory. Although achieving promising results, choosing the hyper-parameters in the trajectory deformation is challenging, which can affect the learning performance [166]. In addition, these methods have not provided any convergence guarantee of the learning process.

Both the above co-active learning and POMDP-based learning require a dedicated setup or process to obtain the *human indented/feedback trajectories*. Specifically, in co-active learning, a robot is switched to the screening and zero-force gravity-compensation modes to obtain a human feedback trajectory, and in the POMDP-based method, the human intended trajectory is obtained by deforming the robot's current trajectory based on a correction using trajectory deformation method. These intermediate steps may introduce inaccurate artificial aspects to the learning process, which could lead to failure of the approach. For example, when a user physically corrects a robot, the magnitude of a correction, i.e., how much the correction should be, can be difficult to determine. If not chosen properly, the given correction may be overshot, i.e., too much correction. Such a overshooting correction can make the obtained human feedback trajectory violate the assumption of improving the robot's motion. In fact, as we will demonstrate in Sections 5.2 and 5.5.3, the more closer the robot is approaching to the expected trajectory, the more difficult the choice of a proper correction magnitude will be, which can lead to learning inefficiency. Also, for POMDP-based methods, when one applies the trajectory deformation, the choice of hyper-parameters will determine the shape of the human intended trajectory and thus finally affect the learning performance, as discussed in [166].

5.1.2 Contributions

This chapter develops a new method to learn a robot objective function incrementally from human's directional corrections. Compared to the existing methods above, the distinctions and contributions of the proposed method are stated as follows.

(1) The proposed method learns a robot control objective function only using the *direction information* of human's corrections. It only requires that a correction, regardless of magnitude, has a direction of incrementally improving robot's current

motion. As we will later show in Sections 5.2 and 5.5.3, the feasible corrections that satisfy such a requirement always account for half of the entire input space, making it more flexible for a human user to choose corrections from.

(2) Unlike existing learning techniques which usually require an intermediate setup to obtain a human indented trajectory, the proposed method learns a control objective function *directly from directional corrections*. The directional corrections can be *sparse*, i.e., the corrections only applied at some time instances within the time horizon of robot's motion.

(3) The proposed learning algorithm is developed based on the cutting plane technique, which has a straightforward intuitive geometric interpretation. We have established the theoretical results to show the convergence of the learned objective function to the human's implicit one.

The proposed method is validated by two human-robot games based on a two-link robot arm and a 6-DoF quadrotor maneuvering system, where a human player, by applying directional corrections, teaches the robot for motion control in environments with obstacles. The experiment results demonstrate that the proposed method enables a non-expert human player to train a robot to learn an effective control objective function for desired motion with few directional corrections.

In the following, Section 5.2 describes the problem. Section 5.3 proposes the main algorithm outline. Section 5.4 provides theoretical results of the algorithm and its detailed implementation. Numerical simulations and comparison are in Section 5.5. Section 5.6 presents the experiments on two human-robot games. Conclusions are drawn in Section 5.8.

5.2 Problem Formulation

Consider a robot with the following dynamics:

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t), \quad \text{with} \quad \boldsymbol{x}_0, \tag{5.1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^n$ is the robot state, $\boldsymbol{u}_t \in \mathbb{R}^m$ is the control input, $\boldsymbol{f} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ is differentiable, and $t = 1, 2, \cdots$ is the time step. As commonly used by objective learning methods such as [3, 11, 12, 29, 33, 38, 164, 165, 166, 167], we suppose that the robot control cost function obeys the following parameterized form

$$J(\boldsymbol{u}_{0:T}, \boldsymbol{\theta}) = \sum_{t=0}^{T} \boldsymbol{\theta}' \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t) + h(\boldsymbol{x}_{T+1}), \tag{5.2}$$

where $\boldsymbol{\phi} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^r$ is a vector of the *specified* features (or basis functions) for the running cost; $\boldsymbol{\theta} \in \mathbb{R}^r$ is a vector of weights, which are *tunable*; and $h(\boldsymbol{x}_{T+1})$ is the final cost that penalizes the final state $\boldsymbol{x}_{T+1}$. For a given choice of $\boldsymbol{\theta}$, the robot chooses a sequence of inputs $\boldsymbol{u}_{0:T}$ over the time horizon T by optimizing (5.2) subject to (5.1), producing a trajectory

$$\boldsymbol{\xi}_{\boldsymbol{\theta}} = \left\{ \boldsymbol{x}^{\boldsymbol{\theta}}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}}_{0:T} \right\}. \tag{5.3}$$

For the purpose of readiablity, we occasionally write the cost function (5.2) as $J(\boldsymbol{\theta})$.

For a specific task, suppose that a human's expectation of the robot's trajectory corresponds to an *implicit* cost function $J(\boldsymbol{\theta}^*)$ in the same form of (5.2) with $\boldsymbol{\theta}^*$. Here, we call $\boldsymbol{\theta}^*$ the *expected weight vector*. In general cases, a human user may neither explicitly write down the value of $\boldsymbol{\theta}^*$ nor demonstrate the corresponding optimal trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}^*}$ to the robot, but the human user can tell whether the robot's current trajectory is *satisfactory* or not. A trajectory of the robot is satisfactory if it minimizes $J(\boldsymbol{\theta}^*)$; otherwise, it is not satisfactory. In order for the robot to achieve $J(\boldsymbol{\theta}^*)$ (and thus generates a satisfactory trajectory), the human user is only able to make corrections to the robot during its motion, based on which the robot updates its guess of $\boldsymbol{\theta}$ towards $\boldsymbol{\theta}^*$.

The process for a robot to learn from human's corrections is iterative. Each iteration basically includes three steps: planning, correction and update. Let $k = 1, 2, 3, \cdots$, denote the iteration index and let $\boldsymbol{\theta}_k$ denote the robot's weight vector guess at iteration k. At $k = 1$, the robot is initialized with an arbitrary weight vector guess $\boldsymbol{\theta}_1$. At iteration $k = 1, 2, 3, \cdots$, the robot first performs trajectory *planning*, i.e. achieves $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by minimizing the cost function $J(\boldsymbol{\theta}_k)$ in (5.2) subject to its dynamics (5.1). During robot's execution of $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$,

the human user gives a *correction* denoted by $\boldsymbol{a}_{t_k} \in \mathbb{R}^m$ to the robot in its input space. Here, $t_k \in \{0, 1, \cdots, T\}$, called *correction time*, indicates at which time step within the horizon T the correction is made. After receiving $\boldsymbol{a}_{t_k}$, the robot then performs *update*, i.e. change its guess $\boldsymbol{\theta}_k$ to $\boldsymbol{\theta}_{k+1}$ according to an update rule to be developed later.

Each human's correction $\boldsymbol{a}_{t_k}$ is assumed to satisfy the following condition:

$$\left\langle -\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*),\ \bar{\boldsymbol{a}}_k \right\rangle > 0, \quad k = 1, 2, 3, \cdots . \tag{5.4}$$

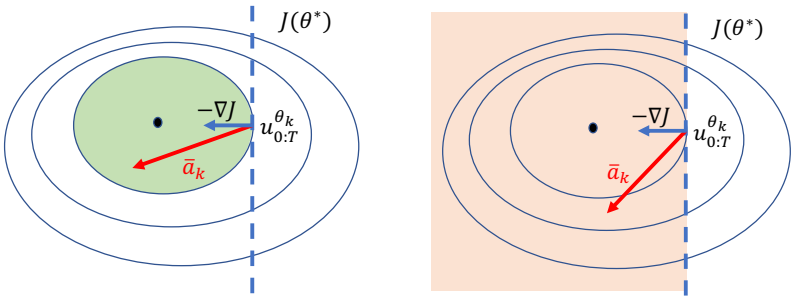
Here

$$\bar{\boldsymbol{a}}_k = \begin{bmatrix} \mathbf{0}' & \cdots & \boldsymbol{a}_{t_k}' & \cdots, \mathbf{0}' \end{bmatrix}' \in \mathbb{R}^{m(T+1)}, \tag{5.5}$$

with $\boldsymbol{a}_{t_k}$ being the t_k-th entry and $\mathbf{0} \in \mathbb{R}^m$ else; $\langle \cdot, \cdot \rangle$ is the dot product; and $-\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)$ is the gradient-descent of $J(\boldsymbol{\theta}^*)$ with respect to $\boldsymbol{u}_{0:T}$ evaluated at robot's current $\boldsymbol{\xi}_{\boldsymbol{\theta}_k} = \{\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}\}$. Note that the condition in (5.4) does not require a specific value to the magnitude of $\boldsymbol{a}_{t_k}$ but requires its direction roughly around the gradient-descent direction of $J(\boldsymbol{\theta}^*)$. Such correction aims to guide the robot's trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ towards reducing its cost under $J(\boldsymbol{\theta}^*)$ unless the trajectory is satisfactory. Thus, we call $\boldsymbol{a}_{t_k}$ satisfying (5.4) the *incremental directional correction.*

The **problem of interest** is to develop a rule to update the robot's weight vector guess $\boldsymbol{\theta}_k$ to $\boldsymbol{\theta}_{k+1}$ such that $\boldsymbol{\theta}_k$ converges to $\boldsymbol{\theta}^*$ as $k = 1, 2, 3, \cdots$, with the human's directional corrections $\boldsymbol{a}_{t_k}$ under the assumption (5.4).

Remark. *We assume that human user's corrections $\boldsymbol{a}_{t_k} \in \mathbb{R}^m$ are in the robot's input space, which means that $\boldsymbol{a}_{t_k}$ can be directly added to the robot's input $\boldsymbol{u}_{t_k}$. This can be satisfied in some cases such as autonomous driving, where a user directly manipulates the steering angle of a vehicle. For other cases where the corrections are not readily in the robot's input space, this requirement could be fulfilled through certain human-robot interfaces, which translate the correction signals into the input space. Then, $\boldsymbol{a}_{t_k}$ denotes the translated correction. The reason why we do not consider the corrections in the robot's state space is that 1) the input corrections may be easier in implementation, and 2) the corrections in the state space can be infeasible for some under-actuated robot systems [169].*



(a) Feasible region (green) for magnitude corrections

(b) Feasible region (red) for directional correction

Figure 5.1. Magnitude corrections v.s. directional corrections. The contour lines and the optimal/satisfactory trajectory (black dot) of the human's implicit cost function $J(\boldsymbol{\theta}^*)$ are plotted. (a): the green region (a sublevel set) shows all feasible magnitude corrections $\bar{\boldsymbol{a}}_k$ that satisfy $J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k} + \bar{\boldsymbol{a}}_k, \boldsymbol{\theta}^*) < J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)$. (b): the orange region (half of the input space) shows all feasible directional corrections $\bar{\boldsymbol{a}}_k$ that satisfy $\langle -\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*), \bar{\boldsymbol{a}}_k \rangle > 0$.

Remark. *The assumption in (5.4) on human's correction $\boldsymbol{a}_{t_k}$ is less restrictive than the one in [38, 164, 166, 167], which requires the cost of the corrected robot's trajectory $\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k} + \bar{\boldsymbol{a}}_k$ is lower than that of original $\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}$, i.e., $J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k} + \bar{\boldsymbol{a}}_k, \boldsymbol{\theta}^*) < J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)$. As shown in Fig. 5.1, this requirement usually leads to constraints in corrections' magnitudes. This is because to guarantee $J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k} + \bar{\boldsymbol{a}}_k, \boldsymbol{\theta}^*) < J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)$, $\|\bar{\boldsymbol{a}}_k\|$ has to be chosen from the $J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)$-sublevel set of $J(\boldsymbol{\theta}^*)$, as marked by the green region. Furthermore, this region will shrink as it gets close to the optimal trajectory (in black dot), thus making $\|\bar{\boldsymbol{a}}_k\|$ more difficult to choose when the robot's trajectory is near satisfactory one. In contrast, the directional corrections satisfying (5.4) always account for half of the entire input space. A human can choose any correction as long as its direction lies in the half space with gradient-descent of $J(\boldsymbol{\theta}^*)$. Thus, (5.4) is more likely to be satisfied especially for non-expert users.*

5.3 Algorithm Outline and Geometric Interpretation

In this section, we will present the outline of the proposed main algorithm for a robot to learn from human's incremental directional corrections and then provide a geometric

interpretation of the main algorithm. First, we present further analysis on the directional corrections.

5.3.1 Equivalent Conditions for Directional Corrections

Before developing the learning procedure, we will show that the assumption in (5.4) is equivalent to a linear inequality posed on the unknown expected weight vector $\boldsymbol{\theta}^*$, as stated in the following lemma.

Lemma 5.3.1. *Suppose that the robot's current weight vector guess is $\boldsymbol{\theta}_k$, and its motion trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}=\{\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}\}$ is a result of minimizing the cost function $J(\boldsymbol{\theta}_k)$ in (5.2) subject to dynamics in (5.1). For $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, given a human's incremental directional correction $\boldsymbol{a}_{t_k}$ satisfying (5.4), one has the following inequality equation:*

$$\langle \boldsymbol{h}_k, \boldsymbol{\theta}^* \rangle + b_k < 0, \quad k = 1, 2, 3 \cdots, \tag{5.6}$$

with

$$\boldsymbol{h}_k = \boldsymbol{H}_1'(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})\bar{\boldsymbol{a}}_k \ \in \mathbb{R}^r, \tag{5.7a}$$

$$b_k = \bar{\boldsymbol{a}}_k' \boldsymbol{H}_2(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}) \nabla h(\boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}) \ \in \mathbb{R}. \tag{5.7b}$$

Here, $\bar{\boldsymbol{a}}_k$ is defined in (5.5); $\nabla h(\boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k})$ is the gradient of the final cost $h(\boldsymbol{x}_{T+1})$ in (5.2) evaluated at $\boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}$; $\boldsymbol{H}_1(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})$ and $\boldsymbol{H}_2(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})$ are the coefficient matrices defined as follows:

$$\boldsymbol{H}_1(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}) = \begin{bmatrix} \boldsymbol{F}_u \boldsymbol{F}_x^{-1} \boldsymbol{\Phi}_x + \boldsymbol{\Phi}_u \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix} \in \mathbb{R}^{m(T+1)\times r}, \tag{5.8a}$$

$$\boldsymbol{H}_2(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}) = \begin{bmatrix} \boldsymbol{F}_u \boldsymbol{F}_x^{-1} \boldsymbol{V} \\ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix} \in \mathbb{R}^{m(T+1)\times n}, \tag{5.8b}$$

with

$$\boldsymbol{F}_x = \begin{bmatrix} I & \frac{-\partial \boldsymbol{f}'}{\partial \boldsymbol{x}_1^{\boldsymbol{\theta}_k}} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & I & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & I & \frac{-\partial \boldsymbol{f}'}{\partial \boldsymbol{x}_{T\text{-}1}^{\boldsymbol{\theta}_k}} \\ \mathbf{0} & \mathbf{0} & \cdots & & I \end{bmatrix}, \quad \boldsymbol{\Phi}_x = \begin{bmatrix} \frac{\partial \phi'}{\partial \boldsymbol{x}_1^{\boldsymbol{\theta}_k}} \\ \frac{\partial \phi'}{\partial \boldsymbol{x}_2^{\boldsymbol{\theta}_k}} \\ \vdots \\ \frac{\partial \phi'}{\partial \boldsymbol{x}_T^{\boldsymbol{\theta}_k}} \end{bmatrix}, \tag{5.9a}$$

$$\boldsymbol{F}_u = \begin{bmatrix} \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_0^{\boldsymbol{\theta}_k}} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_1^{\boldsymbol{\theta}_k}} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_{T\text{-}1}^{\boldsymbol{\theta}_k}} \end{bmatrix}, \quad \boldsymbol{\Phi}_u = \begin{bmatrix} \frac{\partial \phi'}{\partial \boldsymbol{u}_0^{\boldsymbol{\theta}_k}} \\ \frac{\partial \phi'}{\partial \boldsymbol{u}_1^{\boldsymbol{\theta}_k}} \\ \vdots \\ \frac{\partial \phi'}{\partial \boldsymbol{u}_{T\text{-}1}^{\boldsymbol{\theta}_k}} \end{bmatrix}, \tag{5.9b}$$

$$\boldsymbol{V} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_T^{\boldsymbol{\theta}_k}} \end{bmatrix}'. \tag{5.9c}$$

In above, the dimensions of the matrices are $\boldsymbol{F}_x \in \mathbb{R}^{nT \times nT}$, $\boldsymbol{F}_u \in \mathbb{R}^{mT \times nT}$, $\boldsymbol{\Phi}_x \in \mathbb{R}^{nT \times r}$, $\boldsymbol{\Phi}_u \in \mathbb{R}^{mT \times r}$, $\boldsymbol{V} \in \mathbb{R}^{nT \times n}$. *For a general differentiable function* $\boldsymbol{g}(\boldsymbol{x})$ *and a specific* $\boldsymbol{x}^*$, $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}^*}$ *denotes the Jacobian matrix of* $\boldsymbol{g}(\boldsymbol{x})$ *evaluated at* $\boldsymbol{x}^*$.

Proof. The proof of Lemma 5.3.1 consists of two steps: first, we will derive the explicit form of the gradient quantity $\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)$, and second, we will show that (5.4) can be re-written as (5.3.1).

Consider the robot's current trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k} = \{\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}\}$, which satisfies the robot dynamics constraint in (5.1). For any $t = 0, 1, \cdots, T$, define the infinitesimal increments $(\delta \boldsymbol{x}_t, \delta \boldsymbol{u}_t)$ at the state and input $(\boldsymbol{x}_t^{\boldsymbol{\theta}_k}, \boldsymbol{u}_t^{\boldsymbol{\theta}_k})$, respectively. By linearizing the dynamics (5.1) around $(\boldsymbol{x}_t^{\boldsymbol{\theta}_k}, \boldsymbol{u}_t^{\boldsymbol{\theta}_k})$, we have

$$\delta \boldsymbol{x}_{t+1} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}_k}} \delta \boldsymbol{x}_t + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}_k}} \delta \boldsymbol{u}_t, \tag{5.10}$$

where $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_t^{\boldsymbol{\theta}_k}}$ and $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}_k}}$ are the Jacobian matrices of $\boldsymbol{f}$ with respect to $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$, respectively, evaluated at $(\boldsymbol{x}_t^{\boldsymbol{\theta}_k}, \boldsymbol{u}_t^{\boldsymbol{\theta}_k})$. By stacking (5.10) for all $t = 0, 1, \cdots, T$ and also noting $\delta \boldsymbol{x}_0 = \mathbf{0}$ (because $\boldsymbol{x}_0^{\boldsymbol{\theta}_k}$ is given), we have the following compact matrix from

$$-\boldsymbol{A}' \delta \boldsymbol{x}_{1:T+1} + \boldsymbol{B}' \delta \boldsymbol{u}_{0:T} = \mathbf{0}, \tag{5.11}$$

with $\delta \boldsymbol{x}_{1:T+1}=[\delta \boldsymbol{x}_1', \cdots, \delta \boldsymbol{x}_{T+1}']'$, $\delta \boldsymbol{u}_{1:T}=[\delta \boldsymbol{u}_1', \cdots, \delta \boldsymbol{u}_T']'$,

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{F}_x & -\boldsymbol{V} \\ \mathbf{0} & I \end{bmatrix}, \quad \text{and} \quad \boldsymbol{B} = \begin{bmatrix} \boldsymbol{F}_u & \mathbf{0} \\ \mathbf{0} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix}. \tag{5.12}$$

Here I is $n \times n$ identity matrix, $\boldsymbol{F}_x$ and $\boldsymbol{F}_u$ are defined in (5.9). Due to the increments $\delta \boldsymbol{x}_{1:T+1}$ and $\delta \boldsymbol{u}_{1:T}$, the change of the value of the cost $J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)$ in (5.2), denoted as $\delta J(\boldsymbol{\theta}^*)$, can be written as

$$\delta J(\boldsymbol{\theta}^*) = \boldsymbol{C}' \delta \boldsymbol{x}_{1:T+1} + \boldsymbol{D}' \delta \boldsymbol{u}_{0:T}, \quad \text{with} \tag{5.13}$$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{\Phi}_x \boldsymbol{\theta}^* \\ \frac{\partial h'}{\partial \boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}} \end{bmatrix} \quad \text{and} \quad \boldsymbol{D} = \begin{bmatrix} \boldsymbol{\Phi}_u \boldsymbol{\theta}^* \\ \frac{\partial \phi'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \boldsymbol{\theta}^* \end{bmatrix}, \tag{5.14}$$

with $\boldsymbol{\Phi}_x$ and $\boldsymbol{\Phi}_u$ are defined in (5.9). Considering $\boldsymbol{A}$ is always invertible, we solve for $\delta \boldsymbol{x}_{1:T+1}$ from (5.11) and then submit it to (5.13), yielding

$$\delta J(\boldsymbol{\theta}^*) = \boldsymbol{C}' \delta \boldsymbol{x}_{1:T+1} + \boldsymbol{D}' \delta \boldsymbol{u}_{0:T} = \Big(\boldsymbol{C}' (\boldsymbol{A}^{-1})' \boldsymbol{B}' + \boldsymbol{D}' \Big) \delta \boldsymbol{u}_{0:T}. \tag{5.15}$$

Thus, we have

$$\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*) = \boldsymbol{B} \boldsymbol{A}^{-1} \boldsymbol{C} + \boldsymbol{D}. \tag{5.16}$$

The above (5.16) can be further written as

$$\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*) = \boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{C} + \boldsymbol{D} = \begin{bmatrix} \boldsymbol{F}_u & \boldsymbol{0} \\ \boldsymbol{0} & \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_x & -\boldsymbol{V} \\ \boldsymbol{0} & I \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\Phi}_x \boldsymbol{\theta}^* \\ \frac{\partial h'}{\partial \boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Phi}_u \boldsymbol{\theta}^* \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \boldsymbol{\theta}^* \end{bmatrix}$$
$$= \begin{bmatrix} \boldsymbol{F}_u \boldsymbol{F}_x^{-1} \boldsymbol{\Phi}_x \boldsymbol{\theta}^* + \boldsymbol{\Phi}_u \boldsymbol{\theta}^* + \boldsymbol{F}_u \boldsymbol{F}_x^{-1} \boldsymbol{V} \frac{\partial h'}{\partial \boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}} \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \boldsymbol{\theta}^* + \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \frac{\partial h'}{\partial \boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}} \end{bmatrix}, \tag{5.17}$$

where we have used Schur complement to compute the inverse of the block matrix $\boldsymbol{A}$. Using the definition in (5.8), (5.17) can be rewritten as

$$\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*) = \boldsymbol{H}_1(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})\boldsymbol{\theta}^* + \boldsymbol{H}_2(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}) \nabla h(\boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}). \tag{5.18}$$

Substituting (5.18) into the assumption (5.4) and also considering the definitions in (5.7), we obtain

$$\left\langle \nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*),\ \bar{\boldsymbol{a}}_k \right\rangle = \langle \boldsymbol{h}_k, \boldsymbol{\theta}^* \rangle + b_k < 0, \tag{5.19}$$

which leads to (5.6). This completes the proof. □

In Lemma 5.3.1, $\boldsymbol{h}_k$ and b_k in (5.7) are known and depend on both human's correction $\boldsymbol{a}_{t_k}$ and robot's motion trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k} = \{\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}\}$. The above Lemma 5.3.1 states that each incremental directional correction $\boldsymbol{a}_{t_k}$ can be equivalently converted to an inequality constraint on the unknown $\boldsymbol{\theta}^*$.

Remark. *$\boldsymbol{H}_1(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})$ and $\boldsymbol{H}_2(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})$ in Lemma 5.3.1 also appear in Chapter 2 and [12], in which they are shown to be efficiently computed iteratively based on $(\boldsymbol{x}_t^{\boldsymbol{\theta}_k}, \boldsymbol{u}_t^{\boldsymbol{\theta}_k})$, $t = 0, 1, \cdots, T$. Specifically, Define and initialize*

$$\begin{aligned} \boldsymbol{H}_1(\boldsymbol{x}_{0:1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:1}^{\boldsymbol{\theta}_k}) &= \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_0^{\boldsymbol{\theta}_k}} \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}_1^{\boldsymbol{\theta}_k}} + \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}_0^{\boldsymbol{\theta}_k}}, \\ \boldsymbol{H}_2(\boldsymbol{x}_{0:1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:1}^{\boldsymbol{\theta}_k}) &= \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_0^{\boldsymbol{\theta}_k}} \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}_1^{\boldsymbol{\theta}_k}}, \end{aligned} \tag{5.20a}$$

Perform the iteration with each next state-input $(\boldsymbol{x}_{t+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{t+1}^{\boldsymbol{\theta}_k})$ *until* $t = T-1$

$$\begin{aligned}
\boldsymbol{H}_1(\boldsymbol{x}_{0:t+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:t+1}^{\boldsymbol{\theta}_k}) &= \begin{bmatrix} \boldsymbol{H}_1(\boldsymbol{x}_{0:t}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:t}^{\boldsymbol{\theta}_k}) + \boldsymbol{H}_2(\boldsymbol{x}_{0:t}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{1:t}^{\boldsymbol{\theta}_k}) \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}_{t+1}^{\boldsymbol{\theta}_k}} \\ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}_k}} \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{x}_{t+1}^{\boldsymbol{\theta}_k}} + \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}_k}} \end{bmatrix}, \\
\boldsymbol{H}_2(\boldsymbol{x}_{0:t+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:t+1}^{\boldsymbol{\theta}_k}) &= \begin{bmatrix} \boldsymbol{H}_2(\boldsymbol{x}_{0:t}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:t}) \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}_{t+1}^{\boldsymbol{\theta}_k}} \\ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_t^{\boldsymbol{\theta}_k}} \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{x}_{t+1}^{\boldsymbol{\theta}_k}} \end{bmatrix},
\end{aligned} \tag{5.20b}$$

Finally for $t = T$,

$$\begin{aligned}
\boldsymbol{H}_1(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}) &= \begin{bmatrix} \boldsymbol{H}_1(\boldsymbol{x}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T-1}^{\boldsymbol{\theta}_k}) \\ \frac{\partial \boldsymbol{\phi}'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix}, \\
\boldsymbol{H}_2(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}) &= \begin{bmatrix} \boldsymbol{H}_2(\boldsymbol{x}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T-1}^{\boldsymbol{\theta}_k}) \\ \frac{\partial \boldsymbol{f}'}{\partial \boldsymbol{u}_T^{\boldsymbol{\theta}_k}} \end{bmatrix}.
\end{aligned} \tag{5.20c}$$

The above iterative property facilitates the computation of $\boldsymbol{H}_1$ *and* $\boldsymbol{H}_2$ *by avoiding the inverse of the large matrix* $\boldsymbol{F}_x$ *in (5.8), significantly reducing computational cost in solving for (5.8).*

5.3.2 Outline of the Main Algorithm

In order to achieve $\boldsymbol{\theta}^*$, at each iteration k, we let $\boldsymbol{\Omega}_k \subseteq \boldsymbol{\Theta}$ denote a *weight search space* such that $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_k$ and $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_k$ for all $k = 1, 2, 3, \cdots$. This $\boldsymbol{\Omega}_k$ can be thought of as the possible location of $\boldsymbol{\theta}^*$, and $\boldsymbol{\theta}_k$ as a weight vector guess to $\boldsymbol{\theta}^*$. Rather than a rule to guide $\boldsymbol{\theta}_k$ towards $\boldsymbol{\theta}^*$, we will develop a rule to update $\boldsymbol{\Omega}_k$ to $\boldsymbol{\Omega}_{k+1}$ such that a useful scalar measure of the size of $\boldsymbol{\Omega}_k$ will converge to 0.

Main Algorithm (Outline): *In the proposed main algorithm, we initialize the weight search space* $\boldsymbol{\Omega}_0$ *to be*

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \in \mathbb{R}^r \mid -\underline{r}_i \leq [\boldsymbol{\theta}]_i \leq \bar{r}_i,\ i = 1, \cdots, r\}, \tag{5.21}$$

where $\underline{r}_i$ *and* $\bar{r}_i$ *are non-negative constants denoting the lower bound and upper bound for the* i*th entry in* $\boldsymbol{\theta}$ *denoted as* $[\boldsymbol{\theta}]_i$*, respectively. Here,* $\underline{r}_i$ *and* $\bar{r}_i$ *can be chosen large enough*

to include $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_0$. *The learning proceeds with each iteration* $k = 1, 2, \cdots$, *including the following steps:*

Step 1: *Choose a weight vector guess* $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$ *from the weight search space* $\boldsymbol{\Omega}_{k-1}$ *(We will discuss how to choose such* $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$ *in Section 5.4).*

Step 2: *The robot restarts and plans its motion trajectory* $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ *by solving an optimal control problem with the cost function* $J(\boldsymbol{\theta}_k)$ *and dynamics in (5.1). While the robot is executing* $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, *a human user applies a directional correction* $\boldsymbol{a}_{t_k}$ *at time* t_k. *Then, a hyperplane* $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$ *is obtained by (5.6)-(5.7).*

Step 3: *Update the weight search space* $\boldsymbol{\Omega}_{k-1}$ *to* $\boldsymbol{\Omega}_k$*:*

$$\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \boldsymbol{\Theta} \,|\, \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}\,. \tag{5.22}$$

We provide a few remarks to the above outline of the main algorithm. For initialization in (5.21), we allow entries of $\boldsymbol{\theta}$ to have different lower and upper bounds, which may come from the robot's rough pre-knowlege about the range of each weight. Simply but not necessarily, one could initialize

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \in \mathbb{R}^r \mid \|\boldsymbol{\theta}\|_\infty \leq R\}, \tag{5.23}$$

where

$$R = \max\{\underline{r}_i,\ \bar{r}_i,\ i = 1, \cdots, r\}. \tag{5.24}$$

In Step 1, one chooses $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$. Soon we will show $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_k$ for all $k = 1, 2, 3, \cdots$. Thus, one will expect $\boldsymbol{\theta}_k$ to be closer to $\boldsymbol{\theta}^*$ if the main algorithm could make $\boldsymbol{\Omega}_k$ smaller. In fact, the weight search space $\boldsymbol{\Omega}_k$ is non-increasing because $\boldsymbol{\Omega}_k \subseteq \boldsymbol{\Omega}_{k-1}$ by (5.22) in Step 3. A careful choice of $\boldsymbol{\theta}_k$ to guarantee the strict reduction of a size measure of $\boldsymbol{\Omega}_k$ will be given in Section 5.4. In Step 2, the robot's trajectory planning is performed by solving an optimal control problem with the cost function $J(\boldsymbol{\theta}_k)$ in (5.2) and the dynamics constraint in (5.1). This can be done by many trajectory optimization methods such as [7] or existing optimal control solvers such as [50]. With the robot's trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ and the human's directional

correction $\boldsymbol{a}_{t_k}$, the hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$ can be obtained by (5.6)-(5.7). The detailed implementation of the main algorithm with the choice of $\boldsymbol{\theta}_k$ and termination criterion will be presented in next section.

The proposed main algorithm also leads to the following lemma:

Lemma 5.3.2. *Under the proposed main algorithm, one has*

$$\langle \boldsymbol{h}_k, \boldsymbol{\theta}_k \rangle + b_k = 0, \quad \forall\ k = 1, 2, 3, \cdots \tag{5.25}$$

and

$$\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_k, \quad \forall\ k = 1, 2, 3, \cdots \tag{5.26}$$

Proof. First, we prove (5.25). From Step 2 in the main algorithm, we know that the robot's current trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k} = \{\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}\}$ is a result of minimizing the cost function $J(\boldsymbol{\theta}_k)$. This means that $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ must satisfy the optimality condition (i.e., first order condition) of the optimal control problem with the cost function $J(\boldsymbol{\theta}_k)$ in (5.2) and dynamics in (5.1). Following a similar derivation from (5.11) to (5.18) in the proof of Lemma 5.3.1, we can obtain

$$\mathbf{0} = \nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}_k) = \boldsymbol{H}_1(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})\boldsymbol{\theta}_k + \boldsymbol{H}_2(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})\nabla h(\boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}). \tag{5.27}$$

It is worth mentioning that the above optimality condition (5.27) is derived in [12]. Thus,

$$\begin{aligned} 0 = \left\langle \nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}_k), \bar{\boldsymbol{a}}_k \right\rangle &= \left\langle \boldsymbol{H}_1(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})\boldsymbol{\theta}_k, \bar{\boldsymbol{a}}_k \right\rangle + \left\langle \boldsymbol{H}_2(\boldsymbol{x}_{0:T+1}^{\boldsymbol{\theta}_k}, \boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k})\nabla h(\boldsymbol{x}_{T+1}^{\boldsymbol{\theta}_k}), \bar{\boldsymbol{a}}_k \right\rangle \\ &= \langle \boldsymbol{h}_k, \boldsymbol{\theta}_k \rangle + b_k, \end{aligned} \tag{5.28}$$

where the second line is due to the definition of hyperplane in (5.7). This completes the proof of (5.25).

Next, we prove (5.26). We use proof by induction. By the main algorithm, we know $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_0$ for $k = 0$. Assume that $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_{k-1}$ holds for the $(k-1)$-th iteration. According to Step 3 in the main algorithm, we have the relationship

$$\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \boldsymbol{\Theta} \,|\, \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}. \tag{5.29}$$

In order to prove $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_k$ we only need to show that

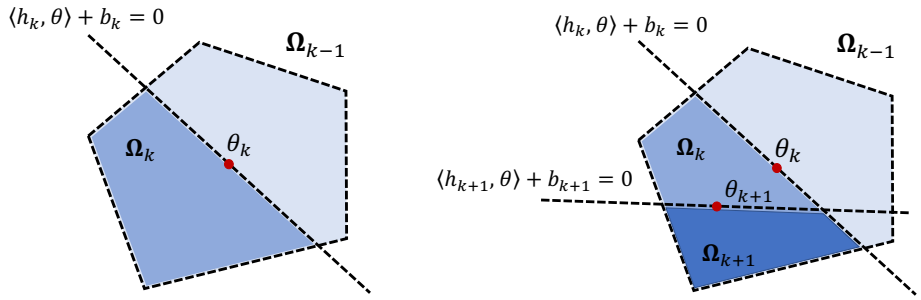
$$\langle \boldsymbol{h}_k, \boldsymbol{\theta}^* \rangle + b_k < 0, \tag{5.30}$$

which is true according to (5.6) in Lemma 5.3.1. Thus, $\boldsymbol{\theta}^* \in \boldsymbol{\Omega}_k$ also holds at the kth iteration. Thus, we conclude that (5.26) holds. This completes the proof of Lemma 5.3.2. □

Lemma 5.3.2 has intuitive geometric explanations. Note that (5.25) suggests $\boldsymbol{\theta}_k$ is always in the hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$. Moreover, (5.26) suggests that although the proposed algorithm directly updates the weight search space $\boldsymbol{\Omega}_k$, the expected weight vector $\boldsymbol{\theta}^*$ always lies in $\boldsymbol{\Omega}_k$. Intuitively, the smaller the search space $\boldsymbol{\Omega}_k$ is, the closer $\boldsymbol{\theta}^*$ is to $\boldsymbol{\theta}_k$.

5.3.3 Geometric Interpretation to Updating Search Space

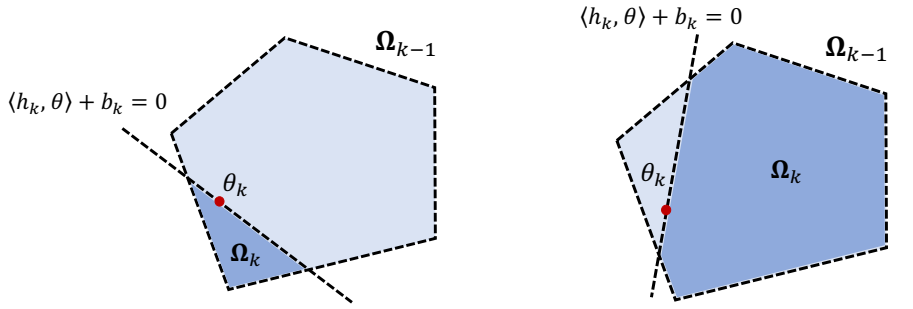
In this part, we will provide an interpretation of the proposed main algorithm through a geometric perspective. For simplicity of illustrations, we assume $\boldsymbol{\theta} \in \mathbb{R}^2$ in this subsection.



(a) At k-th iteration (b) At $(k+1)$-th iteration

Figure 5.2. Illustration of updating $\boldsymbol{\Omega}_k$.

At kth iteration in Fig. 5.2a, a weight vector guess $\boldsymbol{\theta}_k$ (colored in red) is picked from the current weight search space $\boldsymbol{\Omega}_{k-1}$ (colored in light blue), i.e., $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$. By Step 2 in the main algorithm, we obtain a hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$ (in black dashed line), which cuts through the weight search space $\boldsymbol{\Omega}_{k-1}$ into two portions. By (5.25) in Lemma 5.3.2, we know that $\boldsymbol{\theta}_k$ also lies on this hyper-plane because $\langle \boldsymbol{h}_k, \boldsymbol{\theta}_k \rangle + b_k = 0$. By Step 3 in the main algorithm, we only keep one of the two cut portions, which is the interaction space between $\boldsymbol{\Omega}_{k-1}$ and the half space $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0$, and the kept portion will be used as the weight search space for the next iteration, that is, $\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \mid \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}$, as shown in the blue region in Fig. 5.2a. The above procedure repeats also for iteration $k+1$, as shown in the right panel of Fig. 5.2b, and finally produces a smaller search space $\boldsymbol{\Omega}_{k+1}$ colored in the darkest blue in Fig. 5.2b. From (5.22), one has $\boldsymbol{\Omega}_0 \supseteq \cdots \boldsymbol{\Omega}_{k-1} \supseteq \boldsymbol{\Omega}_k \supseteq \boldsymbol{\Omega}_{k+1} \supseteq \cdots$. Moreover, by (5.26) in Lemma 5.3.2, we note that the expected weight vector $\boldsymbol{\theta}^*$ is always inside $\boldsymbol{\Omega}_k$ whenever k is.



(a) A large cut from $\boldsymbol{\Omega}_{k-1}$ (b) A small cut from $\boldsymbol{\Omega}_{k-1}$

Figure 5.3. Illustration of how different directional corrections $\boldsymbol{a}_{t_k}$ affect the reduction of the weight search space $\boldsymbol{\Omega}_{k-1}$.

Besides the above geometric illustration, we also have the following observations:

(1) The key idea of the proposed main algorithm is to cut and remove the weight search space $\boldsymbol{\Omega}_{k-1}$ as each directional correction $\boldsymbol{a}_{t_k}$ is given. Thus, we always expect that $\boldsymbol{\Omega}_{k-1}$ can quickly diminish to a very small space as k increases, because thereby we can say that the robot's current guess $\boldsymbol{\theta}_k$ is close to the expected weight vector $\boldsymbol{\theta}^*$. As shown in Fig. 5.2, the reduction rate of $\boldsymbol{\Omega}_{k-1}$ depends on two factors: the human's directional correction $\boldsymbol{a}_{t_k}$, and how to choose $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$.

(2) From (5.7), we note that the human's directional correction $\boldsymbol{a}_{t_k}$ determines $\boldsymbol{h}_k$, which is the normal vector of hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$. When fixing the choice of the weight vector guess $\boldsymbol{\theta}_k$, we can think of the hyperplane rotates around $\boldsymbol{\theta}_k$ with different choices of $\boldsymbol{a}_{t_k}$, which finally results in different removals of $\boldsymbol{\Omega}_{k-1}$, as illustrated in Fig. 5.3.

(3) How to choose $\boldsymbol{\theta}_k$ from $\boldsymbol{\Omega}_{k-1}$ defines the specific position of the hyperplane $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k = 0$, because the hyperplane is always passing through $\boldsymbol{\theta}_k$ by Lemma 5.3.2. Thus, $\boldsymbol{\theta}_k$ also affects how $\boldsymbol{\Omega}_{k-1}$ is cut and removed. This can be illustrated by comparing Fig. 5.2a with Fig. 5.3a.

Based on the above discussions, the convergence of the proposed main algorithm is determined by the reduction of the weight search space $\boldsymbol{\Omega}_{k-1}$. This depends on both the human's directional corrections $\boldsymbol{a}_{t_k}$ (hard to be predicted by the robot) and the robot's choice of the weight vector guess $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$. In the next section, we will present a way for robot to choose $\boldsymbol{\theta}_k$ to guarantee the convergence of the proposed algorithm.

5.4 Algorithm Implementation with Convergence Analysis

In this section, we will specify the choice of $\boldsymbol{\theta}_k$, provide the convergence analysis of the main algorithm, and finally present a detailed implementation of the algorithm with termination criterion.

5.4.1 Choice of Search Space Center

Under the proposed main algorithm, at each iteration k, the weight search space $\boldsymbol{\Omega}_{k-1}$ is updated according to (5.22), i.e.,

$$\boldsymbol{\Omega}_k = \boldsymbol{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \boldsymbol{\Theta} \mid \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}.$$

In order to evaluate the reduction of the weight search space, it is straightforward to use the volume of the (closure) weight search space $\boldsymbol{\Omega}_k$, denoted as $\mathbf{Vol}(\boldsymbol{\Omega}_k)$, and the zero volume implies the convergence of the search space [170]. By $\boldsymbol{\Omega}_k \subseteq \boldsymbol{\Omega}_{k-1}$ in (5.22), we know that

$\mathbf{Vol}(\mathbf{\Omega}_k)$ is non-increasing. In the following we will further develop a way such that $\mathbf{Vol}(\mathbf{\Omega}_k)$ is strictly decreasing under the proposed algorithm; i.e., there exists a constant $0 \leq \alpha < 1$ such that

$$\mathbf{Vol}(\mathbf{\Omega}_k) \leq \alpha \, \mathbf{Vol}(\mathbf{\Omega}_{k-1}). \tag{5.31}$$

In order to achieve (5.31), we note that different choices of $\boldsymbol{\theta}_k \in \mathbf{\Omega}_{k-1}$ will lead to different reduction of $\mathbf{\Omega}_{k-1}$: as indicated in Fig. 5.3a, a large volume reduction from $\mathbf{\Omega}_{k-1}$ to $\mathbf{\Omega}_k$ is achieved while the choice of $\boldsymbol{\theta}_k$ in Fig. 5.3b leads to a very small volume reduction. This observation motivates us that to avoid a very small volume reduction, one intuitively chooses $\boldsymbol{\theta}_k$ at the *center* of the weight search space $\mathbf{\Omega}_{k-1}$. Specifically, we use the center of the maximum volume ellipsoid inscribed within the search space as defined below.

Definition 5.4.1 (Maximum Volume inscribed Ellipsoid [41])**.** *Given a compact convex set $\mathbf{\Omega}$, the maximum volume ellipsoid (MVE) inscribed within $\mathbf{\Omega}$, defined as $\boldsymbol{E}$, is represented by*

$$\boldsymbol{E} = \{\bar{B}\boldsymbol{\theta} + \bar{\boldsymbol{d}} \,|\, \|\boldsymbol{\theta}\|_2 \leq 1\}. \tag{5.32}$$

Here, $\bar{B} \in \mathbb{S}^r_{++}$ (i.e., a $r \times r$ positive definite matrix); $\bar{\boldsymbol{d}} \in \mathbb{R}^r$ is called the center of $\boldsymbol{E}$; and $\bar{B}$ and $\bar{\boldsymbol{d}}$ solve the optimization:

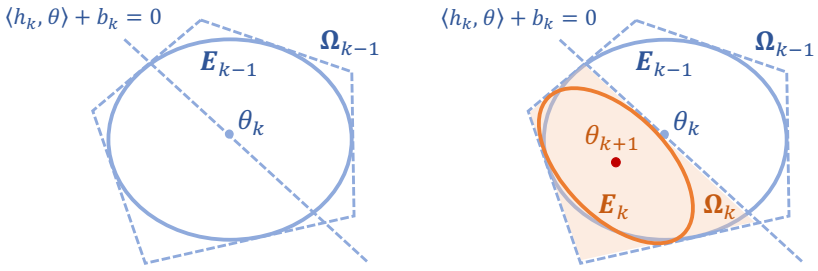
$$\begin{aligned} \max\nolimits_{\boldsymbol{d}, B \in \mathbb{S}^r_{++}} \quad & \log\det B \\ s.t. \quad & \sup\nolimits_{\|\boldsymbol{\theta}\|_2 \leq 1} I_{\mathbf{\Omega}}(B\boldsymbol{\theta} + \boldsymbol{d}) \leq 0, \end{aligned} \tag{5.33}$$

where $I_{\mathbf{\Omega}}(\boldsymbol{\theta}) = 0$ for $\boldsymbol{\theta} \in \mathbf{\Omega}$ and $I_{\mathbf{\Omega}}(\boldsymbol{\theta}) = \infty$ for $\boldsymbol{\theta} \notin \mathbf{\Omega}$.

Based on Definition 5.4.1, we let $\boldsymbol{E}_k$ denote the MVE inscribed within $\mathbf{\Omega}_k$ with $\boldsymbol{d}_k$ denoting the center of $\boldsymbol{E}_k$. For the choice of $\boldsymbol{\theta}_{k+1}$ at iteration $k+1$, we choose the weight vector guess

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{d}_k \tag{5.34}$$

as illustrated in Fig. 5.4. Other choices for $\boldsymbol{\theta}_{k+1}$ as a center of the search space are discussed in Section 5.7.



(a) Center of MVE in $\boldsymbol{\Omega}_{k-1}$ (b) Center of MVE in $\boldsymbol{\Omega}_k$

Figure 5.4. Illustration of choosing weight vector guess $\boldsymbol{\theta}_{k+1}$ as the center of MVE $\boldsymbol{E}_k$ inscribed in weight search space $\boldsymbol{\Omega}_k$.

We now present a computational method to achieve $\boldsymbol{d}_k$, i.e., the center of MVE $\boldsymbol{E}_k$ inscribe within $\boldsymbol{\Omega}_k$. Recall that in the proposed main algorithm, the initialization of $\boldsymbol{\Omega}_0$ in (5.21) is

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \in \mathbb{R}^r \mid -\underline{r}_i \leq [\boldsymbol{\theta}]_i \leq \bar{r}_i,\ i = 1, \cdots, r\},$$

with $[\boldsymbol{\theta}]_i$ the ith entry of $\boldsymbol{\theta}$. This can be equivalently rewritten as a set of linear inequalities:

$$\boldsymbol{\Omega}_0 = \left\{ \boldsymbol{\theta} \;\middle|\; \begin{array}{c} \langle \boldsymbol{e}_i, \boldsymbol{\theta} \rangle - \bar{r}_i \leq 0 \\ -\langle \boldsymbol{e}_i, \boldsymbol{\theta} \rangle - \underline{r}_i \leq 0 \end{array},\quad i = 1, \cdots, r \right\}, \tag{5.35}$$

where $\boldsymbol{e}_i$ is the unit vector with the ith entry equal to 1. Then, following the update in (5.22), $\boldsymbol{\Omega}_k$ is also a compact polytope, which can be written as

$$\boldsymbol{\Omega}_k = \left\{ \boldsymbol{\theta} \;\middle|\; \begin{array}{l} \langle \boldsymbol{e}_i, \boldsymbol{\theta} \rangle - \bar{r}_i \leq 0,\ i = 1, \cdots, r; \\ -\langle \boldsymbol{e}_i, \boldsymbol{\theta} \rangle - \underline{r}_i \leq 0,\ i = 1, \cdots, r; \\ \langle \boldsymbol{h}_j, \boldsymbol{\theta} \rangle + b_j < 0,\ j = 1, \cdots, k \end{array} \right\}. \tag{5.36}$$

As a result, in (5.33), solving the center $\boldsymbol{d}_k$ of the MVE $\boldsymbol{E}_k$ inscribed within $\boldsymbol{\Omega}_k$ becomes a convex programming [41], as stated by the following lemma.

Lemma 5.4.1. *For a polytope $\boldsymbol{\Omega}_k$ in (5.36), the center $\boldsymbol{d}_k$ of the MVE $\boldsymbol{E}_k$ inscribed within $\boldsymbol{\Omega}_k$ can be solved by the following convex optimization:*

$$\begin{aligned} \min_{\boldsymbol{d}, B\in\mathbb{S}^r_{++}} \; & -\log\det B \\ s.t. \; & \|B\boldsymbol{e}_i\|_2 + \langle \boldsymbol{d}, \boldsymbol{e}_i\rangle \le \bar{r}_i, \;\; i{=}1,\cdots r, \\ & \|B\boldsymbol{e}_i\|_2 - \langle \boldsymbol{d}, \boldsymbol{e}_i\rangle \le \underline{r}_i, \;\; i{=}1,\cdots r, \\ & \|B\boldsymbol{h}_j\|_2 + \langle \boldsymbol{d}, \boldsymbol{h}_j\rangle \le -b_j, \;\; j{=}1,\cdots k. \end{aligned} \tag{5.37}$$

The proof of the above lemma can be found in Chapter 8.4.2 in [41, pp.414]. The above convex optimization can be efficiently solved by existing solver e.g. [171]. In practical implementation of solving (5.37), since the number of linear inequalities grows as the iteration k increases, which can increase computational cost, the mechanism for dropping some redundant inequalities in (5.36) can be adopted [170]. Dropping redundant inequalities does not change $\boldsymbol{\Omega}_k$ and its volume reduction (convergence). Please see how to identify the redundant inequalities in [170].

5.4.2 Exponential Convergence and Termination Criterion

In this part, we will investigate convergence of the volume of $\boldsymbol{\Omega}_k$ following the proposed main algorithm and its termination criterion for practical implementation.

Note that the convergence of the proposed algorithm relies on the reduction of $\mathbf{Vol}(\boldsymbol{\Omega}_k)$, which can be guaranteed by the following lemma:

Lemma 5.4.2. *Let $\boldsymbol{\theta}_k \in \mathbb{R}^r$ be chosen as the center of the MVE $\boldsymbol{E}_{k-1}$ inscribed within $\boldsymbol{\Omega}_{k-1}$. Then, the update (5.22) leads to*

$$\frac{\boldsymbol{Vol}(\boldsymbol{\Omega}_k)}{\boldsymbol{Vol}(\boldsymbol{\Omega}_{k-1})} \le (1-\frac{1}{r}). \tag{5.38}$$

Lemma 5.4.2 is a direct theorem from [172]. Lemma 5.4.2 indicates

$$\mathbf{Vol}\left(\boldsymbol{\Omega}_k\right) \le (1-\frac{1}{r})^k \, \mathbf{Vol}\left(\boldsymbol{\Omega}_0\right).$$

Thus, $\mathbf{Vol}(\mathbf{\Omega}_k) \to 0$ exponentially fast, that is, its convergence speed is as fast as $(1-\frac{1}{r})^k \to 0$ as $k \to \infty$.

In order to implement the main algorithm in practice, we will not only need the exponential convergence as established by Lemma 5.4.2, but also a termination criterion, which specifies the maximum number of iterations for a given requirement in terms of $\mathbf{Vol}(\mathbf{\Omega}_k)$. Thus we have the following theorem.

Theorem 5.4.1. *Suppose $\mathbf{\Omega}_0$ is given by (5.21), and at iteration k, $\boldsymbol{\theta}_k$ is chosen as the center $\boldsymbol{d}_{k-1}$ of MVE $\boldsymbol{E}_{k-1}$ inscribed in $\mathbf{\Omega}_{k-1}$. Given a termination condition*

$$\boldsymbol{Vol}(\mathbf{\Omega}_k) \leq (2\epsilon)^r$$

with ϵ a user-specified threshold, the main algorithm runs for $k \leq K$ iterations, namely, the algorithm terminates at most K iterations, where

$$K = \frac{r \log(R/\epsilon)}{-\log(1-1/r)}, \tag{5.39}$$

with R given in (5.24).

Proof. Initially, we have $\mathbf{Vol}(\mathbf{\Omega}_0) \leq (2R)^r$. From Lemma 5.4.2, after k iterations, we have

$$\mathbf{Vol}(\mathbf{\Omega}_k) \leq (1-\frac{1}{r})^k \mathbf{Vol}(\mathbf{\Omega}_0) \leq (1-\frac{1}{r})^k (2R)^r, \tag{5.40}$$

which yields to

$$\log \mathbf{Vol}(\mathbf{\Omega}_k) \leq k \log(1-\frac{1}{r}) + \log(2R)^r. \tag{5.41}$$

When $k = \frac{r \log(R/\epsilon)}{-\log(1-1/r)}$,

$$\log \mathbf{Vol}(\mathbf{\Omega}_k) \leq -r \log(R/\epsilon) + \log(2R)^r. \tag{5.42}$$

The above equation is simplified to

$$\log \mathbf{Vol}(\mathbf{\Omega}_k) \leq \log(2\epsilon)^r, \tag{5.43}$$

which means that the termination condition $\mathbf{Vol}(\mathbf{\Omega}_k) \leq (2\epsilon)^r$ is satisfied. This completes the proof. □

On the above Theorem 5.4.1 we have the following comments.

Remark. *Since both $\boldsymbol{\theta}^*$ and $\boldsymbol{\theta}_k$ are always within $\mathbf{\Omega}_k$ for any $k = 1, 2, 3 \cdots$ by Lemma 5.3.2, the user-specified threshold ϵ in the termination condition $\boldsymbol{Vol}(\mathbf{\Omega}_k) \leq (2\epsilon)^r$ can be understood as an indicator of a distance between the expected weight vector $\boldsymbol{\theta}^*$ (usually unknown in practice) and the robot's weight vector guess $\boldsymbol{\theta}_k$. The threshold ϵ is set based on the desired learning accuracy.*

5.4.3 Implementation of the Main Algorithm

By the termination criterion in Theorem 5.4.1 and the choice of $\boldsymbol{\theta}_k$ in (5.34), one could implement the main algorithm in details as presented in Algorithm 8.

Algorithm 8: Learning from incremental directional corrections

Input: Specify a termination threshold ϵ and use it to compute the maximum iteration K by (5.39).

Initialization: Initial weight search space $\mathbf{\Omega}_0$ in (5.21).

for $k = 1, 2, \cdots, K$ **do**

- Choose a weight vector guess $\boldsymbol{\theta}_k \in \mathbf{\Omega}_{k-1}$ by Lemma 5.4.1;
- Restart and plan a robot trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by solving an optimal control problem with the cost function $J(\boldsymbol{\theta}_k)$ in (5.2) and the dynamics in (5.1);
- Robot executes the trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ while receving the human directional correction $\boldsymbol{a}_{t_k}$;
- Compute the coefficient matrices $\boldsymbol{H}_1(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ and $\boldsymbol{H}_2(\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T})$ based on (5.20), and generate the hyperplane and half space $\langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0$ by (5.6)-(5.7);
- Update the weight search space by $\mathbf{\Omega}_k = \mathbf{\Omega}_{k-1} \cap \{\boldsymbol{\theta} \in \mathbf{\Theta} \mid \langle \boldsymbol{h}_k, \boldsymbol{\theta} \rangle + b_k < 0\}$ by (5.22);

end

Output: $\boldsymbol{\theta}_K$.

5.5 Numerical Examples

In this section, we perform numerical simulations on an inverted pendulum and a two-link robot arm to validate the proposed algorithm and provide comparison with related work.

5.5.1 Inverted Pendulum

The dynamics of a pendulum is given in Appendix A.2. We discretize the continuous dynamics by the Euler method with a fixed time interval $\Delta = 0.2$s. The state and control vectors of the pendulum system are defined as $\boldsymbol{x} = [\alpha, \dot{\alpha}]'$ and $\boldsymbol{u} = u$, respectively, and the initial condition is $\boldsymbol{x}_0 = [0, 0]'$. In the cost function (5.2), we set the weight-feature running cost as

$$\boldsymbol{\phi} = [\alpha^2, \alpha, \dot{\alpha}^2, u^2]' \in \mathbb{R}^4, \tag{5.44a}$$

$$\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \theta_4]' \in \mathbb{R}^4, \tag{5.44b}$$

and set the final cost term as $h(\boldsymbol{x}_{T+1}) = 10(\alpha - \pi)^2 + 10\dot{\alpha}^2$, since our goal is to control the pendulum to reach the vertical position. The time horizon is set as $T = 30$.

In numerical examples, we generate "human's directional corrections" by simulation. Suppose that the expected weight vector is known explicitly: $\boldsymbol{\theta}^*$=$[0.5, 0.5, 0.5, 0.5]'$. Then, at iteration k, the "human's" directional corrections $\boldsymbol{a}_{t_k}$ is generated using the *sign* of the gradient of the true cost function $J(\boldsymbol{\theta}^*)$, that is,

$$\boldsymbol{a}_{t_k} = -\text{sign}\left(\left[\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)\right]_{t_k}\right) \in \mathbb{R}. \tag{5.45}$$

Here, $\left[\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)\right]_{t_k}$ denotes the t_kth entry of ∇J, and the correction time t_k is randomly chosen (evenly distributed) within horizon $[0, T]$. Obviously, the above "human's directional corrections" satisfies the assumption in (5.4).

The initial weight search space $\boldsymbol{\Omega}_0$ is set as

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \,|\, 0 \leq [\boldsymbol{\theta}]_i \leq 5,\ i = 1, 2, 3, 4\}. \tag{5.46}$$

In Algorithm 8, we set the termination parameter as $\epsilon = 10^{-1}$, and the maximum learning iteration solved by (5.39) is $K = 55$. We apply Algorithm 8 to learn the expected weight vector $\boldsymbol{\theta}^*$. To illustrate results, we define the guess error $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ (i.e., the distance square between the weight vector guess and the expected weight vector $\boldsymbol{\theta}^*$), and plot the

guess error $e_{\boldsymbol{\theta}}$ versus the number of iterations k in the top panel of Fig. 5.5. In the bottom panel of Fig. 5.5, we plot the directional correction $\boldsymbol{a}_{t_k}$ applied at each iteration k, where $+1$ and -1 bar denote positive and negative sign (direction) of the correction $\boldsymbol{a}_{t_k}$ in (5.45), respectively, and the number inside the bar denotes the correction time t_k that is randomly picked from $\{0, 1, \cdots, T\}$.

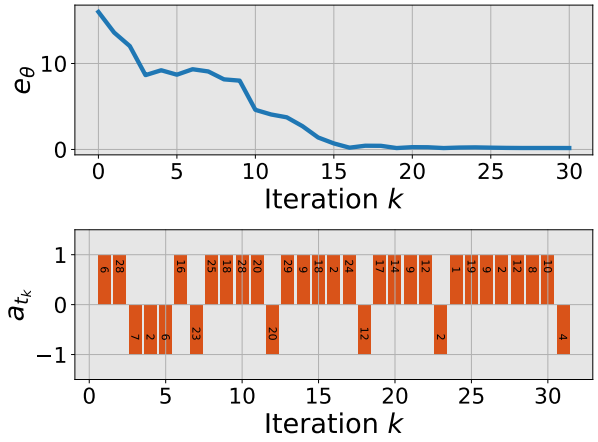


Figure 5.5. Learning a pendulum cost function from incremental directional corrections. The upper panel shows the guess error $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ versus iteration k, and the bottom panel shows the directional correction $\boldsymbol{a}_{t_k}$ (i.e., positive or negative) applied at each iteration k, and the value inside each bar is t_k that is randomly picked within the time horizon $[0, 30]$.

Based on the results in Fig. 5.5, we can see that as the learning iteration k increases, the weight vector guess $\boldsymbol{\theta}_k$ converges to the expected weight vector $\boldsymbol{\theta}^* = [0.5, 0.5, 0.5, 0.5]'$. This shows the validity of the method, as guaranteed by Theorem 5.4.1.

5.5.2 Two-link Robot Arm System

Here, we test the proposed method on a two-link robot arm. The dynamics of the robot arm (moving horizontally) is given in Appendix A.2. The state and control variables for the robot arm control system are defined as $\boldsymbol{x} = [\boldsymbol{q}, \dot{\boldsymbol{q}}]' \in \mathbb{R}^4$ and $\boldsymbol{u} = \boldsymbol{\tau} \in \mathbb{R}^2$, respectively. The initial condition of the robot arm is set as $\boldsymbol{x}_0 = [0, 0, 0, 0]'$. All parameters in the dynamics

are set as units. We discretize the continuous dynamics using the Euler method with a fixed time interval $\Delta = 0.2$s. In the cost function (5.2), we set the weight-feature cost as

$$\boldsymbol{\phi} = [q_1^2,\, q_1,\, q_2^2,\, q_2,\, \|\boldsymbol{u}\|^2]' \in \mathbb{R}^5, \tag{5.47a}$$

$$\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]' \in \mathbb{R}^5, \tag{5.47b}$$

and set the final cost $h(\boldsymbol{x}_{T+1}) = 100\Big((q_1 - \frac{\pi}{2})^2 + q_2^2 + \dot{q}_1^2 + \dot{q}_2^2\Big)$, as we aim to control the robot arm to finally reach and stop at the configuration of $\boldsymbol{q} = [\frac{\pi}{2}, 0]'$. The time horizon is set to be $T = 50$.

We still use simulation to generate "human's directional corrections", as similar to the previous experiment. Suppose that we explicitly know the expected weight vector $\boldsymbol{\theta}^* = [1, 1, 1, 1, 1]'$. Then, at each iteration k, the simulation generates a directional correction $\boldsymbol{a}_{t_k}$ by the sign of the gradient of the true cost function $J(\boldsymbol{\theta}^*)$, that is,

$$\boldsymbol{a}_{t_k} = -\text{sign}\left(\left[\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)\right]_{2t_k:2t_k+1}\right) \in \mathbb{R}^2, \tag{5.48}$$

where $[\nabla J(\boldsymbol{u}_{0:T}^{\boldsymbol{\theta}_k}, \boldsymbol{\theta}^*)]_{2t_k:2t_k+1}$ denotes the entries in ∇J at positions from $2t_k$ to $2t_k + 1$ (because the input dimension is $m = 2$), and the correction time t_k is randomly (in an even distribution) chosen from the time horizon $[0, T]$. Note that the sign operator is applied to its augment entry-wise.

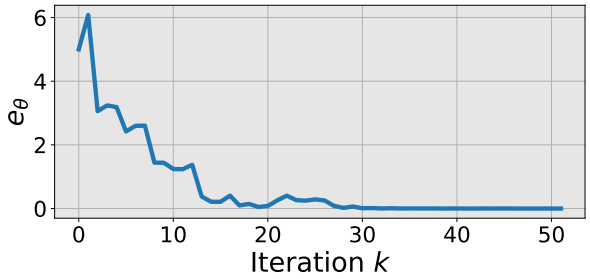


Figure 5.6. $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ versus iteration k in learning a robot-arm cost function from incremental directional corrections.

The initial weight search space $\boldsymbol{\Omega}_0$ is given by

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \mid 0 \leq [\boldsymbol{\theta}]_i \leq 4, \;\; i = 1, 2, \cdots, 5\}. \tag{5.49}$$

We set the termination parameter $\epsilon = 10^{-1}$, then the maximum learning iteration is $K = 83$ by (5.39). We apply Algorithm 8 to learn the expected weight vector $\boldsymbol{\theta}^* = [1, 1, 1, 1, 1]'$. To illustrate results, we define the guess error $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ and plot $e_{\boldsymbol{\theta}}$ versus iteration k in Fig. 5.6. We also plot the directional correction $\boldsymbol{a}_{t_k} = [a_{t_k,1}, a_{t_k,2}]'$ applied at each iteration k in Fig. 5.7, where the value inside the bar is the correction time t_k. Based on the results, we can see that as the iteration increases, the weight vector guess $\boldsymbol{\theta}_k$ converges to the expected weight vector $\boldsymbol{\theta}^*$, as guaranteed by Theorem 5.4.1. This result again demonstrates the effectiveness of the proposed method.

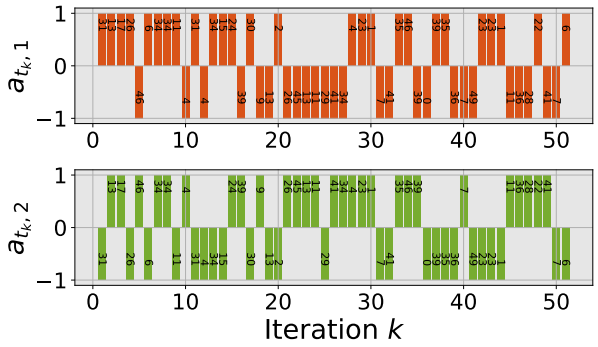


Figure 5.7. The directional correction $\boldsymbol{a}_{t_k} = [a_{t_k,1}, a_{t_k,2}]'$ applied at each iteration k during the learning of the robot-arm cost function. The number inside the bar is the correction time t_k randomly picked within the time horizon $[0, 50]$.

5.5.3 Comparison with Related Work

In this part, we compare the proposed method with two related work [38, 164] based on the inverted pendulum system. The dynamics settings and parameters follow the experiment in Section 5.5.1, and the weight-feature cost function is set as (5.44). According to [38], for each of the human's corrections, we first utilize the trajectory deformation technique [166]

to obtain the corresponding *human intended trajectory*. Specifically, given a correction $\boldsymbol{a}_k$, the human intended trajectory, denoted as $\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k} = \{\bar{\boldsymbol{x}}^{\boldsymbol{\theta}_k}_{0:T+1}, \bar{\boldsymbol{u}}^{\boldsymbol{\theta}_k}_{0:T}\}$, can be solved by

$$\bar{\boldsymbol{u}}^{\boldsymbol{\theta}_k}_{0:T} = \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T} + M^{-1}\bar{\boldsymbol{a}}_k, \tag{5.50}$$

where $\boldsymbol{\xi}_{\boldsymbol{\theta}_k} = \{\boldsymbol{x}^{\boldsymbol{\theta}_k}_{0:T+1}, \boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}\}$ is the robot's current trajectory; M is a matrix that smoothly propagates the local correction augmented vector in (5.5) along the rest of the trajectory [168]; and $\bar{\boldsymbol{x}}^{\boldsymbol{\theta}_k}_{0:T+1}$ in $\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k}$ is obtained from the robot dynamics in (5.1) given $\bar{\boldsymbol{u}}^{\boldsymbol{\theta}_k}_{0:T}$. For both [164] and [38], the learning update is

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha\Big(\boldsymbol{\phi}(\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k}) - \boldsymbol{\phi}(\boldsymbol{\xi}_{\boldsymbol{\theta}_k})\Big), \tag{5.51}$$

where $\boldsymbol{\phi}(\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k})$ and $\boldsymbol{\phi}(\boldsymbol{\xi}_{\boldsymbol{\theta}_k})$ are the vectors of feature values for the human intended trajectory $\bar{\boldsymbol{\xi}}_{\boldsymbol{\theta}_k}$ and the robot's original trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, respectively. In this comparison experiment, we set M in (5.50) as the finite differencing matrix [166], and α is 0.0006 (for having best performance). For all methods, the simulated human's corrections are the same, as shown in Fig. 5.8. To illustrate performance, we still use the guess error $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$.

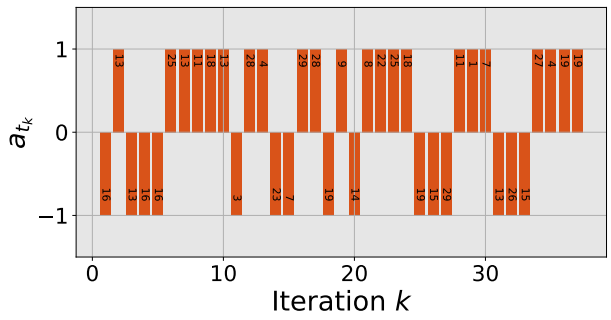


Figure 5.8. The correction $\boldsymbol{a}_{t_k}$ at each iteration k. The value labeled inside the bar is the randomly chosen correction time t_k.

We draw the guess error versus iterations for both [38, 164] and the proposed method in Fig. 5.9. By comparing both results, we can see a clear advantage of the proposed method over [164] [38] in terms of higher learning accuracy. As we have discussed in the related work, the learning update (5.51) used in [38, 164] only guarantees the convergence of the regret, which is defined as the averaged error of the cost values between the human intended

trajectory and the robot's trajectory under $J(\boldsymbol{\theta}^*)$ over the entire learning process. Thus, it does not directly lead to the convergence of $\boldsymbol{\theta}_k$ towards $\boldsymbol{\theta}^*$, as illustrated in Fig. 5.9. In contrast, Fig. 5.9 illustrates that using the proposed approach, the learned cost function quickly converges to $J(\boldsymbol{\theta}^*)$, as guaranteed by Theorem 5.4.1.

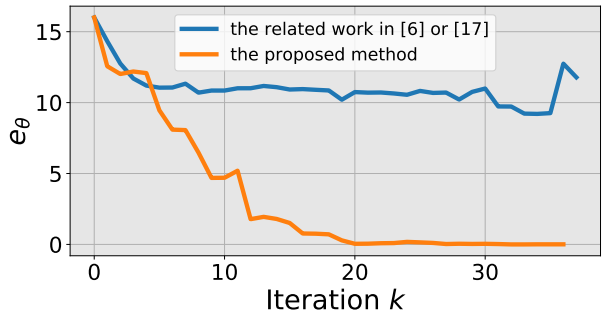


Figure 5.9. Comparison between [38, 164] and the proposed method. The figure shows $e_{\boldsymbol{\theta}} = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ v.s. iteration k.

Throughout the experiment, we find that the methods [38, 164] are not robust against the corrections $\boldsymbol{a}_{t_k}$ of very larger magnitude. Given a correction $\boldsymbol{a}_{t_k}$ with a larger magnitude, [164] and [38] are more likely to diverge especially. This is because larger correction magnitude $\|\boldsymbol{a}_{t_k}\|$ can be overshot (i.e., too large), which thus violates their assumption of improving the robot's motion, i.e., $J(\bar{\boldsymbol{u}}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*) < J(\boldsymbol{u}^{\boldsymbol{\theta}_k}_{0:T}, \boldsymbol{\theta}^*)$. We also find that the correction overshooting is more likely to happen at the end of learning process when the robot motion gets close to the optimal one, as we have explained in Section 5.2. This issue has also been illustrated in Fig. 5.9. At the iteration $k = 35$ in Fig. 5.9, we see that $e_{\boldsymbol{\theta}}$ instead becomes increased as we use a constant correction magnitude throughout the learning process. In contrast, the proposed method only leverages the direction of $\boldsymbol{a}_{t_k}$ (negative or positive) regardless of $\|\boldsymbol{a}_{t_k}\|$; thus there is no overshooting issue with the proposed method, This shows more flexibility of the proposed method than [38] and [164] in terms of choosing proper corrections.

5.6 Human-Robot Games

In this section, we develop two human-robot games, where a real human player online teaches a robot for motion planning through directional corrections. These games are used to validate the effectiveness of the proposed approach in practice. The games are developed on two robot systems/environments: a two-link robot arm and a 6-DoF quadrotor system, respectively. For each environment, a human player visually inspects robot's motion on a computer screen, meanwhile providing directional corrections through a keyboard. The goal of each game is to train a robot to learn an effective control objective function such that the robot successfully avoids obstacles and reaches a target position.

We have released all codes of the two games for the readers to have hands-on experience with. Please access at https://github.com/wanxinjin/Learning-from-Directional-Corrections.

5.6.1 Two-link Robot Arm Game

System Setup

In this game, the dynamics of a two-link robot arm and its parameters follow those in Section 5.5.2. The initial state of the arm is $\boldsymbol{x} = [-\frac{\pi}{2}, 0, 0, 0]'$, as shown in Fig. 5.10. For the parameterized cost function $J(\boldsymbol{\theta})$ in (5.2), we set the weight-feature running cost as

$$\boldsymbol{\phi} = [q_1^2,\, q_1,\, q_2^2,\, q_2,\, \|\boldsymbol{u}\|^2]' \in \mathbb{R}^5, \tag{5.52a}$$

$$\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5]' \in \mathbb{R}^5. \tag{5.52b}$$

Here, the weight-feature cost $\boldsymbol{\theta}'\boldsymbol{\phi}$ is a general second-order polynomial function. It is worth noting that in practice, if one has no prior knowledge/experience about how to choose good features, the general polynomial features are always a good choice. For the final cost $h(\boldsymbol{x}_{T+1})$ in $J(\boldsymbol{\theta})$ in (5.2), we set

$$h(\boldsymbol{x}_{T+1}) = 100\Big((q_1 - \frac{\pi}{2})^2 + q_2^2 + \dot{q}_1^2 + \dot{q}_2^2\Big), \tag{5.53}$$

because we aim the robot arm to finally reach and stop at the target vertical pose, i.e., $q_1^{\text{target}} = \frac{\pi}{2}$ and $q_2^{\text{target}} = 0$, as depicted in Fig. 5.10. It is also worth noting that in practice, the final cost function $h(\boldsymbol{x}_{T+1})$ in (5.2) can always be set as the distance to the target state. The time horizon in this robot arm game is set as $T = 50$ (that is, $50\Delta = 10$s).

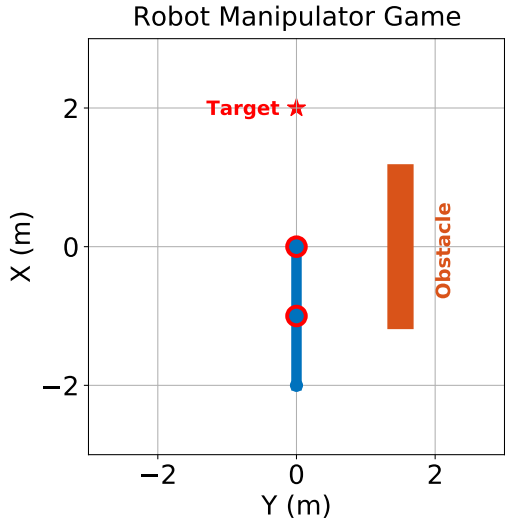


Figure 5.10. The robot arm game. The goal is to let a human player teach the robot arm to learn a valid cost function (i.e., the expected weight vector $\boldsymbol{\theta}^*$) by applying incremental directional corrections, such that it successfully moves from the initial condition (current pose) to the target (upward pose) while avoiding the obstacle.

Since we choose the polynomial features in (5.52a), different weight vector $\boldsymbol{\theta}$ leads to different robot's trajectories to the target pose. As shown in Fig. 5.10, we place an obstacle (colored in orange) in the workspace of the robot arm. Without human's intervention, the robot will move and crash into the obstacle. The goal of the game is to let a human player make directional corrections to the robot arm while it is moving, until the robot arm learns a valid cost function $J(\boldsymbol{\theta}^*)$ (i.e., the expected weight vector $\boldsymbol{\theta}^*$) to successfully avoid the obstacle and reach the target.

Human Correction Interface

For the above robot arm game, we use keyboards as the interface for a human player to provide directional corrections. We customize the (*up, down, left, right*) keys in a keyboard

and associate them with the directional corrections as listed in Table 5.1. During the game, at each iteration, a human player is allowed to press one or multiple keys from (*up, down, left, right*), and the keyboard interface is listening to which key(s) the human player hits and recording the time step of the keystroke(s). The recorded information, i.e., the pressed keys and stroke time, is translated into the directional correction $\boldsymbol{a}_{t_k}$ in the robot's input space according to Table 5.1. For example, at iteration k, while the robot arm is graphically executing the trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$, a human player hits the *up* and *left* keys simultaneously at the time step 10; then the corresponding correction information is translated into $\boldsymbol{a}_{t_k} = [1, 1]'$ with $t_k = 10$ according to Table 5.1.

Table 5.1. Correction interface for the robot arm game.

Keys	Directional correction	Interpretation of correction
up	$\boldsymbol{a} = [1, 0]$	add counter-close-wise torque to Joint 1
down	$\boldsymbol{a} = [-1, 0]$	add close-wise torque to Joint 1
left	$\boldsymbol{a} = [0, 1]$	add counter-close-wise torque to Joint 2
right	$\boldsymbol{a} = [0, -1]$	add close-wise torque to Joint 2

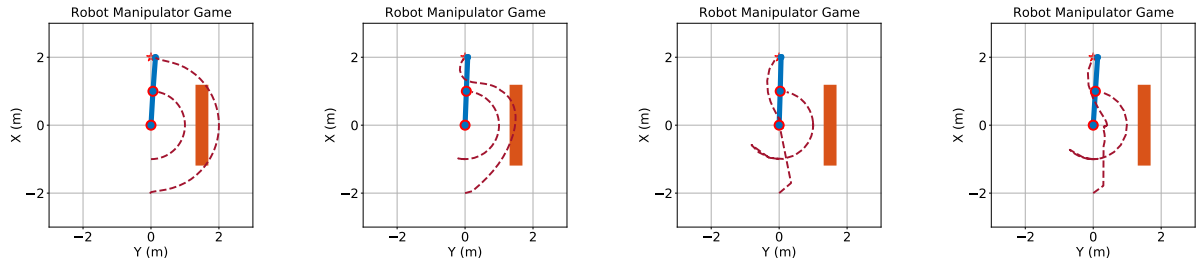
Game Procedure

The procedure of this robot arm game is as follows. By default, the robot's initial weight search space $\boldsymbol{\Omega}_0$ is set as

$$\boldsymbol{\Omega}_0 = \{\boldsymbol{\theta} \,|\, \theta_1, \theta_3 \in [0, 1], \theta_2, \theta_4 \in [-3, 3], \theta_5 \in [0, 0.5]\}. \tag{5.54}$$

As stipulated by the main algorithm, at each iteration k, the robot arm first chooses a weight vector guess $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$ by Lemma 5.4.1 and then plans the corresponding motion trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ (by minimizing the cost function $J(\boldsymbol{\theta}_k)$ subject to the robot's dynamics). While the robot arm is graphically executing the planned trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ on the computer screen, a human player inspects the ongoing motion $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ and provides the directional correction $\boldsymbol{a}_{t_k}$ via the keyboard interface based on the rules in Table 5.1. Each time the keyboard interface detects the humans player's directional correction $\boldsymbol{a}_{t_k}$, the robot arm incorporates such correction

$\boldsymbol{a}_{t_k}$ to update the weight search space from $\boldsymbol{\Omega}_{k-1}$ to $\boldsymbol{\Omega}_k$ by Step 3 in the main algorithm. This planning-correction-update procedure repeats until the robot arm successfully avoids the obstacle and reaches the target and then the human player will not intervene the robot arm any more—mission accomplished.



(a) Iteration $k = 1$ (b) Iteration $k = 2$ (c) Iteration $k = 3$ (d) Iteration $k = 4$

Figure 5.11. An illustrative result for the robot arm game. The goal of this game is to let a human player to correct the robot arm while it is acting, until the robot arm learns a valid control objective function (i.e., expected $\boldsymbol{\theta}^*$) for successfully avoiding the obstacle and reaching the target pose. Corresponding to the above sub-figures, the robot's weight vector guess $\boldsymbol{\theta}_k$ and the player's directional correction $\boldsymbol{a}_k$ at each iteration k are listed in Table 5.2. In (a), and the robot arm randomly chooses an initial weight vector guess $\boldsymbol{\theta}_1 \in \boldsymbol{\Omega}_0$ and its resulting motion crashes into the obstacle. In (d), the robot arm successfully learns a valid cost function (i.e., the expected $\boldsymbol{\theta}^*$) to avoid the obstacle and reach the target—mission accomplished.

Table 5.2. An illustrative result for the robot arm game.

Iteration k	Robot's current weight vector guess $\boldsymbol{\theta}_k$	A human player's directional correction $\boldsymbol{a}_{t_k}$ and correction time t_k
$k = 1$	$\boldsymbol{\theta}_k = [0.50, 0.00, 0.50, -0.00, 0.25]'$	$\boldsymbol{a}_{t_k} = [0, 1]$ (i.e., *left* key pressed) and $t_k = 11$
$k = 2$	$\boldsymbol{\theta}_k = [0.50, 0.00, 0.50, -1.50, 0.25]'$	$\boldsymbol{a}_{t_k} = [0, 1]$ (i.e., *left* key pressed) and $t_k = 16$
$k = 3$	$\boldsymbol{\theta}_k = [0.50, 0.00, 0.34, -2.03, 0.25]'$	$\boldsymbol{a}_{t_k} = [-1, 0]$ (i.e., *down* key pressed) and $t_k = 34$
$k = 4$	$\boldsymbol{\theta}_k = [0.50, 1.48, 0.36, -2.00, 0.25]'$	*Mission accomplished!* $\boldsymbol{\theta}^* = \boldsymbol{\theta}_k$

Results and Analysis

We present an illustrative result in Fig. 5.11, where we show that the robot arm can learn a valid cost function after only four rounds of human's directional corrections (i.e., four iterations). At each iteration k, the robot's current weight vector guess $\boldsymbol{\theta}_k$ and the

player's correction $\boldsymbol{a}_{t_k}$ are in Table 5.2. From Table 5.2 and Fig. 5.11, we observe that the robot arm has successfully learned an effective cost function to avoid the obstacle and reach the target after four rounds of human's directional corrections. These results indicate that the effectiveness of the proposed method for real human users to train a robot through directional corrections. We also have the following comments.

(i) Throughout the game, we find that the successful teaching of the robot *does not require a human player to have prior experience or much practice with the game*. The result shown in Table 5.2 and Fig. 5.11 is just one of many examples, and a novice player can readily provide another sequence of corrections, such as using different combinations of keys and corrections times, to successfully train the robot arm within few iterations.

(ii) We emphasize that the human's corrections for the robot arm is very intuitive. For example, in Fig. 5.11a, at first iteration, as we see that the robot arm is crashing into the obstacle, the human correction could be a counter-clock-wise torque applied to joint 2 in order to make the second link bend inward. Thus, we need to press the *left* key according to Table 5.1. For another example, in Fig. 5.11c, since the first joint is shown moving too fast in counter-clock-wise, the human correction needs to give a clock-wise toque to the first joint in order to make it slow down, and thus the player needs to press the *down* key according to Table 5.1. To gain a better understanding of the method and the game, we encourage the reader to download the game codes and have hand-on experience with the robot arm game.

5.6.2 6-DoF Quadrotor Maneuvering Game

System Setup

The dynamics of a quadrotor drone flying in SE(3) (i.e., full position and attitude) space is given in Appendix A.3. We define the state vector for the quadrotor as

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{r}_I & \boldsymbol{v}_I & \boldsymbol{q}_{B/I} & \boldsymbol{\omega}_B \end{bmatrix} \in \mathbb{R}^{13}, \tag{5.55}$$

and define the control input vector as

$$\boldsymbol{u} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 \end{bmatrix}' \in \mathbb{R}^4. \tag{5.56}$$

We discretize the above continuous dynamics of the quadrotor by the Euler method with a fixed time interval $\Delta = 0.1$s. To achieve SE(3) control for a quadrotor, we need to carefully design the attitude error. As in [157], we define the attitude error between the quadrotor's attitude $\boldsymbol{q}$ and a target attitude $\boldsymbol{q}^{\text{target}}$ as

$$e(\boldsymbol{q}, \boldsymbol{q}_{\text{target}}) = \frac{1}{2}\text{trace}(I - R'(\boldsymbol{q}^{\text{target}})R(\boldsymbol{q})), \tag{5.57}$$

where $R(\boldsymbol{q}) \in \mathbb{R}^{3\times 3}$ is the direction cosine matrix corresponding to $\boldsymbol{q}$ (see [158] for more details).

In the cost function in (5.2), we set the final cost $h(\boldsymbol{x}_{T+1})$ as

$$h(\boldsymbol{x}_{T+1}) = \|\boldsymbol{r}_I - \boldsymbol{r}_I^{\text{target}}\|^2 + 10\|\boldsymbol{v}_I\|^2 + 100e(\boldsymbol{q}_{B/I}, \boldsymbol{q}_{B/I}^{\text{target}}) + 10\|\boldsymbol{w}_B\|^2, \tag{5.58}$$

because we always want the quadrotor to finally land on a target position given by $\boldsymbol{r}_I^{\text{target}}$ in a target attitude $\boldsymbol{q}_{B/I}^{\text{target}}$. Here, $\boldsymbol{r}_I = [r_x, r_y, r_z]'$ is the position of the quadrotor expressed in the world frame. We set the weight-feature cost in (5.2) as

$$\boldsymbol{\phi} = \begin{bmatrix} r_x^2 & r_x & r_y^2 & r_y & r_z^2 & r_z & \|\boldsymbol{u}\|^2 \end{bmatrix}' \in \mathbb{R}^7, \tag{5.59a}$$

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 & \theta_5 & \theta_6 & \theta_7 \end{bmatrix}' \in \mathbb{R}^7. \tag{5.59b}$$

Here, feature vector $\boldsymbol{\phi}$ consists of general polynomial features and different weight vectors $\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7]'$ will determine how the quadrotor reaches the target (that is, the specific path of the quadrotor). It is again worth noting that in practical situations, if one has no prior knowledge/experience about how to choose good features, general polynomial features are always a good choice, as in (5.59a).

As shown in Fig. 5.12, the goal of this quadrotor game is to let a human player teach the quadrotor to fly from the initial position $\boldsymbol{r}_I(0) = [-8, -8, 5]'$ (bottom left), passing through a gate (colored in brown), and finally land on a specified target position $\boldsymbol{r}_I^{\text{target}} = [8, 8, 0]'$ (upper

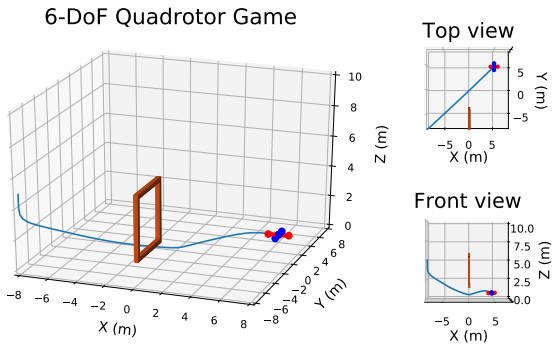


Figure 5.12. The 6-DoF quadrotor game. The goal of this game is to let a human player to teach a 6-DoF quadrotor system to learn a valid control cost function (i.e., the expected weight vector $\boldsymbol{\theta}^*$) by providing directional corrections, such that it can successfully fly from the initial position (in bottom left), pass through a gate (colored in brown), and finally land on the specified target (in upper right).

right) with the target attitude $\boldsymbol{q}_{B/I}^{\text{target}} = [1, 0, 0, 0]'$. The initial attitude of the quadrotor is $\boldsymbol{q}_{B/I}(0) = [1, 0, 0, 0]'$ and initial velocity quantities are zeros. The game time horizon is set as $T = 50$, that is, $T\Delta = 5$s.

Human Correction Interface

In the 6-DoF quadrotor game, we use keyboards as the interface for a human player to provide directional corrections. Specifically, we use the ('*up*', '*down*', '*w*', '*s*', '*a*', '*d*') keys and associate them with specific directional correction signals, as listed in Table 5.3. During the game (i.e., algorithm progress), a human player is allowed to press one or multiple combinations of the keys in Table 5.3. The interface is listening to the keystrokes from the human player, and once detected, the keystrokes are translated into the directional corrections according to Table 5.3. Together with the pressed keys, the time step at which a key is hit is also recorded, as the correction time t_k. For example, suppose that while the computer screen is graphically playing the quadrotor executing the trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ (at iteration k), the human player presses '*s*' key at the time step 5; then, according to Table 5.3, the translated human correction will be $\boldsymbol{a}_{t_k} = [0, -1, 0, 1]'$ with the correction time $t_k = 5$.

Table 5.3. Correction interface for 6-DoF quadrotor game.

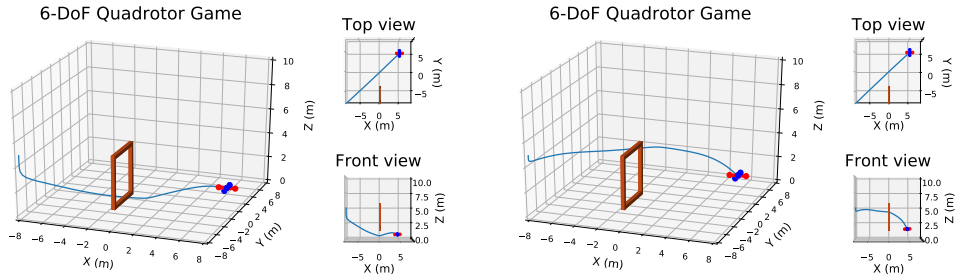
Keys	Directional correction	Interpretation of correction
'up'	T_1=1, T_2=1, T_3=1, T_4=1	Upward force applied at COM
'down'	T_1= − 1, T_2= − 1, T_3= − 1, T_4= − 1	Downward force applied at COM
'w'	T_1=0, T_2=1 T_3=0, T_4= −1	Negative torque along body-axis x
's'	T_1=0, T_2= − 1 T_3=0, T_4=1	Positive torque along body-axis x
'a'	T_1=1, T_2=0 T_3= − 1, T_4=0	Negative torque along body-axis y
'd'	T_1= − 1, T_2=0 T_3=1, T_4=0	Positive torque along body-axis y

Game Procedure

The procedure of playing the 6-DoF quadrotor game is as follows. By default, the initial weight research space $\boldsymbol{\Omega}_0$ is set as

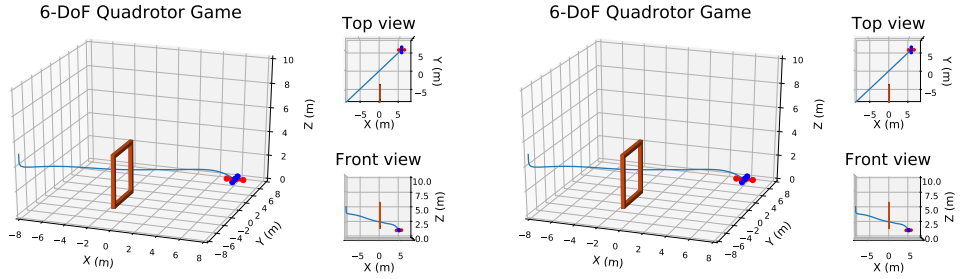
$$\boldsymbol{\Omega}_0=\{\boldsymbol{\theta} \,|\, \theta_1, \theta_3, \theta_5 \in [0,1], \theta_2, \theta_4, \theta_6 \in [-8,8], \theta_7 \in [0,0.5]\}. \tag{5.60}$$

As stated by the main algorithm, at each iteration k, the quadrotor chooses a weight vector guess $\boldsymbol{\theta}_k \in \boldsymbol{\Omega}_{k-1}$ by Lemma 5.4.1 and plans a motion trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ by minimizing the cost function $J(\boldsymbol{\theta}_k)$ subject to the dynamics constraint in (A.3). While the quadrotor is graphically executing $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ on the computer screen, the keyboard interface is listening to player's directional corrections. Once detecting a player's correction $\boldsymbol{a}_{t_k}$, the quadrotor uses such correction to update the weight search space from $\boldsymbol{\Omega}_{k-1}$ to $\boldsymbol{\Omega}_k$ following Step 3 in the main algorithm. This planning-correction-update procedure repeats until the quadrotor successfully flies through the gate and lands on the target position, and then the human player will not intervene the quadrotor any more—mission accomplished.



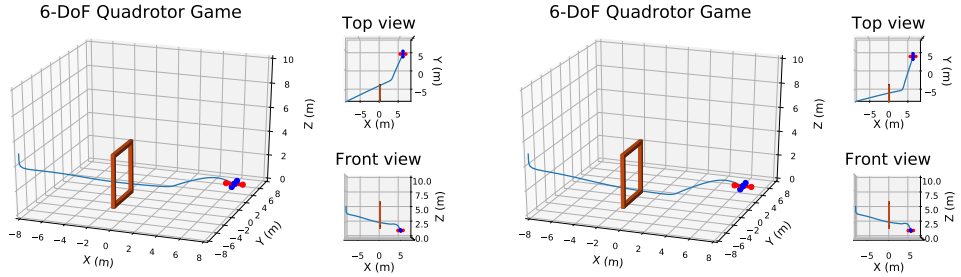
(a) Iteration $k = 1$

(b) Iteration $k = 2$



(c) Iteration $k = 3$

(d) Iteration $k = 4$



(e) Iteration $k = 5$

(f) Iteration $k = 6$

Figure 5.13. An illustrative result for the 6-DoF quadrotor game. The goal of this game is to let a human player, through providing directional corrections, to teach a 6-DoF quadrotor to learn a valid control cost function (i.e., expected $\boldsymbol{\theta}^*$) for successfully flying from the initial position, passing through a gate, and finally landing on the target position. Corresponding to each iteration k in the above sub-figures, we also list the robot's current weight vector guess $\boldsymbol{\theta}_k$ and the human player's directional correction $\boldsymbol{a}_k$ in Table 5.4. In (a), at iteration $k = 1$, the quadrotor chooses an initial weight vector guess $\boldsymbol{\theta}_1 \in \boldsymbol{\Omega}_0$. In (c), at iteration $k = 3$, since the human player does not provide any correction, the quadrotor's trajectory at this iteration is the same with the one in (d) (iteration $k = 4$). In (f), at iteration $k = 6$, the quadrotor successfully flies through the gate and lands on the target position, which means that a valid quadrotor cost function is successfully learned—mission accomplished.

Table 5.4. An illustrative result for the 6-DoF quadrotor game.

Iteration k	quadrotor's current weights guess $\boldsymbol{\theta}_k$	A human player's directional corrections $\boldsymbol{a}_{t_k}$ and correction time steps t_k
$k=1$	$\boldsymbol{\theta}_k = [0.50,\ 0.00,\ 0.50, -0.00,\ 0.50, -0.00,\ 0.25]'$	$\boldsymbol{a}_{t_k} = [1,1,1,1]$ (i.e., *up* key pressed) and $t_k = 8$ $\boldsymbol{a}_{t_k} = [1,1,1,1]$ (i.e., *up* key pressed) and $t_k = 20$
$k=2$	$\boldsymbol{\theta}_k = [0.50, -0.00,\ 0.50, -0.00,\ 0.50, -3.99,\ 0.25]'$	$\boldsymbol{a}_{t_k} = [-1,-1,-1,-1]$ (i.e., *down* key pressed) and $t_k = 14$
$k=3$	$\boldsymbol{\theta}_k = [0.50, -1.70,\ 0.50, -1.70,\ 0.52, -1.89,\ 0.25]'$	*No correction provided (because the human player hesitated)*
$k=4$	$\boldsymbol{\theta}_k = [0.50, -1.70,\ 0.50, -1.70,\ 0.52, -1.89,\ 0.25]'$	$\boldsymbol{a}_{t_k} = [0,-1,0,1]$ (i.e., *'s'* key pressed) and $t_k = 13$
$k=5$	$\boldsymbol{\theta}_k = [0.50, -2.76,\ 0.50, -2.45,\ 0.60, -2.22,\ 0.25]'$	$\boldsymbol{a}_{t_k} = [0,-1,0,1]$ (i.e., *'s'* key pressed) and $t_k = 19$
$k=6$	$\boldsymbol{\theta}_k = [0.50, -3.11,\ 0.50,\ 4.89,\ 0.65 - 2.67,\ 0.25]'$	*Mission accomplished!* $\boldsymbol{\theta}^* = \boldsymbol{\theta}_k$

Results and Analysis

We present one illustrative result in Fig. 5.13 and Table 5.4. Fig. 5.13 illustrates the execution of the quadrotor's trajectory $\boldsymbol{\xi}_{\boldsymbol{\theta}_k}$ at different iterations k. Table 5.4 presents the weight vector guess $\boldsymbol{\theta}_k$ and the human player's correction $\boldsymbol{a}_{t_k}$ at each iteration k. The results show that within five rounds of directional corrections (i.e., five iterations), the quadrotor is able to learn a valid cost function to successfully fly through the given gate and land on the target landing position. The results again demonstrate the efficiency of the proposed method for a human user to train a robot through directional corrections. Also, we have the following comments.

(i) It is worth mentioning that the successful teaching of the quadrotor in the above game *does not require a human player to have prior experience or much practice with the game*. The above result is just one of the many results, and a new player can choose another sequence of corrections and quickly teach the quadrotor to learn a valid cost function for accomplishing the task.

(ii) The choice of the directional corrections is very intuitive and straightforward. For example, as in Fig. 5.13a, the quadrotor is flying too low, the player thus has pressed *up* key (Table 5.4) to let the quadrotor fly higher according to Table 5.3. Also, as in Fig. 5.13d the quadrotor is flying too left relative to the gate, the player thus have pressed *'s'* key (i.e., a positive torque along the quadrotor's body x-axis) to let the quadrotor tilt right towards the gate. We encourage the reader to have hands-on experience with the above quadrotor game.

5.7 Other Choices of Search Space Center

For the search space $\boldsymbol{\Omega}_k \subset \mathbb{R}^r$, we choose $\boldsymbol{\theta}_k$ as the center of Maximum Volume Ellipsoid (MVE) inscribe $\boldsymbol{\Omega}_k$. Other choices for $\boldsymbol{\theta}_k$ could be the center of gravity [173], the Chebyshev center [174], the analytic center [175], etc.

Center of Gravity. The center of gravity for a polytope $\boldsymbol{\Omega}$ is defined as

$$\boldsymbol{\theta}_{\text{cg}} = \frac{\int_{\boldsymbol{\Omega}} \boldsymbol{\theta} d\boldsymbol{\theta}}{\int_{\boldsymbol{\Omega}} d\boldsymbol{\theta}}. \tag{5.61}$$

As proven by [176], the volume reduction rate using the center of gravity is

$$\frac{\textbf{Vol}(\boldsymbol{\Omega}_{k+1})}{\textbf{Vol}(\boldsymbol{\Omega}_k)} \leq 1 - \frac{1}{e} \approx 0.63, \tag{5.62}$$

which may lead to faster convergence than the rate $(1 - 1/r)$ using the center of MVE in Section 5.4 when the parameter space is high dimensional (i.e., r large). However, for a polytope described by a set of linear inequalities, it requires much higher computational cost to obtain the center of gravity in (5.61) than solving the center of MVE inscribed in $\boldsymbol{\Omega}_k$ [170].

Chebyshev Center. Chebyshev center is defined as the center of the largest Euclidean ball that lies inside the polytope $\boldsymbol{\Omega}$. The Chebyshev center for a polytope can be computed by solving a linear program [41], which is also efficient. But the Chebyshev center is not affinely invariant to the transformations of coordinates [170]. Therefore, a linear mapping of features may lead to an inconsistent weight vector estimation.

Analytic Center. Given a polytope $\boldsymbol{\Omega} = \{\boldsymbol{\theta} \,|\, \langle \boldsymbol{h}_i, \boldsymbol{\theta} \rangle + b_i < 0,\ i = 1, \cdots, m\}$, the analytic center is defined as

$$\boldsymbol{\theta}_{\text{ac}} = \min_{\boldsymbol{\theta}} -\sum_{i=1}^{m} \log(b_i - \boldsymbol{h}_i' \boldsymbol{\theta}). \tag{5.63}$$

As shown by [177, 178], using the analytic center achieves a good trade-off in terms of simplicity and practical performance, which however does not easily lead to the volume reduction analysis in Lemma 5.4.2 as choosing the center of MVE.

5.8 Conclusions

In this chapter, we have proposed an approach which enables a robot to learn an objective function incrementally from human user's directional corrections. The directional corrections, applied by a human user at certain time steps during the robot's motion, can be any input correction as long as it points in a direction of improving the robot's current motion under an implicit objective function. The proposed learning method is based on the cutting plane technique, which only utilizes the direction of a correction to update the objective function guess. We establish the theoretical results to show the convergence of the learned objective function towards the implicit one. We demonstrate the effectiveness of the method using numerical simulations and two human-robot games. The results show the proposed method outperforms the state of the art, and that it enables a non-expert human user to teach a robot to learn an effective control objective function for satisfactory motion with few directional corrections.

PART III

SUMMARY AND FUTURE DIRECTIONS

6. SUMMARY AND FUTURE DIRECTIONS

6.1 Summary

This thesis considers learning and control in autonomous robots. Towards efficient autonomy, the first portion of this thesis proposes to embed optimal control theory into learning paradigms, which lead to a series of new control-induced learning methods. The contributions of this potion lie in two directions.

Contribution 1: New methods for Inverse Optimal Control. (I) We have developed the IOC method for learning objective function from incomplete trajectory observations. First, we show the relationship between any segment data of trajectory and the unknown objective function (parameter); second, we show the iterative property of IOC and analyze how additional data points and features are incorporated in the IOC process; and third, we give necessary conditions for the minimal observation length required for IOC. The proposed method enables to learn an objective function incrementally by finding the minimal required observations. (II) We have developed a multi-phase IOC method for learning the phase-dependent objective functions. For a trajectory resulting from optimizing multi-phase objective functions, the method not only recovers the underlying control objective function for each motion phase, but also estimates the time transition of each phase (i.e., at which time step the objective function has changed). (III) We have developed a distributed IOC algorithm which enables to learn an objective function with both data and computation distributed. (IV) We have provided some novel applications of the above new methods, including human motion prediction and segmentation.

Contribution 2: Pontryagin Differentiable Programming Methodology. Our contribution to both machine learning and control fields is an innovative methodology, named as Pontryagin differentiable programming (PDP), which provides an end-to-end framework that is able to solve a broad range of learning and control takes, including inverse reinforcement learning, system identification, policy optimization, and planning, etc. This PDP framework has the following new features. First, we unify the problems of learning objective functions, dynamics models, and control policies within the same formulation based on optimal control models, where different tasks differ only in different unknown aspects (param-

eterized models) and different loss function. Second, we differentiate through Pontryagin's Maximum Principle, and this allows to obtain the analytical derivative of a trajectory with respect to tunable parameters in an optimal control system, enabling end-to-end learning of dynamics, policies, or/and control objective functions; and third, we propose an auxiliary control system in backward pass of the PDP framework, and the output of this auxiliary control system is the analytical derivative of the original system's trajectory with respect to the parameters, which can be iteratively solved using standard control tools.

The second portion of this thesis focuses on the innovative robot learning paradigm with human guidance on the loop. We seek to answer the question of how to take the advantage of the human guidance to boost the efficiency in robot learning, while maintaining the burden of human providing guidance as low as possible? We have two contributions.

Contribution 3: Learning from Sparse Demonstrations. We have presented an approach which enables a robot to learn an objective function from sparse demonstrations of an expert. The demonstrations are given by a small number of sparse waypoints; the waypoints are desired outputs of the robot's trajectory at certain time instances, sparsely located within a demonstration time horizon. The duration of the expert's demonstration may be different from the actual duration of the robot's execution. The proposed method is able to jointly learn an objective function and a time-warping function such that the robot's reproduced trajectory has minimal distance to the sparse demonstration waypoints. Unlike existing inverse reinforcement learning techniques, the proposed approach, based on Pontryagin Differentiable Programming, directly minimizes of the distance between the robot's trajectory and the sparse demonstration waypoints and allows simultaneous learning of an objective function and a time-warping function.

Contribution 4: Learning from Directional Corrections. We have developed a new learning scheme that enables a robot to learn a control objective function incrementally from human user's corrections. The human's corrections can be as simple as directional corrections—corrections that indicate the direction of a control change without indicating its magnitude—applied at some time instances during the robot's motion. We only assume that each of the human's corrections, regardless of its magnitude, points in a direction that

improves the robot's current motion relative to an implicit objective function. The proposed method uses the direction of a correction to update the estimate of the objective function based on the cutting plane technique. We establish the theoretical results to show that this process of incremental correction and update guarantees convergence of the learned objective function to the implicit one.

6.2 Future Directions

The following will provide a brief outlook of future research directions at the intersection of control, learning, and optimization.

Topic 1: Control Perspective on Deep Learning

One important research trend in deep learning community is to achieve the explainability of artificial intelligence. Our belief is that control theory can contribute to this goal by providing abundant modeling or analysis tools. Representative examples include the recent attempt to leverage optimal control theory to explain deep neural network [179], which leads to new insights of deep learning and new training algorithms, and use of Koopman theory to facilitate the training of the deep neural networks [180]. It is fair to say that a wide range of learning tasks can find their counterparts problem in control fields, as we have discussed in [35]. Thus, control theory is expected to provide a fundamental understanding of deep learning and potentially bring out new results and learning algorithms.

Topic 2: End-to-End Differentiable Control

Modern control theory provides standardized approaches for control analysis, design, and optimizations. However, due to their mathematical complexity, model control theories seem largely incomprehensible to engineers who are not skilled in the art. Furthermore, many control methods require practitioners' high expertise and experience for model selection and specification (such as finding Lyapunov functions, shaping cost functions). All these difficulties motive a question: can we automate the process of control modeling/design/analysis through machine learning by taking advantage of history data (e.g., design data, experi-

mental data, and demonstration data)? The answer to this question can potentially make modern control approaches more accessible to the general public. In the previous Pontryagin Differentiable Programming work, we have made initial progress towards the end-to-end differentiable control. We have borrowed the concept of *end-to-end learning* and developed a 'supervised' way to train models by directly optimizing a design index with respect to the control model of interest. Such a prototype provides a promising set of future directions.

Topic 3: Safe/Certificated Learning and Control

Deploying a learning agent from simulation to real word will potentially lead to safety or robustness concerns. This can be caused by 1) discrepancy between computational models and real-word environments, 2) inherent uncertainty of environments and training process, 3) excessive search strategies, and 4) exogenous disturbances to and malicious attacks in observation data. These challenges are particularly urgent in human-involved tasks. On the other hand, robustness and safety are at the core of control research, where many theories and tools (such as invariant set, reachable set, certificate functions, set membership approaches, robust control, differential game theories) are available. Hence, in the future, it is worth exploring how to leverage these control tools to develop new theoretical foundations for safe learning [181, 182].

Topic 4: Learning with Humans on the Loop

Our previous research focuses on the methodologies that utilize minimal human guidance for efficient robot learning. But such techniques still emphasize the dominance of human role in robot learning, which thus may cause burden on the human user. The future research could consider how to leverage the active role of a robot in its learning progress or how do develop a more natural and efficient human-robot interaction scheme that enables a smooth switch of dominance between human and robot control. One interesting question in these topics is how to define and evaluate the robot's confidence in a learning/control task such that the human guidance is given only on request.

REFERENCES

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Dimitri P Bertsekas. *Dynamic programming and optimal control*. Vol. 1. 2. Athena scientific Belmont, MA, 1995.

[3] Andrew Y Ng, Stuart J Russell, et al. "Algorithms for inverse reinforcement learning." In: *International Conference on Machine Learning*. Vol. 1. 2000, p. 2.

[4] Rudolf Emil Kalman. "When is a linear control system optimal?" In: *Journal of Basic Engineering* 86.1 (1964), 51–60.

[5] Marc Deisenroth and Carl E Rasmussen. "PILCO: A model-based and data-efficient approach to policy search". In: *International Conference on Machine Learning*. 2011, pp. 465–472.

[6] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. "When to trust your model: Model-based policy optimization". In: *Advances in Neural Information Processing Systems*. 2019, pp. 12519–12530.

[7] Weiwei Li and Emanuel Todorov. "Iterative linear quadratic regulator design for nonlinear biological movement systems". In: *International Conference on Informatics in Control, Automation and Robotics*. 2004, pp. 222–229.

[8] Jacques Vlassenbroeck and Rene Van Dooren. "A Chebyshev technique for solving nonlinear optimal control problems". In: *IEEE transactions on automatic control* 33.4 (1988), pp. 333–340.

[9] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural ordinary differential equations". In: *Advances in Neural Information Processing Systems*. 2018, pp. 6571–6583.

[10] Rolf Johansson. *System modeling and identification*. Prentice Hall, 1993.

[11] Wanxin Jin, Dana Kulić, Jonathan Feng-Shun Lin, Shaoshuai Mou, and Sandra Hirche. "Inverse optimal control for multiphase cost functions". In: *IEEE Transactions on Robotics* 35.6 (2019), pp. 1387–1398.

[12] Wanxin Jin, Dana Kulić, Shaoshuai Mou, and Sandra Hirche. "Inverse optimal control from incomplete trajectory observations". In: *The International Journal of Robotics Research* 40.6-7 (2021), pp. 848–865.

[13] Wanxin Jin, Zihao Liang, and Shaoshuai Mou. “Inverse Optimal Control from Demonstration Segments”. In: *arXiv preprint arXiv:2010.15034* (2020).

[14] Wanxin Jin and Shaoshuai Mou. “Distributed inverse optimal control”. In: *Automatica* 129 (2021), p. 109658.

[15] Dimitri P Bertsekas et al. *Dynamic programming and optimal control: Vol. 1*. Athena scientific Belmont, 2000.

[16] Emanuel Todorov. “Optimality principles in sensorimotor control”. In: *Nature neuroscience* 7.9 (2004), pp. 907–915.

[17] CK Chow and DH Jacobson. “Studies of human locomotion via optimal programming”. In: *Mathematical Biosciences* 10.3-4 (1971), pp. 239–306.

[18] Gergely Csibra and György Gergely. “‘Obsessed with goals’: Functions and mechanisms of teleological interpretation of actions in humans”. In: *Acta psychologica* 124.1 (2007), pp. 60–78.

[19] Anca D Dragan, Kenton CT Lee, and Siddhartha S Srinivasa. “Legibility and predictability of robot motion”. In: *ACM/IEEE International Conference on Human-Robot Interaction*. IEEE. 2013, pp. 301–308.

[20] Debora Clever, R Malin Schemschat, Martin L Felis, and Katja Mombaur. “Inverse optimal control based identification of optimality criteria in whole-body human walking on level ground”. In: *IEEE International Conference on Biomedical Robotics and Biomechatronics*. 2016, pp. 1192–1199.

[21] Taesung Park and Sergey Levine. “Inverse optimal control for humanoid locomotion”. In: *Robotics: Science and Systems*. 2013.

[22] Katja Mombaur, Anh Truong, and Jean-Paul Laumond. “From human to humanoid locomotion—an inverse optimal control approach”. In: *Autonomous Robots* 28.3 (2010), pp. 369–383.

[23] Jim Mainprice, Rafi Hayne, and Dmitry Berenson. “Predicting human reaching motion in collaborative tasks using inverse optimal control and iterative re-planning”. In: *IEEE International Conference on Robotics and Automation*. 2015, pp. 885–892.

[24] Pieter Abbeel, Adam Coates, and Andrew Y Ng. “Autonomous helicopter aerobatics through apprenticeship learning”. In: *The International Journal of Robotics Research* 29.13 (2010), pp. 1608–1639.

[25] J Zico Kolter, Pieter Abbeel, and Andrew Y Ng. "Hierarchical apprenticeship learning with application to quadruped locomotion". In: *Advances in Neural Information Processing Systems*. 2008, pp. 769–776.

[26] Markus Kuderer, Shilpa Gulati, and Wolfram Burgard. "Learning driving styles for autonomous vehicles from demonstration". In: *IEEE International Conference on Robotics and Automation*. IEEE. 2015, pp. 2641–2646.

[27] Jim Mainprice, Rafi Hayne, and Dmitry Berenson. "Goal set inverse optimal control and iterative replanning for predicting human reaching motions in shared workspaces". In: *IEEE Transactions on Robotics* 32.4 (2016), pp. 897–908.

[28] Jonathan Feng-Shun Lin, Vincent Bonnet, Adina M Panchea, Nacim Ramdani, Gentiane Venture, and Dana Kulić. "Human motion segmentation using cost weights recovered from inverse optimal control". In: *IEEE-RAS International Conference on Humanoid Robots*. 2016, pp. 1107–1113.

[29] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. "Maximum Entropy Inverse Reinforcement Learning." In: *AAAI Conference on Artificial intelligence*. 2008, pp. 1433–1438.

[30] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. "Maximum entropy deep inverse reinforcement learning". In: *arXiv preprint arXiv:1507.04888* (2015).

[31] Neha Das, Sarah Bechtle, Todor Davchev, Dinesh Jayaraman, Akshara Rai, and Franziska Meier. "Model-Based Inverse Reinforcement Learning from Visual Demonstrations". In: (2020).

[32] Pieter Abbeel and Andrew Y Ng. "Apprenticeship learning via inverse reinforcement learning". In: *International Conference on Machine learning*. 2004.

[33] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. "Maximum margin planning". In: *International Conference on Machine Learning*. 2006, pp. 729–736.

[34] Peter Englert, Ngo Anh Vien, and Marc Toussaint. "Inverse KKT: Learning cost functions of manipulation tasks from demonstrations". In: *The International Journal of Robotics Research* 36 (2017), pp. 1474–1488.

[35] Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. "Pontryagin differentiable programming: An end-to-end learning and control framework". In: *Advances in Neural Information Processing Systems*. 2020.

[36] Bastien Berret, Enrico Chiovetto, Francesco Nori, and Thierry Pozzo. “Evidence for composite cost functions in arm movement planning: an inverse optimal control approach”. In: *PLoS computational biology* 7.10 (2011).

[37] Dizan Vasquez, Billy Okal, and Kai O Arras. “Inverse reinforcement learning algorithms and features for robot navigation in crowds: an experimental comparison”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 1341–1346.

[38] Andrea Bajcsy, Dylan P Losey, Marcia K O’Malley, and Anca D Dragan. “Learning robot objectives from physical human interaction”. In: *Proceedings of Machine Learning Research* 78 (2017), pp. 217–226.

[39] Wanxin Jin, Todd D Murphey, and Shaoshuai Mou. “Learning from Incremental Directional Corrections”. In: *arXiv preprint arXiv:2011.15014* (2020).

[40] Arezou Keshavarz, Yang Wang, and Stephen Boyd. “Imputing a convex objective function”. In: *IEEE International Symposium on Intelligent Control*. 2011, pp. 613–619.

[41] Stephen Boyd and Lieven Vandenberghe. “Convex optimization. 2004”. In: *Cambridge Univ. Pr* (2004).

[42] Anne-Sophie Puydupin-Jamin, Miles Johnson, and Timothy Bretl. “A convex approach to inverse optimal control and its application to modeling human locomotion”. In: *International Conference on Robotics and Automation*. 2012, pp. 531–536.

[43] Miles Johnson, Navid Aghasadeghi, and Timothy Bretl. “Inverse optimal control for deterministic continuous-time nonlinear systems”. In: *Conference on Decision and Control*. 2013, pp. 2906–2913.

[44] LS Pontryagin, VG Boltyansky, RV Gamkrelidze, and EF Mischenko. “The mathematical theory of optimal processes”. In: *United Kingdom: Interscience Publishers* (1962).

[45] Navid Aghasadeghi and Timothy Bretl. “Inverse optimal control for differentially flat systems with application to locomotion modeling”. In: *IEEE International Conference on Robotics and Automation*. 2014, pp. 6018–6025.

[46] Alessandro Vittorio Papadopoulos, Luca Bascetta, and Gianni Ferretti. “Generation of human walking paths”. In: *Autonomous Robots* 40.1 (2016), pp. 59–75.

[47] Timothy L Molloy, Dorian Tsai, Jason J Ford, and Tristan Perez. "Discrete-time inverse optimal control with partial-state information: A soft-optimality approach with constrained state estimation". In: *IEEE Conference on Decision and Control*. 2016, pp. 1926–1932.

[48] Timothy L Molloy, Jason J Ford, and Tristan Perez. "Finite-horizon inverse optimal control for discrete-time nonlinear systems". In: *Automatica* 87 (2018), pp. 442–446.

[49] Andrea Bajcsy, Dylan P Losey, Marcia K O'Malley, and Anca D Dragan. "Learning from physical human corrections, one feature at a time". In: *ACM/IEEE International Conference on Human-Robot Interaction*. 2018, pp. 141–149.

[50] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

[51] Michael A Patterson and Anil V Rao. "GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming". In: *ACM Transactions on Mathematical Software (TOMS)* 41.1 (2014), pp. 1–37.

[52] Shaoshuai Mou, Ji Liu, and A Stephen Morse. "A distributed algorithm for solving a linear algebraic equation". In: *IEEE Transactions on Automatic Control* 60.11 (2015), pp. 2863–2878.

[53] Angelia Nedic, Asuman Ozdaglar, and Pablo A Parrilo. "Constrained consensus and optimization in multi-agent networks". In: *IEEE Transactions on Automatic Control* 55.4 (2010), pp. 922–938.

[54] Guannan Qu and Na Li. "Harnessing smoothness to accelerate distributed optimization". In: *IEEE Transactions on Control of Network Systems* 5.3 (2017), pp. 1245–1260.

[55] Xuan Wang, Jingqiu Zhou, Shaoshuai Mou, and Martin J Corless. "A distributed algorithm for least squares solutions". In: *IEEE Transactions on Automatic Control* 64.10 (2019), pp. 4217–4222.

[56] R McN Alexander. "The gaits of bipedal and quadrupedal animals". In: *The International Journal of Robotics Research* 3.2 (1984), pp. 49–59.

[57] Vladimir Joukov, Jonathan Feng Shun Lin, Kevin Westemann, and Dana Kulić. "Real-time unlabeled marker pose estimation via constrained extended Kalman filter". In: *International Symposium on Experimental Robotics*. 2018.

[58] Carolyn Kisner, Lynn Allen Colby, and John Borstad. *Therapeutic exercise: foundations and techniques*. Fa Davis, 2017.

[59] Jin Chen, Per Jönsson, Masayuki Tamura, Zhihui Gu, Bunkei Matsushita, and Lars Eklundh. "A simple method for reconstructing a high-quality NDVI time-series data set based on the Savitzky–Golay filter". In: *Remote sensing of Environment* 91.3-4 (2004), pp. 332–344.

[60] Raphael Dumas, Laurence Cheze, and J-P Verriest. "Adjustments to McConville et al. and Young et al. body segment inertial parameters". In: *Journal of biomechanics* 40.3 (2007), pp. 543–553.

[61] Claudia Pérez-D'Arpino and Julie A Shah. "Fast target prediction of human reaching motion for cooperative human-robot manipulation tasks using time series classification". In: *IEEE international conference on robotics and automation*. IEEE. 2015, pp. 6175–6182.

[62] Julieta Martinez, Michael J Black, and Javier Romero. "On human motion prediction using recurrent neural networks". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2891–2900.

[63] Jonathan F. S. Lin and Dana Kulić. "Human pose recovery using wireless inertial measurement units". In: *Physiological Measurement* 33.12 (2012), pp. 2099–2115. ISSN: 0967-3334. DOI: 10.1088/0967-3334/33/12/2099.

[64] Josip Cesic, Vladimir Joukov, Ivan Petrovic, and Dana Kulić. "Full body human motion estimation on lie groups using 3D marker position measurements". In: *IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2016, pp. 826–833. ISBN: 978-1-5090-4718-5. DOI: 10.1109/HUMANOIDS.2016.7803369.

[65] R Dumas, L Chéze, and J.-P. Verriest. "Adjustments to McConville et al. and Young et al. body segment inertial parameters". In: *Journal of Biomechanics* 40 (2007), pp. 543–553. ISSN: 0021-9290. DOI: http://dx.doi.org/10.1016/j.jbiomech.2006.02.013.

[66] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. "Continuous deep q-learning with model-based acceleration". In: *International Conference on Machine Learning*. 2016, pp. 2829–2838.

[67] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. "Learning continuous control policies by stochastic value gradients". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2944–2952.

[68] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. "Generalizing Koopman theory to allow for inputs and control". In: *SIAM Journal on Applied Dynamical Systems* 17.1 (2018), pp. 909–930.

[69] Ian Abraham and Todd D Murphey. "Active Learning of Dynamics for Data-Driven Control Using Koopman Operators". In: *IEEE Transactions on Robotics* 35.5 (2019), pp. 1071–1083.

[70] Bernard O Koopman. "Hamiltonian systems and transformation in Hilbert space". In: *Proceedings of the National Academy of Sciences of the United States of America* 17.5 (1931), p. 315.

[71] Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. "A data–driven approximation of the koopman operator: Extending dynamic mode decomposition". In: *Journal of Nonlinear Science* 25.6 (2015), pp. 1307–1346.

[72] Masahiko Haruno, Daniel M Wolpert, and Mitsuo Kawato. "Mosaic model for sensorimotor learning and control". In: *Neural computation* 13.10 (2001), pp. 2201–2220.

[73] Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised learning for physical interaction through video prediction". In: *Advances in Neural Information Processing Systems*. 2016, pp. 64–72.

[74] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. "Embed to control: A locally linear latent dynamics model for control from raw images". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2746–2754.

[75] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. "Recurrent network models for human dynamics". In: *IEEE International Conference on Computer Vision*. 2015, pp. 4346–4354.

[76] Amy Zhang, Sainbayar Sukhbaatar, Adam Lerer, Arthur Szlam, and Rob Fergus. "Composable planning with attributes". In: *International Conference on Machine Learning*. 2018, pp. 5842–5851.

[77] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[78] Steindor Saemundsson, Alexander Terenin, Katja Hofmann, and Marc Deisenroth. "Variational Integrator Networks for Physically Structured Embeddings". In: *International Conference on Artificial Intelligence and Statistics*. 2020, pp. 3078–3087.

[79] Michael Lutter, Christian Ritter, and Jan Peters. "Deep lagrangian networks: Using physics as model prior for deep learning". In: *arXiv preprint arXiv:1907.04490* (2019).

[80] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. "Symplectic odenet: Learning hamiltonian dynamics with control". In: *arXiv preprint arXiv:1909.12077* (2019).

[81] Jiequn Han and Weinan E. "Deep learning approximation for stochastic control problems". In: *Deep Reinforcement Learning Workshop, Advances in Neural Information Processing Systems* (2016).

[82] Qianxiao Li, Long Chen, Cheng Tai, and E Weinan. "Maximum principle based algorithms for deep learning". In: *Journal of Machine Learning Research* 18.1 (2017), pp. 5998–6026.

[83] Qianxiao Li and Shuji Hao. "An optimal control approach to deep learning and applications to discrete-weight neural networks". In: *arXiv preprint arXiv:1803.01299* (2018).

[84] Weinan E, Jiequn Han, and Qianxiao Li. "A mean-field optimal control formulation of deep learning". In: *Research in the Mathematical Sciences* 6.1 (2019).

[85] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. "You only propagate once: Painless adversarial training using maximal principle". In: *arXiv preprint arXiv:1905.00877* (2019).

[86] Martin Benning, Elena Celledoni, Matthias J Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb. "Deep learning as optimal control problems: models and numerical methods". In: *arXiv preprint arXiv:1904.05657* (2019).

[87] Hailiang Liu and Peter Markowich. "Selection dynamics for deep neural networks". In: *arXiv preprint arXiv:1905.09076* (2019).

[88] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. "Control of memory, active perception, and action in minecraft". In: *arXiv preprint arXiv:1605.09128* (2016).

[89] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[90] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[91] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. "Model-based reinforcement learning: A survey". In: *arXiv preprint arXiv:2006.16712* (2020).

[92] Richard S Sutton. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Machine learning proceedings 1990*. Elsevier, 1990, pp. 216–224.

[93] Richard S Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM Sigart Bulletin* 2.4 (1991), pp. 160–163.

[94] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. "Adapting deep visuomotor representations with weak pairwise constraints". In: *Algorithmic Foundations of Robotics XII*. Springer, 2020, pp. 688–703.

[95] Jeff G Schneider. "Exploiting model uncertainty estimates for safe dynamic control learning". In: *Advances in Neural Information Processing Systems*. 1997, pp. 1047–1053.

[96] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. "Using inaccurate models in reinforcement learning". In: *International Conference on Machine Learning*. 2006, pp. 1–8.

[97] Sergey Levine and Pieter Abbeel. "Learning neural network policies with guided policy search under unknown dynamics". In: *Advances in Neural Information Processing Systems*. 2014, pp. 1071–1079.

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

[99] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. "Value iteration networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2154–2162.

[100] Jiongmin Yong and Xun Yu Zhou. *Stochastic controls: Hamiltonian systems and HJB equations*. Vol. 43. Springer Science & Business Media, 1999.

[101] David H Jacobson and David Q Mayne. "Differential dynamic programming". In: (1970).

[102] Hans Georg Bock and Karl-Josef Plitt. "A multiple shooting algorithm for direct solution of optimal control problems". In: *IFAC Proceedings Volumes* 17.2 (1984), pp. 1603–1608.

[103] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[104] S Joe Qin and Thomas A Badgwell. "An overview of nonlinear model predictive control applications". In: *Nonlinear model predictive control*. Springer, 2000, pp. 369–392.

[105] Yang Wang and Stephen Boyd. "Fast model predictive control using online optimization". In: *IEEE Transactions on control systems technology* 18.2 (2009), pp. 267–278.

[106] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. "Off-road obstacle avoidance through end-to-end learning". In: *Advances in Neural Information Processing Systems*. 2006, pp. 739–746.

[107] Brandon Amos and J Zico Kolter. "Optnet: Differentiable optimization as a layer in neural networks". In: *International Conference on Machine Learning* (2017).

[108] Po-Wei Wang, Priya L Donti, Bryan Wilder, and Zico Kolter. "SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver". In: *International Conference on Machine Learning* (2019).

[109] Bryan Wilder, Bistra Dilkina, and Milind Tambe. "Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization". In: *AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 1658–1665.

[110] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. "End-to-end differentiable physics for learning and control". In: *Advances in Neural Information Processing Systems*. 2018, pp. 7178–7189.

[111] Priya Donti, Brandon Amos, and J Zico Kolter. "Task-based end-to-end model learning in stochastic optimization". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5484–5494.

[112] Masashi Okada, Luca Rigazio, and Takenobu Aoshima. "Path integral networks: End-to-end differentiable optimal control". In: *arXiv preprint arXiv:1706.09597* (2017).

[113] Marcus Pereira, David D Fan, Gabriel Nakajima An, and Evangelos Theodorou. "MPC-inspired neural network policies for sequential decision making". In: *arXiv preprint arXiv:1802.05803* (2018).

[114] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. "Differentiable MPC for End-to-end Planning and Control". In: *Advances in Neural Information Processing Systems*. 2018, pp. 8289–8300.

[115] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. "Universal Planning Networks". In: *arXiv preprint arXiv:1804.00645* (2018).

[116] Hilbert J Kappen. "Path integrals and symmetry breaking for optimal control theory". In: *Journal of Statistical Mechanics: Theory and Experiment* (2005).

[117] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[118] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. "Learning from the hindsight plan—episodic mpc improvement". In: *IEEE International Conference on Robotics and Automation*. 2017, pp. 336–343.

[119] Petros Ioannou and Bariş Fidan. *Adaptive control tutorial*. SIAM, 2006.

[120] Richard Monopoli. "Model reference adaptive control with an augmented error signal". In: *IEEE Transactions on Automatic Control* 19.5 (1974), pp. 474–484.

[121] Nhan T Nguyen. "Model-reference adaptive control". In: *Model-Reference Adaptive Control*. Springer, 2018, pp. 83–123.

[122] Chang-chieh Hang and PC Parks. "Comparative studies of model reference adaptive control systems". In: *IEEE transactions on automatic control* 18.5 (1973), pp. 419–428.

[123] Iven MY Mareels, Brian DO Anderson, Robert R Bitmead, Marc Bodson, and Shankar S Sastry. "Revisiting the MIT rule for adaptive control". In: *Adaptive Systems in Control and Signal Processing 1986*. Elsevier, 1987, pp. 161–166.

[124] PV Osborn, HP Whitaker, and A Kezer. "New developments in the design of model reference adaptive control systems". In: *Institute of Aerospace Science, New York*, pp. 61–39.

[125] Patric Parks. "Liapunov redesign of model reference adaptive control systems". In: *IEEE Transactions on Automatic Control* 11.3 (1966), pp. 362–367.

[126] Michael Green and John B Moore. "Persistence of excitation in linear systems". In: *Systems & control letters* 7.5 (1986), pp. 351–360.

[127] Michael Athans. "The matrix minimum principle". In: *Information and Control* 11.5-6 (1967), pp. 592–606.

[128] Huibert Kwakernaak and Raphael Sivan. *Linear optimal control systems*. Vol. 1. New York: Wiley-Interscience, 1972.

[129] Jonathan Ho and Stefano Ermon. "Generative adversarial imitation learning". In: *arXiv preprint arXiv:1606.03476* (2016).

[130] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. "Dynamic mode decomposition with control". In: *SIAM Journal on Applied Dynamical Systems* 15.1 (2016), pp. 142–161.

[131] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *International Conference on Learning Representations* (2015).

[132] Peng Xu, Fred Roosta, and Michael W Mahoney. "Second-order optimization for non-convex machine learning: An empirical study". In: *SIAM International Conference on Data Mining*. 2020, pp. 199–207.

[133] Saeed Ghadimi and Mengdi Wang. "Approximation Methods for Bilevel Programming". In: *arXiv preprint arXiv:1802.02246* (2018).

[134] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[135] Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. Vol. 55. U.S. Government Printing Office, 1948.

[136] Gamal Elnagar, Mohammad A Kazemi, and Mohsen Razzaghi. "The pseudospectral Legendre method for discretizing optimal control problems". In: *IEEE Transactions on Automatic Control* 40.10 (1995), pp. 1793–1796.

[137] Michael Szmuk and Behcet Acikmese. "Successive convexification for 6-DoF Mars rocket powered landing with free-final-time". In: *AIAA Guidance, Navigation, and Control Conference*. 2018, p. 0617.

[138] Wanxin Jin, Todd D Murphey, Dana Kulić, Neta Ezer, and Shaoshuai Mou. "Learning from Sparse Demonstrations". In: *arXiv:2008.02159* (2020).

[139] Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. "Recent advances in robot learning from demonstration". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3 (2020).

[140] Miha Deniša, Andrej Gams, Aleš Ude, and Tadej Petrič. "Learning compliant movement primitives through demonstration and statistical generalization". In: *IEEE/ASME transactions on mechatronics* 21.5 (2015), pp. 2581–2594.

[141] Christina Moro, Goldie Nejat, and Alex Mihailidis. "Learning and Personalizing Socially Assistive Robot Behaviors to Aid with Activities of Daily Living". In: *ACM Transactions on Human-Robot Interaction* 7.2 (2018), pp. 1–25.

[142] Dean A Pomerleau. "Efficient training of artificial neural networks for autonomous navigation". In: *Neural computation* 3.1 (1991), pp. 88–97.

[143] Peter Englert, Alexandros Paraschos, Marc Peter Deisenroth, and Jan Peters. "Probabilistic model-based imitation learning". In: *Adaptive Behavior* 21.5 (2013), pp. 388–403.

[144] Sylvain Calinon, Florent Guenter, and Aude Billard. "On learning, representing, and generalizing a task in a humanoid robot". In: *IEEE Transactions on Systems, Man, and Cybernetics* 37.2 (2007), pp. 286–298.

[145] Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Bölöni, and Sergey Levine. "Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration". In: *IEEE International Conference on Robotics and Automation*. IEEE. 2018, pp. 3758–3765.

[146] Faraz Torabi, Garrett Warnell, and Peter Stone. "Behavioral cloning from observation". In: *arXiv preprint arXiv:1805.01954* (2018).

[147] Peter Moylan and Brian Anderson. "Nonlinear regulator theory and an inverse optimal control problem". In: *IEEE Transactions on Automatic Control* 18.5 (1973), pp. 460–465.

[148] James MacGlashan and Michael L Littman. "Between imitation and intention learning". In: *International Joint Conference on Artificial Intelligence*. 2015.

[149] Peter Kingston and Magnus Egerstedt. "Time and output warping of control systems: Comparing and imitating motions". In: *Automatica* 47.8 (), pp. 1580–1588.

[150] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. "An algorithmic perspective on imitation learning". In: *arXiv preprint arXiv:1811.06711* (2018).

[151] Harold W Kuhn and Albert W Tucker. "Nonlinear programming". In: *Traces and Emergence of Nonlinear Programming*. Springer, 2014, pp. 247–258.

[152] Andreas Doerr, Nathan D Ratliff, Jeannette Bohg, Marc Toussaint, and Stefan Schaal. "Direct Loss Minimization Inverse Optimal Control." In: *Robotics: Science and Systems*. 2015.

[153] Michael JD Powell. "The BOBYQA algorithm for bound constrained optimization without derivatives". In: *Cambridge NA Report, University of Cambridge, Cambridge* (2009), pp. 26–46.

[154] Luis Miguel Rios and Nikolaos V Sahinidis. "Derivative-free optimization: a review of algorithms and comparison of software implementations". In: *Journal of Global Optimization* 56.3 (2013), pp. 1247–1293.

[155] Baris Akgun, Maya Cakmak, Jae Wook Yoo, and Andrea Lockerd Thomaz. "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective". In: *ACM/IEEE international conference on Human-Robot Interaction*. 2012, pp. 391–398.

[156] Michael A Nielsen. *Neural networks and deep learning*. Determination press San Francisco, CA, 2015.

[157] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. "Geometric tracking control of a quadrotor UAV on SE(3)". In: *IEEE Conference on Decision and Control*. 2010, pp. 5420–5425.

[158] Jack B Kuipers. *Quaternions and rotation sequences*. Vol. 66. Princeton University Press, 1999.

[159] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*. Vol. 87. Springer Science & Business Media, 2013.

[160] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. "Gradient descent converges to minimizers". In: *arXiv preprint arXiv:1602.04915* (2016).

[161] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. "Escaping from saddle points—online stochastic gradient for tensor decomposition". In: *Conference on Learning Theory*. 2015, pp. 797–842.

[162] Jason D Lee, Ioannis Panageas, Georgios Piliouras, Max Simchowitz, Michael I Jordan, and Benjamin Recht. "First-order methods almost always avoid saddle points". In: *arXiv preprint arXiv:1710.07406* (2017).

[163] Jerome Bracken and James T McGill. “Mathematical programs with optimization problems in the constraints”. In: *Operations Research* 21.1 (1973), pp. 37–44.

[164] Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. “Learning preferences for manipulation tasks from online coactive feedback”. In: *The International Journal of Robotics Research* 34.10 (2015), pp. 1296–1313.

[165] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. “Learning trajectory preferences for manipulators via iterative improvement”. In: *Advances in neural information processing systems*. 2013, pp. 575–583.

[166] Jason Y Zhang and Anca D Dragan. “Learning from extrapolated corrections”. In: *International Conference on Robotics and Automation*. IEEE. 2019, pp. 7034–7040.

[167] Dylan P Losey and Marcia K O’Malley. “Including uncertainty when learning from human corrections”. In: *arXiv preprint arXiv:1806.02454* (2018).

[168] Anca D Dragan, Katharina Muelling, J Andrew Bagnell, and Siddhartha S Srinivasa. “Movement primitives via optimization”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 2339–2346.

[169] Russ Tedrake. “Underactuated robotics: Learning, planning, and control for efficient and agile machines course notes for MIT 6.832”. In: *Working draft edition* 3 (2009).

[170] Stephen Boyd and Lieven Vandenberghe. “Localization and cutting-plane methods”. In: *From Stanford EE 364b lecture notes* (2007).

[171] Steven Diamond and Stephen Boyd. “CVXPY: A Python-embedded modeling language for convex optimization”. In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[172] Sergei Pavlovich Tarasov. “The method of inscribed ellipsoids”. In: *Soviet Mathematics-Doklady*. Vol. 37. 1. 1988, pp. 226–230.

[173] Donald J Newman. “Location of the maximum on unimodal surfaces”. In: *Journal of the ACM (JACM)* 12.3 (1965), pp. 395–398.

[174] Jack Elzinga and Thomas G Moore. “A central cutting plane algorithm for the convex programming problem”. In: *Mathematical Programming* 8.1 (1975), pp. 134–145.

[175] Jean-Louis Goffin and Jean-Philippe Vial. “On the computation of weighted analytic centers and dual ellipsoids with the projective algorithm”. In: *Mathematical Programming* 60.1-3 (1993), pp. 81–92.

[176] Branko Grünbaum et al. “Partitions of mass-distributions and of convex bodies by hyperplanes.” In: *Pacific Journal of Mathematics* 10.4 (1960), pp. 1257–1261.

[177] David S Atkinson and Pravin M Vaidya. “A cutting plane algorithm for convex programming that uses analytic centers”. In: *Mathematical Programming* 69.1-3 (1995), pp. 1–43.

[178] Yu Nesterov. “Cutting plane algorithms from analytic centers: efficiency estimates”. In: *Mathematical Programming* 69.1 (1995), pp. 149–176.

[179] Guan-Horng Liu and Evangelos A Theodorou. “Deep learning theory review: An optimal control and dynamical systems perspective”. In: *arXiv preprint arXiv:1908.10920* (2019).

[180] Akshunna S Dogra and William T Redman. “Optimizing Neural Networks via Koopman Operator Theory”. In: *arXiv preprint arXiv:2006.02361* (2020).

[181] Wanxin Jin, Shaoshuai Mou, and George J Pappas. “Safe Pontryagin Differentiable Programming”. In: *arXiv preprint arXiv:2105.14937* (2021).

[182] Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. “Neural certificates for safe control policies”. In: *arXiv preprint arXiv:2006.08465* (2020).

[183] Mark W Spong and Mathukumalli Vidyasagar. *Robot dynamics and control*. John Wiley & Sons, 2008.

A. EXPERIMENTAL ENVIRONMENTS

This appendix describes the experimental systems (environments) which has been used for evaluations of the methods developed in the previous sections in this thesis. We have made different simulation environments/systems in Table 3.2 as a standalone Python package, which is available at https://github.com/wanxinjin/Pontryagin-Differentiable-Programming. This environment package is easy to use and has user-friendly interfaces for customization.

A.1 Inverted Pendulum

The dynamics of the pendulum is

$$\ddot{\alpha} = \frac{-g}{l}\sin\alpha - \frac{d}{ml^2}\dot{\alpha} + \frac{u}{ml^2} \tag{A.1}$$

with α being the angle between the pendulum and direction of gravity, u being the torque applied to the pivot, $l = 1\text{m}$, $m = 1\text{kg}$, and $d = 0.1$ being the length, mass, and damping ratio of the pendulum, respectively; gravity constant $g = 10\text{m/s}^2$.

A.2 Two-link Robot Arm

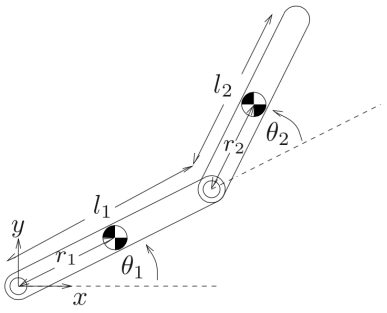


Figure A.1. Two-link robot arm with coordinate definitions

As shown in Fig. A.1, the dynamics of the two-link arm [183] is

$$M(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + C(\boldsymbol{\theta},\dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}} + \boldsymbol{g}(\boldsymbol{\theta}) = \boldsymbol{\tau}, \tag{A.2}$$

where $\boldsymbol{\theta} = [\theta_1, \theta_2]' \in \mathbb{R}^2$ is the joint angle vector; $M(\boldsymbol{\theta}) \in \mathbb{R}^{2\times 2}$ is the positive-definite inertia matrix; $C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \in \mathbb{R}^{2\times 2}$ is the Coriolis matrix; $\boldsymbol{g}(\boldsymbol{\theta}) \in \mathbb{R}^2$ is the gravity vector; and $\boldsymbol{\tau} = [\tau_1, \tau_2]' \in \mathbb{R}^2$ are the input torques applied to each joint. The parameters of the two-link robot arm in Fig. A.1 are as follows. The mass of each link is $m_1 = 1\text{kg}$, $m_2 = 1\text{kg}$; the length of each link is $l_1 = 1\text{m}$, $l_2 = 1\text{m}$; the distance from the joint to the center of mass for each link is $r_1 = 0.5\text{m}$, $r_2 = 0.5\text{m}$; the moment of inertia with respect to the center of mass for each link is $I_1 = 0.5\text{kgm}^2$, $I_2 = 0.5\text{kgm}^2$; and the gravity constant $g = 10\text{m/s}^2$. When the robot moves horizontally, $\boldsymbol{g}(\boldsymbol{\theta}) = \mathbf{0}$.

A.3 6-DoF Maneuvering Quadrotor

The equation of motion of a quadrotor flying in SE(3) (i.e., full position and attitude) space is

$$\dot{\boldsymbol{r}}_I = \dot{\boldsymbol{v}}_I, \tag{A.3a}$$

$$m\dot{\boldsymbol{v}}_I = m\boldsymbol{g}_I + \boldsymbol{f}_I, \tag{A.3b}$$

$$\dot{\boldsymbol{q}}_{B/I} = \frac{1}{2}\Omega(\boldsymbol{\omega}_B)\boldsymbol{q}_{B/I}, \tag{A.3c}$$

$$J_B\dot{\boldsymbol{\omega}}_B = \boldsymbol{\tau}_B - \boldsymbol{\omega}_B \times J_B\boldsymbol{\omega}_B. \tag{A.3d}$$

Here, subscripts $_B$ and $_I$ denote quantities expressed in the quadrotor's body frame and world frame, respectively; m is the mass of the quadrotor; $\boldsymbol{r}_I \in \mathbb{R}^3$ and $\boldsymbol{v}_I \in \mathbb{R}^3$ are its position and velocity, respectively; $J_B \in \mathbb{R}^{3\times 3}$ is its moment of inertia expressed in the body frame; $\boldsymbol{\omega}_B \in \mathbb{R}^3$ is its angular velocity; $\boldsymbol{q}_{B/I} \in \mathbb{R}^4$ is the unit quaternion [158] describing the attitude of the quadrotor's body frame with respect to the world frame; (A.3c) is the time derivative of the quaternion with $\Omega(\boldsymbol{\omega}_B)$ the matrix form of $\boldsymbol{\omega}_B$ used for quaternion multiplication [158]; $\boldsymbol{\tau}_B \in \mathbb{R}^3$ is the torque vector applied to the quadrotor; and $\boldsymbol{f}_I \in \mathbb{R}^3$ is the total force vector applied to the its center of mass (COM). The total force magnitude $\|\boldsymbol{f}_I\| = f \in \mathbb{R}$ (along

the z-axis of the quadrotor's body frame) and the torque $\boldsymbol{\tau}_B = [\tau_x, \tau_y, \tau_z]$ are generated by thrusts $[T_1, T_2, T_3, T_4]$ of the four rotating propellers, which has the following relation

$$\begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -l_{\mathrm{w}}/2 & 0 & l_{\mathrm{w}}/2 \\ -l_{\mathrm{w}}/2 & 0 & l_{\mathrm{w}}/2 & 0 \\ c & -c & c & -c \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}, \tag{A.4}$$

with l_{w} denoting the quadrotor's wing length and c a fixed constant, here $c = 0.1$. In the dynamics, gravity constant $\|\boldsymbol{g}_I\|$ is set as $10\mathrm{m/s}^2$ and the other parameters are units.

A.4 Cartpole

The equation of the motion for the cartpole system is

$$\ddot{x} = \frac{u + m_p \sin\theta(l\dot{\theta}^2 - g\cos\theta)}{m_c + m_p(\sin\theta)^2}, \tag{A.5a}$$

$$\ddot{\theta} = \frac{u\cos\theta + m_p l\dot{\theta}^2 \cos\theta \sin\theta - (m_c + m_p)g * \sin\theta}{l(m_c + m_p(\sin\theta)^2)}, \tag{A.5b}$$

with the constants set as $m_c = 0.5\mathrm{kg}$, $m_p = 0.5\mathrm{kg}$, $g = 10\mathrm{m/s}^2$, $l = 1\mathrm{m}$.