Efficient, Interpretable Robot Learning via Control-Theoretic Approaches

by

Xubo Lyu

M.Sc., Beihang University, 2018
B.Sc., Northeastern University at Qinhuangdao, 2015

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

© Xubo Lyu 2024
SIMON FRASER UNIVERSITY
Spring 2024

Copyright in this work is held by the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.

Declaration of Committee

Name:	**Xubo Lyu**
Degree:	**Doctor of Philosophy**
Thesis title:	**Efficient, Interpretable Robot Learning via Control-Theoretic Approaches**
Committee:	**Chair:** Hang Ma Assistant Professor, Computing Science
	Mo Chen Supervisor Assistant Professor, Computing Science
	Ye Pu Committee Member Assistant Professor, Department of Electrical and Electronic Engineering University of Melbourne
	Ke Li Committee Member Assistant Professor, Computing Science
	Jason Peng Examiner Assistant Professor, Computing Science
	Inna Sharf External Examiner Professor Department of Mechanical Engineering McGill University

Abstract

The realization of robotic intelligence requires the robots to autonomously sense, plan and control themselves within unknown environments. With the advancements in machine learning and deep neural networks, learning-based approaches have demonstrated great potential in enabling robots to accomplish complex tasks by actively learning from data. Reinforcement Learning (RL), as a representative technique, empowers robots to learn skills from interactive experiences of the environment via high-dimensional states and observations. However, learning-based approaches can suffer from data inefficiency, making it costly and unrealistic to apply them to real-world physical systems. Additionally, learning-based approaches lack formal mathematical tools for the analysis and interpretation of the learning results. In contrast, the long-established, classical control-theoretic approaches can model and derive control policies for dynamical systems in a data-efficient way and are equipped with well-developed theories for system and control analysis. However, they often operate under the assumption of having prior knowledge about the system dynamics and the environment and are limited to small-scale problems with low-dimensional state space.

In this dissertation, we aim to combine the control-theoretic principle with modern learning-based approaches to achieve an efficient and interpretable robot learning process while scaling up the classical control techniques. In chapter 3, we provide methods to improve learning efficiency with control-based value functions in the subspaces of high-dimensional problems that the learning-based approaches are tasked to solve. These value functions are efficiently computed via control approaches and can be seamlessly integrated into the learning process as novel reward and baseline functions. Then in chapter 5, we study the multi-agent centralized learning efficiency under the hierarchy framework and show that a conditional policy coupled together a special form of trajectory can achieve efficient asynchronous, hierarchical decision-making. In chapter 4, we incorporate a linear controller and a linear latent model into the gradient-driven end-to-end optimization over contrastive latent representation space, making the learned system and control explainable as well as extending the classical control from low to high-dimensional complex nonlinear scenarios.

Keywords: Robotics; Reinforcement Learning; Control Theory;

Dedication

To my parents Baoling Yang and Shunchang Lyu, my sister Linlin Lyu, and my grandma Yueting Zhang. Also to my wife Lili Liu, and our beloved baby Glenn. And to my lovely neice and nephew Dayue, Darui.

Acknowledgements

I would like to express my heartfelt gratitude to the many people who have played a pivotal role in my Ph.D. journey. Without their support, encouragement, and contributions, the completion of this thesis would not have been possible.

First and foremost, I express my profound thanks to my advisor, Prof. Mo Chen, whose expertise, mentorship, patience, and rationality have deeply shaped and improved my academic growth. Meanwhile, I extend my heartfelt gratitude to my co-adviser, Prof. Ye Pu, who has devoted invaluable guidance throughout my Ph.D. journey.

I would like to extend my sincere appreciation to the esteemed members of my thesis committee: Prof. Jason Peng, Prof. Manolis Savva, Prof. Inna Sharf, and Prof. Hang Ma for their honored presence, valuable insights, and critical review. Your expertise and unwavering support have greatly enriched the quality of this thesis.

To my lab mates and colleagues at Simon Fraser University Multi-Agent Robotics System Lab and the University of Melbourne, including but not limited to Site Li, Rakesh Shrestha, Seth Siriya, Hanyang Hu, Joe Zhang, Payam Nikdel, Mohammad Mahdavian, Sahar Leisiazar, Pedram Agand, Jack Thomas, Geoff Nagy, Michael Lu, Minh Bui, Jimin Rhim, Atefeh Sahraee, Salar Asayesh, Tara Toufighi, Sriraj Meenavilli and many others. I am grateful for the time we spent together. The fun time, stimulating discussions, and collaborative spirit have made this journey more enriching and enjoyable. I'm also thankful to the industrial collaborators Amin Banitalebi Dehkordi and Yong Zhang, who provided guidance and support for my research.

I'd like to especially thank my friends whom I met in Vancouver, Kai Zhao, Zhenqi Zhu, Anjian Li, Zhitian Zhang, and their families. They gave me invaluable warmth and support when I first stepped onto this new place.

Finally, I express my profound thanks to my family for their continuous encouragement and understanding during the challenges of this endeavor. In many ways, they are the reason I persevere. My gratitude also goes to all those whose names may not appear here but who have supported me in various ways. Thank you all!

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Robots are revolutionizing how we live and work. Whether it is the ever-present cars for daily transportation, drones soaring above for surveillance and photography, or the precision of robotic arms in industrial assembly, alongside the versatile quadruped and humanoid robots offering a range of services, these machines consistently contribute to improved efficiency, safety, and a notable reduction in human error in every facet of our lives.

Depending on the level of autonomy, robots can be roughly categorized as non-autonomous, semi-autonomous, and fully-autonomous. Non-autonomous robots, like human-driven cars, require extensive manual control. Semi-autonomous robots, such as industrial robot arms, can be programmed by pre-defined rules to finish repetitive tasks and require human intervention for adaptation. Fully-autonomous robots, exemplified by self-driving vacuums and cars, possess the capability to self-maintain, actively sense their environment, and independently perform tasks without the need for human intervention.

Achieving fully-autonomous robots is the hallmark of robotic research, and the road toward it is promising yet challenging. One critical concern is that autonomous robots need to adapt themselves to dynamic and *a priori* unknown environments. They must also react reasonably based on rich, high-dimensional sensory observations (e.g. from cameras, Radars, and LiDARs) of the surrounding environment. For instance, self-driving cars must properly navigate under ever-changing conditions, from different types of roads, unexpected construction, and crowded pedestrian areas, to diverse weather and lighting patterns. These dynamic environmental factors are impossible to predict in advance and can vary every day. Thus, it is infeasible to design decision rules for such infinite, real-time situations manually.

1.1 Learning- and Control-Based Methods

One way to enable the adaptation of autonomous robots in a dynamic and complex environment is to allow them to *self-learn* knowledge and skills from a large amount of prior data or experiences within various environments. That is also referred to as robot learning. Reinforcement Learning (RL), as a representative robot learning paradigm, is one type

of machine learning technique specially designed for the decision-making and skill acquisition of autonomous robots. It has been demonstrated to be a powerful technique for many highly complex robot control problems on drones, quadruped robots, manipulators [94, 60, 96, 35, 102, 173, 150, 128, 101, 55, 70]. RL enables a robot to learn strategies from historical data which is generated through robot-environment interactive experiences. It has the advantage of dealing with problems with potentially high-dimensional states and observations and is capable of directly learning a control policy mapping between states and actions. Despite the advantages, RL also suffers from several issues. One of them is data inefficiency, i.e., it requires an impractically large number of training samples to acquire specific behavioral policy, especially when learning purely from ordinary experiences from scratch. This is costly and even unrealistic in many robotic scenarios. Another issue lies in the learning process and outcomes being encapsulated within a "black box" due to the extensive use of parameterized structure (e.g. neural networks), which results in a lack of explainability. This means we are unable to gain insights into the reasons behind or the mechanisms through which the robotic system achieves specific performance levels.

In contrast to data-driven robot learning, classical control-theoretic approaches, exemplified by optimal control, stand out as an analytical, computational framework for modeling and solving robot control problems. These approaches have been widely used in developing autonomous systems for decades, with numerous well-known control techniques such as Linear Quadratic Regulator (LQR), Model Predictive Control (MPC), Reachability Analysis, etc [152, 20, 26, 105, 106, 153, 24, 113, 92]. Unlike RL methods that learn control policy from prior data, classical control approaches solve problems with explicitly defined objectives and conditions analytically or numerically without the need for data. Therefore, classical control is data-efficient in solution finding and allows for theoretical analysis of the system and solution. Moreover, the theory behind classical control-theoretic approaches has been well-established for decades thus providing good interpretability for its computational process and outcomes. However, classical control usually operates under the assumption that the known, accurate system and environment model is available, which limits their uses to problems with relatively low-dimensional states and perfect system knowledge.

Given the pros and cons of learning and control techniques, in this dissertation, we seek the combinational methods to inherit the benefits of both, that is, combining the control-theoretic principle with modern learning-based approaches to achieve an efficient and interpretable robot learning process while scaling up the classical control techniques. The presence of control provides us with approximate system priors for reduced sample complexity, guided RL exploration, and theoretical analysis while the presence of learning enables direct control policy optimization over high-dimensional, sensory states and observations, which is widely used in modern robot systems like self-driving cars. In this dissertation, we will focus on reinforcement learning and optimal control to be the typical learning-based and control-based approaches respectively.

1.2 Thesis Overview

The work presented in this thesis explores the combinational use of learning- and control-based methods for robot control. The findings showcase improved data efficiency and interpretability in learning, along with improved scalability on classical control. We start with a review of fundamental concepts in reinforcement learning and optimal control in Chapter 2. Then, the following chapters are organized as follows.

Chapter 3: Efficient Learning via Control-Based Value Functions. One type of RL method, also known as model-free RL, learns control policies from scratch purely based on the interactive environment data. Though flexible to be applied to realistic robot control problems with high-dimensional (hi-dim), continuous states, and observations, it tends to be very data-inefficient and sometimes with poor exploration of the environment. In this chapter, we present two works to incorporate optimal control-based value functions into the learning process to alleviate its data inefficiency and enhance the exploration. We do this by first identifying a relatively lower-dimensional (lo-dim) optimal control (OC) problem from the given hi-dim RL problem. This OC problem summarizes key system dynamics and environment information in a lo-dim subspace and allows efficient offline computation of its value function in a data-free way. Then this OC value function can be used in the RL process as an informative reward or baseline function. Even though the OC problem and its related models are lo-dim and approximate in terms of the given hi-dim RL problem, we show that its value function is still helpful in improving the data efficiency and exploration of the learning process. Our methods can be extended to involve more task-specific information and easily incorporated into any RL algorithm as an add-on without altering the RL structure and are compatible with other techniques that may facilitate the RL process.

Chapter 4: Scalable and Interpretable Learning-Based Control. A key limitation of classical control methods is their poor scalability when applied to hi-dim robot control problems, primarily stemming from the computational complexity associated with non-linearity, and at times, non-convexity of the problem objective function and system model, especially with the use of neural networks. In this chapter, we explore the use of Koopman Operator Theory [64, 13], a mathematical framework of using linear-evolving latent representation and model to describe a nonlinear dynamical system, in the RL process. We present task-oriented Koopman-based control that utilizes end-to-end RL as well as a contrastive state representation to simultaneously learn the latent embedding, a linear Koopman operator, and the associated LQR controller. With the learned linear latent system model and its LQR controller, our work scales optimal control from low to high-dimensional, complex nonlinear systems, including pixel-based tasks and a real robot with lidar observations. Meanwhile, our method provides extra interpretability to the learning process due to the linear constraints of the learned model and controller. For example, we

can conduct stability and controllability analysis on the learned linear latent dynamical system to gain insights into the best choice of latent state dimensions.

Chapter 5: Efficient Multi-Agent Centralized Learning over Asynchronous Options. In this chapter, we primarily deal with the efficiency of multi-agent centralized learning. Specifically, we consider the problem where multiple robots coordinate in a centralized manner among a set of discrete higher-level actions (i.e. actions encompassing multiple primitive steps), also known as options [134]. One issue of this setup is that multi-robot option executions are often asynchronous, that is, agents may select and complete their options at different time steps. This may cause inefficiency in centralized learning, as a subset of robots may complete their options earlier, resulting in unnecessary waiting times for the others which is a potential waste of time. Therefore, we proposed a conditional reasoning approach that can employ multi-agent centralized policy gradient RL asynchronously over options. With our method, the centralized robots can make decisions more efficiently: they do not need to make synchronous decisions (e.g. waiting for the others to complete the options), but make decisions whenever they need them, without interfering with the others. We demonstrate the effectiveness of our method on several option-based multi-agent cooperative tasks through empirical validation.

Chapter 2

Background

The works in this thesis mainly investigate the combinational methods of reinforcement learning and optimal control, which enables a more efficient, interpretable learning process as well as scalable control capability. In this chapter, we provide a review of fundamental concepts related to our works, ranging from reinforcement learning, optimal control, Koopman operator theory as well as option-based multi-agent reinforcement learning, and introduce the notations that we will be using in the following chapters.

2.1 Reinforcement Learning

Our works use reinforcement learning (RL) as the framework for learning-based robot control. In this section, we briefly introduce the related concepts of RL methods. RL is a subfield of machine learning that focuses on the decision-making process of intelligent agents interacting with an environment, aiming to optimize their long-term cumulative rewards. To do this, RL methods enable agents (e.g. robots) to learn behaviors through a trial-and-error process and reward (or penalize) an agent's behaviors to make them more likely to be repeated (or avoided) in the future.

2.1.1 Markov Decision Process

RL is typically formulated as a process where an agent interacts with an environment, which is mathematically modeled as a Markov decision process (MDP). In a typical MDP, the interactions between the agent and environment are organized into episodes. At the beginning of each episode, the agent starts in an initial state $\mathbf{s}_0 \in \mathcal{S}$ sampled from an initial state distribution $\mathbf{s}_0 \sim p(\mathbf{s}_0)$, where $\mathcal{S}$ denotes the state space of the MDP. At each discrete time step k[1], the agent observes states $\mathbf{s}_k \in \mathcal{S}$ and selects an action $\mathbf{a}_k \in \mathcal{A}$ from its policy $\mathbf{a}_k \sim \pi(\mathbf{a}_k \mid \mathbf{s}_k)$. The action $\mathbf{a}_k$ is then executed within the environment, resulting in a new state $\mathbf{s}_{k+1}$ and a scalar reward $r_k = r(\mathbf{s}_k, \mathbf{a}_k, \mathbf{s}_{k+1})$ in response, where $\mathbf{s}_{k+1}$ is

[1]In this thesis, we denote discrete time step as k, continues time step as t.

sampled from the MDP dynamics $\mathbf{s}_{k+1} \sim p(\mathbf{s}_{k+1} \mid \mathbf{s}_k, \mathbf{a}_k)$ and r_k reflects the desirability of the state transition in the context of given task and is usually heuristically determined. This interaction process can repeat up to a time horizon of T (could be infinite) and form a trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, r_0, \mathbf{s}_1, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, r_{T-1}, \mathbf{s}_T)$. The objective is to find an optimal policy π^* maximizing the expected discounted return $J(\pi)$,

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.1}$$

$$J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[R(\tau) \right] = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[\sum_{k=0}^{T-1} \gamma^t r_k \right], \tag{2.2}$$

where $p(\tau|\pi) = p(\mathbf{s}_0) \prod_{k=0}^{T-1} p(\mathbf{s}_{k+1}|\mathbf{s}_k, \mathbf{a}_k) \pi(\mathbf{a}_k|\mathbf{s}_k)$ is the distribution of trajectories induced by a policy π. $R(\tau) = \sum_{k=0}^{T-1} \gamma^t r_k$ is the trajectory return, $\gamma \in [0, 1]$ is a discount factor.

2.1.2 Value Functions and Value Iteration

An optimal policy can be found indirectly by first estimating its performance, namely, the value of the policy. Value function is a fundamental concept in RL and serves as a crucial component in almost every RL method. The value function estimates a policy's expected return when one initiates from every possible state and/or action and continues to follow this particular policy indefinitely. As shown in Eq. (2.3), the on-policy state value function $V^\pi(\mathbf{s})$ gives the estimated expected return if you start in state $\mathbf{s}$ and always act according to policy π, while the optimal state value function $V^*(\mathbf{s})$ did the same except one always acts according to the optimal policy π^* (i.e. replacing π with π^*) in the environment,

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{\tau \sim p(\tau|\pi, \mathbf{s}_0 = \mathbf{s})} \left[R(\tau) \right] \\ Q^\pi(\mathbf{s}, \mathbf{a}) &= \mathbb{E}_{\tau \sim p(\tau|\pi, \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a})} \left[R(\tau) \right]. \end{aligned} \tag{2.3}$$

Similarly, the on-policy state-action value function $Q^\pi(\mathbf{s}, \mathbf{a})$, commonly referred to as the Q-function, estimates a policy's expected future return from acting a at state $\mathbf{s}$ and then following π for all future timesteps, while its optimal version uses the optimal policy.

All these value functions obey special self-consistency equations called Bellman equations [11]. The basic idea behind the Bellman equations is that the value of the starting state is the reward one expects to get from being there, plus the value of wherever one lands next,

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s}), \mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} \left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma V^\pi(\mathbf{s}') \right], \\ Q^\pi(\mathbf{s}, \mathbf{a}) &= \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} \left[r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \mathbb{E}_{\mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s})} \left[Q^\pi(\mathbf{s}', \mathbf{a}') \right] \right], \end{aligned} \tag{2.4}$$

where $\mathbf{s}'$ and $\mathbf{a}'$ are the next state and action after $\mathbf{s}$ and $\mathbf{a}$. The optimal version is

$$\begin{aligned} V^*(\mathbf{s}) &= \max_{\mathbf{a}} \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[r(\mathbf{s},\mathbf{a},\mathbf{s}') + \gamma V^*\left(\mathbf{s}'\right) \right], \\ Q^*(\mathbf{s},\mathbf{a}) &= \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[r(\mathbf{s},\mathbf{a},\mathbf{s}') + \gamma \max_{\mathbf{a}'} \left[Q^{\pi}\left(\mathbf{s}',\mathbf{a}'\right) \right] \right], \end{aligned} \tag{2.5}$$

and the right-hand side of the Bellman equation is also known as the Bellman backup.

Bellman equation can be used iteratively to compute the optimal value function, resulting in the classical method named *value iteration*. It uses dynamic programming to maintain an approximate value function and iteratively improves it until convergence.

$$V_{i+1}(\mathbf{s}) \leftarrow \max_{\mathbf{a}} \left\{ \sum_{\mathbf{s}'} p\left(\mathbf{s}' \mid \mathbf{s},\mathbf{a}\right) \left\{ r\left(\mathbf{s},\mathbf{a},\mathbf{s}'\right) + \gamma V_i\left(\mathbf{s}'\right) \right\} \right\} \tag{2.6}$$

where i is the number of iterations. Value iteration starts at $i = 0$ and V_0 as a guess of the value function. It then iterates, repeatedly computing V_{i+1} for all states $\mathbf{s}$, until it converges to V^*. One can then derive the optimal control policy as follows:

$$\pi^*(\mathbf{s}) = \arg\max_{\mathbf{a}} \left\{ \sum_{\mathbf{s}'} p\left(\mathbf{s}' \mid \mathbf{s},\mathbf{a}\right) \left\{ r\left(\mathbf{s},\mathbf{a},\mathbf{s}'\right) + \gamma V^*\left(\mathbf{s}'\right) \right\} \right\}, \quad \forall \ \mathbf{s} \in \mathcal{S}. \tag{2.7}$$

With the strong theoretical guarantee of convergence, value iteration is efficient in solving RL problems with lo-dim state and action spaces, especially when these spaces can be represented in discrete, tabular form. Similarly, state action value function $Q(\mathbf{s},\mathbf{a})$ also can be used in the value iteration method, shown in Eq. (2.8). The good thing about $Q(\mathbf{s},\mathbf{a})$ is that it has an explicit representation of the value at each state-action pair, making it easy to derive the optimal policy even if the transition model $P(\mathbf{s}' \mid \mathbf{s},\mathbf{a})$ is not available.

$$Q^*(\mathbf{s},\mathbf{a}) \leftarrow \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[r(\mathbf{s},\mathbf{a},\mathbf{s}') + \gamma \max_{\mathbf{a}'} Q^*\left(\mathbf{s}',\mathbf{a}'\right) \right] \tag{2.8}$$

Estimating the value function $V(\mathbf{s})$ or $Q(\mathbf{s},\mathbf{a})$ and then deriving the optimal policy from them is one of the mainstream methods of RL, named value-based method. More variants of are exemplified by SARSA [111, 131], TD(λ) [131], and Q-learning [131].

2.1.3 Policy Gradient Methods

Other than the value-based method, an alternative yet powerful RL method is to directly search and optimize the policy within the policy space, without estimating the value function beforehand, which therefore is named the policy-based method.

One typical policy-based method that is mostly used in this thesis is the policy gradient method [133]. The policy gradient method iteratively updates the parameters of a policy via gradient ascent using an empirical estimate of the gradient of the policy's expected return

to its parameters $\nabla_\pi J(\pi)$. To optimize the policy directly, we usually represent the policy via a set of parameters θ as a parameterized policy π_θ. To optimize the policy directly with regard to the aforementioned objective $J(\pi_\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} R(\tau)$, we utilize iterative gradient ascent, where the policy parameters are updated at ith iteration with learning rate α by:

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta J\left(\pi_\theta\right)|_{\theta_i}. \tag{2.9}$$

In practice, we need an expression for the policy gradient which we can numerically compute. This involves two steps: 1) deriving the analytical gradient of policy performance, which turns out to have the form of an expected value, and then 2) forming a sample estimate of that expected value, which can be computed with data from a finite number of agent-environment interaction steps. From the results of [133, 160], the policy gradient can be derived and estimated in an expectation form:

$$\begin{aligned}
\nabla_\theta J\left(\pi_\theta\right) &= \nabla_\theta \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} R(\tau) \\
&= \nabla_\theta \int_\tau p(\tau \mid \pi_\theta) R(\tau) \\
&= \int_\tau \nabla_\theta p(\tau \mid \pi_\theta) R(\tau) \\
&= \int_\tau P(\tau \mid \pi_\theta) \nabla_\theta \log p(\tau \mid \pi_\theta) R(\tau) \\
&= \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \nabla_\theta \log p(\tau \mid \theta) R(\tau) \\
&\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{k=0}^{T} \nabla_\theta \log \pi_\theta\left(\mathbf{a}_k \mid \mathbf{s}_k\right) R(\tau),
\end{aligned} \tag{2.10}$$

where we can estimate it with a sample mean by collecting a set of episodes $\mathcal{D} = \{\tau_i\}_{i=1,\ldots,N}$ where each episode is obtained by letting the agent act in the environment using the policy π_θ and $|\mathcal{D}|$ is the number of trajectories in $\mathcal{D}$. In practice, the return $R(\tau)$ in Eq. (2.10) can be replaced by various other choices and result in different policy gradient methods.

$$R(\tau) \leftarrow \sum_{k'=k}^{T} \gamma^{k'-k} r\left(\mathbf{s}_{k'}, \mathbf{a}_{k'}, \mathbf{s}_{k'+1}\right) - b(\mathbf{s}_{k'}) \tag{2.11}$$

$$R(\tau) \leftarrow r(\mathbf{s}_k, \mathbf{a}_k, \mathbf{s}_{k+1}) + \gamma V^{\pi_\theta}(\mathbf{s}_{k+1}) - V^{\pi_\theta}(\mathbf{s}_k), \tag{2.12}$$

where Eq. (2.11) leads to policy gradient with general baseline functions $b(\cdot)$ while Eq. (2.12) leads to actor-critic method. These choices lead to the same expected value for the policy gradient and help to reduce the variance of the gradient estimate. Schulman et al. later proposed Generalized Advantage Estimation (GAE) [119], which is the most advanced return and widely used in various policy gradient algorithms such as REINFORCE [160], TRPO

[117], and PPO [120]. The state value function in Eq. (2.12) can be estimated by

$$V^i = \arg\min_V \mathbb{E}_{(\mathbf{s}_k, r_k, \mathbf{s}'_k)\sim\mathcal{D}} \left[\left(y_k - V\left(\mathbf{s}_k\right) \right)^2 \right], \tag{2.13}$$

where $y_k = r_k + \gamma V^{i-1}\left(\mathbf{s}'_k\right)$ is a target value that estimates the expected return of $\mathbf{s}_k$, and is computed using the value function from the previous iteration V^{i-1}. This method for computing target values is referred to as a single-step bootstrap, and multi-step variants of bootstrapping, such as TD(λ) [254], can also be used.

2.2 Optimal Control

The thesis explores integrating control-based solutions to enable an efficient learning process. In this section, we introduce related concepts about optimal control problems and methods. Optimal control is a fundamental discipline that determines the optimal control strategy for a given system with known dynamics, such that a certain optimality criterion is achieved. The optimality criterion is usually represented by a cost function that depends on both the system's states and the control inputs, meanwhile constrained by the system dynamics (a.k.a. model). In continuous cases, the objective of optimal control is to establish a set of differential equations that describe the trajectories of the control inputs in such a way as to minimize the cost function. In discrete cases, the objective is to find a control sequence over a horizon of time steps to minimize the cost function.

Differently from RL, optimal control involves a known cost function and dynamical system model, allowing for mathematical computation of its optimal solution using analytical or numerical methods, without the necessity of learning from environmental data. Mathematically, a (continuous) optimal control problem is defined by the dynamics function $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ at continuous time t for the state $\mathbf{x}(t)$ and control input $\mathbf{u}(t)$ as well as an objective function J over the trajectory $\mathbf{x}(\cdot)$ and $\mathbf{u}(\cdot)$ over the entire time horizon t_f. $\mathbf{x}(t) \in \mathbb{R}^{n_\mathbf{x}}$ and $\mathbf{u}(t) \in \mathbb{R}^{n_\mathbf{u}}$ are real-valued vectors at all t with dimensions $n_\mathbf{x}$, $n_\mathbf{u}$.

$$J(\mathbf{x}(\cdot), \mathbf{u}(\cdot)) = h(\mathbf{x}_{t_f}) + \int_0^{t_f} l(\mathbf{x}(t), \mathbf{u}(t), t)dt \tag{2.14}$$

$$\begin{aligned} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \text{ for all } t \quad & \text{(System Dynamics constraint)} \\ \mathbf{x}(0) = \mathbf{x}_0 \quad & \text{(Initial state)} \end{aligned} \tag{2.15}$$

The term $l(\mathbf{x}(t), \mathbf{u}(t), t)$ is known as the instantaneous cost (or running cost) which is accumulated over time and $h(\mathbf{x}_{t_f})$ is terminal cost. They should be chosen to be nonnegative and to penalize certain undesirable states, velocities, or controls. The goal of optimal control

is to find state-control trajectories $\{\mathbf{x}(t), \mathbf{u}(t) : 0 \leq t \leq t_f\}$ such that J is minimized:

$$\begin{gathered} \mathbf{x}^\star(\cdot), \mathbf{u}^\star(\cdot) = \arg \min_{\mathbf{x}(\cdot), \mathbf{u}(\cdot)} J(\mathbf{x}(\cdot), \mathbf{u}(\cdot)) \text{ such that} \\ \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \text{ for all } t \quad \text{(System Dynamics constraint)} \\ \mathbf{x}(0) = \mathbf{x}_0 \quad \text{(Initial state)} \end{gathered} \tag{2.16}$$

A variety of behaviors can be specified in this framework by modifying the instantaneous cost. For example, (1) trajectory tracking for a trajectory $\mathbf{x}_D(t)$ can be implemented by penalizing squared error $l(\mathbf{x}(t), \mathbf{u}(t), t) = \|\mathbf{x}(t) - \mathbf{x}_D(t)\|^2$, (2) minimizing effort can be defined in terms of a control penalty $\|\mathbf{u}(t)\|^2$, (3) minimum time to hit a target $\mathbf{x}_{\text{tgt}}$ could be implemented as an indicator function $I[\mathbf{x} \neq \mathbf{x}_{\text{tgt}}]$ where $I[\cdot]$ is 1 if the condition is true, and 0 otherwise, (4) obstacle avoidance can be implemented by a repulsive barrier that decreases to 0 when the distance to the closest obstacle d exceeds some minimum buffer distance $d_{\min}$ and increases to infinity as the distance shrinks to 0. One common form of this barrier is $l(\mathbf{x}(t), \mathbf{u}(t), t) = 1/d^2 - 1/d_{\min}^2$ when $d < d_{\min}$ and $l(\mathbf{x}(t), \mathbf{u}(t), t) = 0$ otherwise.

Optimal control problem can also be formulated as a discrete version [141] where the continuous time is replaced by a sequence of discrete time steps, which allows for numerical approximation and computation. However, the core concepts and components are still the same. Depending on the different forms and meanings of cost functional and system dynamics, several types of optimal control problems used in this thesis are illustrated.

2.2.1 Time-to-Reach (TTR) Problem

Many applications in engineering deal with minimum time control problems, whose goal is to compute a control function that is capable of steering a dynamical system from a given initial state to a desired target configuration in the shortest possible time [75, 168, 30]. In this case, the time needed to reach the target is considered a cost function. Hence, a minimum-time problem can be recast in the form of an optimal control problem.

A TTR problem is formally defined as a two-player-zero-sum differential game following the continuous system dynamics $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{d}(t))$. A formal definition of the TTR problem is about finding the time to reach the target $\Gamma \in \mathbb{R}^n$ with a compact boundary from any starting point at $\boldsymbol{x} = \mathbf{x}(0)$ when player I wants to maximize the time, while player II uses a strategy to minimize the time with knowledge of player I's current and past decisions [168]. To this end, we begin by introducing the mathematical notion of the time to reach a closed target with a compact boundary, given player I's strategy as control $\mathbf{u}(\cdot)$ and player II's strategy as disturbance $\mathbf{d}(\cdot)$ over the entire trajectory horizon of the system,

$$\mathcal{T}_{\boldsymbol{x}}[\mathbf{u}(\cdot), \mathbf{d}(\cdot)] = \min\{t \mid \mathbf{x}(t) \in \Gamma\}, \tag{2.17}$$

We use $\mathbf{d_u}$ to denote the control of player II $\mathbf{d}(t)$ with knowing the strategy of the control $\mathbf{u}(t)$ of player I. Then, we have a differential game problem

$$\phi(\boldsymbol{x}) := \min_{\mathbf{d}(\cdot)} \max_{\mathbf{u}(\cdot)} \mathcal{T}_{\boldsymbol{x}}[\mathbf{u}(\cdot), \mathbf{d_u}(\cdot)], \tag{2.18}$$

where $\phi(\cdot)$ is called the lower value function of the differential game problem. In this paper, we assume that the minimum and the maximum in Eq. (2.18) exist. Under this assumption, the capturability set $\mathcal{C}^* = \{\boldsymbol{x} \in \mathbb{R}^n \mid \phi(\boldsymbol{x}) < +\infty\}$ coincides with the state space $\mathbb{R}^n$. Applying the dynamic programming principle, we can derive the following stationary Hamilton-Jacobi-Isaacs (HJI) equation with viscosity solution is the value function ϕ of the differential game Eq. (2.18):

$$\begin{aligned} H(\boldsymbol{x}, \nabla\phi(\boldsymbol{x})) &= 0, \quad \text{in } \mathcal{C}^* \backslash \Gamma \\ \phi(\boldsymbol{x}) &= 0, \quad \text{on } \Gamma, \end{aligned} \tag{2.19}$$

where H denotes the Hamiltonian defined by

$$H(\boldsymbol{x}, \boldsymbol{p}) = \min_{a} \max_{b} \left\{ -\boldsymbol{p}^\top \mathbf{f}(\boldsymbol{x}, a, b) - 1 \right\}. \tag{2.20}$$

Detailed derivations and discussions are presented in [7, 9, 34]. The viscosity solution $\phi(\cdot)$ of Eq. (2.19) is the so-called *time-to-reach* function, and it is a weak solution that is consistent and unique in the domain. Having the lower value function ϕ as the HJI viscosity solution guarantees that the numerical approximation we found for the viscosity solution will be the one for the lower value function due to the viscosity solution's uniqueness and consistency. Mathematical tools from [93, 168] have been developed to solve Eq. (2.19) efficiently. The TTR problem can reduce to a uni-variate optimization that is only dependent on $\min_{\mathbf{d}(\cdot)}$ or $\max_{\mathbf{u}(\cdot)}$ where Eq. (2.19) becomes a Hamilton-Jacobi-Bellman (HJB) equation.

2.2.2 Linear Quadratic Regulator Problem

The simplest class of optimal control problems is Linear Time-Invariant (LTI) systems with its costs defined in a quadratic form of state $\mathbf{x}$ and control $\mathbf{u}$, which is called the LQ problem. The solution to the LQ problem is provided by the linear–quadratic regulator (LQR). The optimal control for this problem can be determined analytically as a closed-form function of the states. LQR problem has finite/infinite and discrete/continuous variations, and we only discuss the infinite-horizon discrete LQR here.

Consider a discrete-time linear system with a quadratic cost described by

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{2.21}$$

$$J = \sum_{k=0}^{\infty} \left((\mathbf{x}_k - \bar{\mathbf{x}})^\top \mathbf{Q} (\mathbf{x}_k - \bar{\mathbf{x}}) + \mathbf{u}_k^\top \mathbf{R} \mathbf{u}_k \right), \tag{2.22}$$

where $\bar{\mathbf{x}}$ is the desired goal state. $\mathbf{Q}$ and $\mathbf{R}$ are positive semi-definite symmetric matrices respectively. The magnitude of entries of $\mathbf{Q}$ penalizes error from the desired state, and the magnitude of entries of $\mathbf{R}$ penalizes control effort. The optimal control at each time step k can be shown to be a linear function of $\mathbf{x}_k$,

$$\mathbf{u}_k = -\mathbf{K}(\mathbf{x}_k - \bar{\mathbf{x}}), \tag{2.23}$$

where the LQR gain $\mathbf{K} = \left(\mathbf{R} + \mathbf{B}^\top \mathbf{P} \mathbf{B} \right)^{-1} \left(\mathbf{B}^\top \mathbf{P} \mathbf{A} \right)$ is computed as a function of an unknown matrix $\mathbf{P}$. $\mathbf{P}$ is a unique positive definite solution to the discrete-time algebraic Riccati equation (DARE) [140]

$$\mathbf{P} = \mathbf{A}^\top \mathbf{P} \mathbf{A} - \mathbf{A}^\top \mathbf{P} \mathbf{B} (\mathbf{R} + \mathbf{B}^\top \mathbf{P} \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{P} \mathbf{A} + \mathbf{Q}. \tag{2.24}$$

Numerical methods are available for solving the Riccati equation for $\mathbf{P}$. One simple way is to iteratively update $\mathbf{P}$ following Eq. (2.24) from the right-hand side to the left-hand side for many iterations until $\mathbf{P}$ converges. The significance of this LQR controller is that the performance metric is made explicit rather than implicit. Moreover, it gives a closed-form globally-optimal solution for any linear dynamical system model specified by $\mathbf{A}, \mathbf{B}$, so if more information is gathered that yields a better estimate for $\mathbf{A}$ and $\mathbf{B}$, the LQR method can be applied directly to obtain the optimal gains.

2.2.3 Model Predictive Control (MPC)

MPC method is a process for solving the optimal control problem by deriving a closed-loop controller from open-loop trajectories. Generally speaking, it simply replans a new trajectory starting from the sensed state at each step. It executes some small portion of that trajectory, senses the new state, and then replans again. By repeating this process, MPC can cope with unexpected disturbances by dynamically calculating paths to return to desirable states. MPC can handle very complex, nonlinear system dynamics and general forms of the cost function, alongside unequal or equal constraints.

To define the MPC process, consider an optimal control problem in discrete time, where we have the control loop operating at rate Δt and a time horizon $T = N\Delta t$ for optimization,

$$\begin{aligned}
\min_{\mathbf{x}_{0:N+1},\mathbf{u}_{0:N}} \quad & h\left(\mathbf{x}_{N+1}\right) + \sum_{k=0}^{N} l\left(\mathbf{x}_k, \mathbf{u}_k\right) & & \\
\text{subject to:} \quad & \mathbf{x}_0 = \hat{\mathbf{x}}_0, & & \\
& \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right)\Delta t, & \forall k = 0, \ldots, N, & \\
& g\left(\mathbf{x}_k, \mathbf{u}_k\right) \leq 0 & \forall k = 0, \ldots, N, & \\
& \mathbf{x}_{\text{lb}} \leq \mathbf{x}_k \leq \mathbf{x}_{\text{ub}}, & \forall k = 0, \ldots, N, & \\
& \mathbf{u}_{\text{lb}} \leq \mathbf{u}_k \leq \mathbf{u}_{\text{ub}}, & \forall k = 0, \ldots, N, & \\
& g_{\text{terminal}}\ \left(\mathbf{x}_{N+1}\right) \leq 0, & &
\end{aligned} \tag{2.25}$$

where f is the system dynamics, g is general constraints of the problem, and $\mathbf{x}_{\text{lb}}, \mathbf{x}_{\text{ub}}, \mathbf{u}_{\text{lb}}, \mathbf{x}_{\text{ub}}$ are state and control ranges. MPC process performs the following steps:

1. Sense the current state $\mathbf{x}_c$ as initial state.
2. Compute a finite-horizon optimal trajectory $\mathbf{x}_{0:T}, \mathbf{u}_{0:T}$ with initial state $\mathbf{x}_0 = \mathbf{x}_c$.
3. Only execute the first control $\mathbf{u}_c = \mathbf{u}_{0:\Delta t}$ for $t \in [0, \Delta t)$.
4. Repeat from step 1.

Nonlinear programming methods such as direct collocation and interior point method (e.g. IPOPT solver [149]) are often used to solve MPC problems. As employed in practice, MPC is a moderately complex procedure, and from a theoretical perspective, it is difficult to analyze and prove stability and convergence properties. However, with careful tuning, MPC can be an extremely high-performing and practical nonlinear optimal control technique. There are several design decisions of note when implementing such an MPC controller: (1) The time step Δt must be long enough to allow the computation in step 2 to find an optimal trajectory. (2) The time horizon T used in step 2 is an important variable because it should be long enough for MPC to benefit from predictive lookahead, but not too long such that computation time exceeds Δt. (3) Model mismatch between the true system's dynamics and the model used in the optimizer will, in general, degrade MPC performance.

2.3 Koopman Operator Theory

The thesis explores using learning to enable scalable optimal control with interpretable learning outcomes. We leverage Koopman operator theory and its related methods to achieve this. The Koopman operator theory [64], first introduced in 1931, has recently emerged as a leading framework for obtaining linear representations of nonlinear dynamical systems from data. Its core idea is based on Koopman's finding that a nonlinear dynamical system may be represented by an infinite-dimensional linear operator acting on the space of

measurement functions of the system state. This ability to embed nonlinear dynamics in a linear framework is particularly promising, as powerful and comprehensive techniques for linear systems analysis and control can be used for the prediction, estimation, and control of nonlinear systems.

Consider a discrete-time dynamical system that allows for external inputs:

$$\mathbf{x}_{k+1} = \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right), \tag{2.26}$$

where $\mathbf{x} \in \mathbb{R}^{n_\mathbf{x}}$ and $\mathbf{u} \in \mathbb{R}^{n_\mathbf{u}}$. Define the Koopman embedding function $\mathbf{\Psi}(\mathbf{x_k}, \mathbf{u_k})$ where $\mathbf{\Psi} : \mathbb{R}^{n_\mathbf{x}} \times \mathbb{R}^{n_\mathbf{u}} \to \mathbb{R}^{n_\mathbf{\Psi}}$. The Koopman operator with control inputs $\mathcal{K}$ is given by:

$$\mathcal{K}\mathbf{\Psi}(\mathbf{x}, \mathbf{u}) \triangleq \mathbf{\Psi}(\mathbf{f}(\mathbf{x}, \mathbf{u}), *). \tag{2.27}$$

where $*$ indicates a choice of definition with $* = \mathbf{u}$ indicating the input is dynamically evolving while $* = \mathbf{0}$ indicating the input does not evolve but only affects the state evolution. In the thesis, we assume $* = \mathbf{0}$ and the Koopman operator is only attempting to propagate the observable functions at the current state and inputs to the future observable functions on the state, such that $\mathcal{K}\mathbf{\Psi}(\mathbf{x}_k, \mathbf{u}_k) \triangleq \mathbf{\Psi}(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \mathbf{0})$. In practice, a common choice of $\mathbf{\Psi}(\mathbf{x}, \mathbf{u})$ is to decouple its state and control component [124, 170]:

$$\mathbf{\Psi}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{\Psi}_\mathbf{x}(\mathbf{x}) & \mathbf{\Psi}_\mathbf{u}(\mathbf{u}) \end{bmatrix}^\top, \tag{2.28}$$

where $\mathbf{\Psi}_\mathbf{x}(\cdot)$ is a state-dependent function, $\mathbf{\Psi}_\mathbf{u}(\cdot)$ is a control-dependent function and in this work, we assume it is identically linear to control $\mathbf{\Psi}_\mathbf{u} = \mathbf{u}$. By merging Eq. (2.28), (2.27), (2.26) and letting latent state and control representation to be $\mathbf{z} = \mathbf{\Psi}_\mathbf{x}(\mathbf{x})$ and $\mathbf{v} = \mathbf{\Psi}_\mathbf{u}(\mathbf{u})$, we have a linear dynamical model in the latent space $\mathbf{z} \in \mathbb{R}^{n_\mathbf{z}}$ and $\mathbf{v} \in \mathbb{R}^{n_\mathbf{v}}$:

$$\mathbf{\Psi}(\mathbf{x}_{k+1}, \mathbf{0}) = \begin{bmatrix} \mathbf{A} & \mathbf{B} \end{bmatrix} \begin{bmatrix} \mathbf{\Psi}_\mathbf{x}(\mathbf{x}) & \mathbf{\Psi}_\mathbf{u}(\mathbf{u}) \end{bmatrix}^\top \implies \mathbf{z}_{k+1} = \mathbf{A}\mathbf{z}_k + \mathbf{B}\mathbf{v}_k, \tag{2.29}$$

where $\mathbf{K} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \end{bmatrix}$. Now Eq. (2.29) becomes our familiar linear system over the latent space that allows well-established linear control and analysis, despite the considered discrete-time dynamical system $\mathbf{f}$ is nonlinear. However, the Koopman operator requires the embedding function to be infinite-dimensional, which is not realistic from a practical standpoint. Therefore, the Koopman operator is usually approximated with a linear finite-dimensional embedding function in a data-driven way.

2.3.1 Classical Koopman Control

Extended Dynamical Mode Decomposition (EDMD) [158] is one of the representative classical methods for Koopman control. To approximate the Koopman operator, it chooses

embedding function $\mathbf{\Psi}(\mathbf{x}, \mathbf{u})$ as a set of user-defined scalar-valued functions:

$$\mathbf{\Psi}(\mathbf{x}, \mathbf{u}) = [\psi_1(\mathbf{x}, \mathbf{u}), \psi_2(\mathbf{x}, \mathbf{u}), \dots,]\,. \tag{2.30}$$

Next, the relation described by Eq. (2.27) is now given as

$$\mathbf{\Psi}\left(\mathbf{x}_{k+1}, \mathbf{u}_k\right) = \mathbf{K}\mathbf{\Psi}\left(\mathbf{x}_k, \mathbf{u}_k\right) + \mathbf{e}\left(\mathbf{x}_k, \mathbf{u}_k\right). \tag{2.31}$$

where $\mathbf{e}\left(\mathbf{x}_k, \mathbf{u}_k\right)$ is a residual (approximation error). Note that the matrix $\mathbf{K}$ advances $\mathbf{\Psi}$ forward one-time step. Assuming that the total number of trajectories of the system has been collected such that

$$X = [\mathbf{x}_1, \mathbf{u}_1, \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N, \mathbf{u}_N, \mathbf{x}_{N+1}]\,, \tag{2.32}$$

where N is the number of recorded data points. The matrix $\mathbf{K}$ can be computed in a least-squares approach, where $\mathbf{K}$ is determined by

$$\min \frac{1}{2} \sum_{i=1}^{N-1} \|\mathbf{e}\left(\mathbf{x}_i, \mathbf{u}_i\right)\|^2 = \min_{\mathbf{K}} \frac{1}{2} \sum_{i=1}^{N-1} \|\Psi\left(\mathbf{x}_{i+1}, \mathbf{u}_i\right) - \Psi\left(\mathbf{x}_i, \mathbf{u}_i\right)\mathbf{K}\|^2. \tag{2.33}$$

Solving the least-squares problem yields

$$\mathbf{K} = \mathbf{G}^{\dagger}\mathbf{M} \tag{2.34}$$

where $\dagger$ denotes the Moore-Penrose pseudoinverse and

$$\mathbf{G} = \frac{1}{N} \sum_{i=1}^{N-1} \Psi\left(\mathbf{x}_i, \mathbf{u}_i\right)^{\top} \Psi\left(\mathbf{x}_i, \mathbf{u}_i\right),$$
$$\mathbf{M} = \frac{1}{N} \sum_{i=1}^{N-1} \Psi\left(\mathbf{x}_i, \mathbf{u}_i\right)^{\top} \Psi\left(\mathbf{x}_{i+1}, \mathbf{u}_i\right).$$

Note that the computational burden of this approach grows as the dimension of Ψ increases. The approach generally yields a better approximation as the dimension of Ψ increases. Furthermore, the number of data points and their distribution across the state space will have a large effect on the computed $\mathbf{K}$ matrix. The recorded data points need not come from a single trajectory nor be sequential [157]. Multiple trajectories and trajectories with missing data points can be used. The only requirement is the sum of residuals given in Eq. (2.31) be defined by consecutive states $(\mathbf{x}_k, \mathbf{x}_{k+1})$ spaced equally in time.

2.3.2 Deep Learning-Based Koopman Control

The embedding function $\mathbf{\Psi}(\cdot)$ provides intrinsic coordinates that globally linearize nonlinear dynamics. Thus, finding an appropriate embedding function is the key to Koopman control.

Despite the immense promise of Koopman embeddings, Obtaining $\mathbf{\Psi}(\cdot)$ has proven difficult in all but the simplest systems as it is usually unknown and has to be summarized from data. Classical methods such as EDMD require manual effort to select a set of nonlinear basis functions, but often less accurate and scalable to complex systems. As the prevalence of deep learning, the use of multi-layer neural networks is capable of representing any arbitrary function, including the desired Koopman embedding function. Therefore, deep learning-based approaches have recently been widely applied to find Koopman embeddings and operators for control. Koopman embedding function $\mathbf{\Psi}$ and operator $\mathbf{K}$ are two components represented by neural networks (NNs). Following Eq. (2.28), we can have

$$\mathbf{\Psi}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{\Psi}_\psi(\mathbf{x}) & \mathbf{\Psi}_\varphi(\mathbf{u}) \end{bmatrix}^\top, \tag{2.35}$$

where ψ and φ are NNs parameters. In practice, multi-layer feed-forward networks are often used. The objective is to identify key intrinsic latent embedding $\mathbf{\Psi}_\psi$ and $\mathbf{\Psi}_\varphi$ for state and control, along with dynamical system coefficients $\mathbf{K}$. Normally, there are three high-level requirements for the network, which correspond to three types of loss functions used in the network training. Here, we assume $\mathbf{\Psi}_\varphi(\mathbf{u}) = \mathbf{u}$.

- Intrinsic latent states that are useful for reconstruction. We seek to identify a few latent states $\mathbf{z} = \psi(\mathbf{x})$ where the dynamics evolve, along with an inverse $\mathbf{x} = \psi^{-1}(\mathbf{z})$ so that the state x may be recovered. This can be achieved using an autoencoder structure [6], where ψ is the encoder and ψ^{-1} is the decoder. The dimension $\mathrm{Dim}(\mathbf{z})$ of the auto-encoder subspace is a hyperparameter of the network, and this choice may be guided by knowledge of the system. Reconstruction accuracy of the auto-encoder is achieved using the following loss: $\left\|\mathbf{x} - \psi^{-1}(\psi(\mathbf{x}))\right\|^2$.

- Linear dynamics. To discover the Koopman embedding function $\mathbf{\Psi}(\cdot)$, we learn the linear dynamics $\mathbf{K} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \end{bmatrix}$ on the latent states, i.e., $\mathbf{z}_{k+1} = \mathbf{A}\mathbf{z}_k + \mathbf{B}\mathbf{u}_k$. Linear dynamics are achieved using the following loss: $\left\|\psi\left(\mathbf{x}_{k+1}\right) - \mathbf{A}\psi\left(\mathbf{x}_k\right) - \mathbf{B}\mathbf{u}_k\right\|^2$. More generally, the linear prediction can be enforced over m time steps.

- Future state prediction. Finally, the intrinsic latent states must enable future state prediction. Specifically, we identify linear dynamics in the matrix $\mathbf{K}$. This corresponds to the loss $\left\|\mathbf{x}_{k+1} - \psi^{-1}\left(\mathbf{A}\psi\left(\mathbf{x}_k\right) + \mathbf{B}\mathbf{u}_k\right)\right\|$.

After identifying the latent linear dynamics for the considered nonlinear system, any classical control methods such as LQR and MPC can be subsequently used to derive the control over the latent space.

2.4 Multi-Agent Hierarchical RL

The last work of the thesis investigates the efficiency of multi-agent centralized learning over higher-level options. In this section, we briefly describe the related basics of multi-agent learning and options framework.

2.4.1 Multi-Agent RL

RL can be extended to involve multiple robots. Specifically, multi-agent RL (MARL) addresses the sequential decision-making problem of multiple autonomous agents that operate in a common environment, each of which aims to optimize its long-term return by interacting with the environment and other agents [14, 172]. Unlike single-agent RL, MARL requires that the system state evolution and the reward received by each agent are influenced by the joint actions of all agents. Many important real-world tasks are multi-agent by nature, such as taxi coordination [81], and supply chain management [41].

Based on the MDPs of the single-agent case, one direct generalization of MDP that captures the intertwinement of multiple agents is Markov games (MGs) [123]. Originating from the seminal work [82], the framework of MGs has long been used in the literature to develop MARL algorithms. Formally. a Markov game is defined by a tuple $(N, \mathcal{S}, \{\mathcal{A}^i\}, p, \{r^i\}, \gamma)$, where $i = 1, 2, ..., N$ and N is the number of agents. At time step k, each agent i executes an action $\mathbf{a}_k^i$ from its action space $\mathcal{A}^i$, according to the system state $\mathbf{s}_k$ belonging to the state space $\mathcal{S}$. The system then transitions to state $\mathbf{s}_{k+1}$ following the system transition probability $p(\mathbf{s}_{k+1} \mid \mathbf{s}_k, \mathbf{a}_k)$, where $\mathbf{a}_k := \mathbf{a}_k^1 \times \cdots \times \mathbf{a}_k^N$. Then each agent receives individual reward $r^i(\mathbf{s}_k, \mathbf{a}_k, \mathbf{s}_{k+1})$. The goal of agent i is to optimize its long-term reward, by finding the policy $\pi^i : \mathcal{S} \to \mathcal{A}^i$ such that $\mathbf{a}_k^i \sim \pi^i(\cdot \mid \mathbf{s}_k)$. Correspondingly, the value-function $V^i : \mathcal{S} \to \mathbb{R}$ of agent i is defined as follows,

$$V^i(s) := \mathbb{E}\left[\sum_{k\geq 0} \gamma^k r^i(\mathbf{s}_k, \mathbf{a}_k, \mathbf{s}_{k+1}) \mid \mathbf{a}_k^i \sim \pi^i(\cdot \mid \mathbf{s}_k), \mathbf{s}_0 = \mathbf{s}\right] \tag{2.36}$$

Depending on the relationship and objective of agents, MARL methods can be fully cooperative, fully competitive, and a mix of the two. In this thesis, we mainly discuss the cooperative setting, where agents collaborate to optimize a common long-term return. Two common agent types, that is heterogeneous and homogeneous, are often used in cooperative MARL. Homogeneous agents affiliated with the environment hold the same policy. The policy gives out different actions based on the agent's observation. Heterogeneous agents require each to maintain its policy, which can accept different environment observation dimensions or output actions with diverse dimensions. Methods for solving a MARL problem can be categorized by their learning styles:

- Independent Learning [137]. The key idea is to establish an independent policy for each agent from the multi-agent system and train it using RL, ignoring other agents and system states. Based on this idea, if every agent learns its policy independently, we can obtain a set of policies that jointly solve the task. However, it can suffer from instability arising from the non-stationarity of the environment induced by simultaneously learning and exploring agents.

- Centralized Training Decentralized Execution (CTDE): In this setting, agents are trained together in a centralized manner where they can access all available information, including the global state, other agents' status, and rewards. However, during the execution stage, agents are forced to make decisions based on their local observations, without access to centralized information or communication. Centralized critic [171, 37] and value decomposition [108, 129] are two main techniques used in CTDE.

- Fully Centralized: In this class, all agents and their action spaces are combined into one, and a standard RL pipeline is used to update a joint policy or Q-value function. For instance, a five-agent discrete control problem can be transformed into a single-agent multi-discrete control problem.

2.4.2 Options Framework

We now describe important concepts related to hierarchical reinforcement learning (HRL), especially the Options Framework [134], which defines a two-level hierarchy, and introduces options as temporally extended actions. The term "options" is meant as a generalization of primitive "actions" used in standard RL. In contrast to a primitive action that is often executed over one time step, the execution of an option typically requires multiple time steps. We now define options under the single-agent setup. $\mathbf{o}$ are defined as triples $\langle \mathcal{I}^{\mathbf{o}}, \beta^{\mathbf{o}}, \pi^{\mathbf{o}} \rangle$, where $\mathcal{I}^{\mathbf{o}} \subseteq \mathcal{S}$ is the initiation state set and $\beta^{\mathbf{o}} : \mathcal{S} \to [0, 1]$ is the (stochastic) option termination condition. $\pi^{\mathbf{o}}(\mathbf{a} \mid \mathbf{s}) : \mathcal{S} \to \mathcal{A}$ is a deterministic option policy that selects primitive actions to achieve the target of the option. A higher level policy $\mu(\mathbf{o} \mid \mathbf{s})$ determines which option to select at state $\mathbf{s}$. At state $\mathbf{s}_k \in \mathcal{I}^{\mathbf{o}}$, μ selects option $\mathbf{o}_k$, then the next action $\mathbf{a}_k$ according to $\pi^{\mathbf{o}}$ is executed, and the state transition to $\mathbf{s}_{k+1}$ to see if the option is terminated or continues the $\mathbf{a}_{k+1}$. On reaching the termination condition in state $\mathbf{s}_T$, an agent can select a new option based on μ from the set $\mathcal{O}(\mathbf{s}_T) := \{\mathbf{o} \mid \mathbf{s}_T \in \mathcal{I}^{\mathbf{o}}\}$.

The Bellman equation can be generalized to option framework [134]. For any Markov policy $\mu(\mathbf{o} \mid \mathbf{s})$, the state-value function can be written

$$\begin{aligned} V^{\mu}(\mathbf{s}) &= \mathbb{E}\left\{ r_{k+1} + \cdots + \gamma^{\zeta} r_{k+\zeta} + \gamma^{\zeta} V^{\mu}(\mathbf{s}_{k+\zeta}) \mid \mathbb{I}(\mu, \mathbf{s}, k) \right\} \\ &= \sum_{\mathbf{o} \in \mathcal{O}_{\mathbf{s}}} \mu(\mathbf{o} \mid \mathbf{s}) \left[r(\mathbf{s}, \mathbf{o}, \mathbf{s}') + \sum_{\mathbf{s}'} p(\mathbf{s}' \mid \mathbf{s}, \mathbf{o}) V^{\mu}(\mathbf{s}') \right] \end{aligned} \tag{2.37}$$

where ζ is the duration of the first option selected by $\pi(\mathbf{o} \mid \mathbf{s})$, $\mathbb{I}(o, s, k)$ denotes the event of $\mathbf{o}$ being initiated in state $\mathbf{s}$ at time k. Similar generalizations can be done for optimal state/state-action value functions. The objective of option-based RL is to find μ that maximizes the long-term expected return over the choices of options. In practice, options can be given as fixed temporal action sequences by a system designer, or they can be also learned within the RL task.

2.4.3 Multi-Agent Centralized Option Selection

The options framework was originally discussed for single-agent cases but can be used for multi-agent scenarios. Due to options' nature of varying time length, when determined and executed in a *centralized* way by multiple agents, options have to be organized with termination strategies to be synchronized. This is because the centralized controller makes decisions for all agents at the same time step [165, 164]. Typically, three termination strategies $\eta_{\text{any}}, \eta_{\text{all}}$, and η_{continue} were introduced and analyzed in [32]. In η_{any} termination scheme, When any one agent completes an action, all other agents interrupt their options, and the next decision epoch occurs when a new joint option needs to be selected. In η_{all} scheme, When an agent completes an option, it waits (takes the idle action) until all the other agents finish their current options. Then, the next decision epoch occurs and all the agents choose the next joint option together. Both termination strategies require all the agents to choose their options at every decision epoch *synchronously*. In η_{continue} termination scheme, the other agents whose activities have not terminated are not interrupted, and only those agents whose options have terminated select new options. In this termination strategy, only a subset of the agents choose new options at each decision epoch, given the options being performed by the other agents.

The three termination strategies can be used in centralized multi-agent learning over options. Synchronous schemes such as η_{any} and η_{all} can be directly used but either break the completeness of options or cause inefficient option execution due to a subset of agents waiting for the others. Asynchronous strategy η_{continue} is ideal as it allows agents to execute decisions independently even under a centralized learning setup. However, it is still unclear how to apply asynchronous terminations in centralized scenarios for multi-agent RL.

Chapter 3

Data-Efficiency and Exploration via Control-Based Value Functions

This chapter is based on my papers "TTR-Based Reward for Reinforcement Learning with Implicit Model Priors" [87][1], and "Optimal Control-Based Baseline for Guided Exploration in Policy Gradient Method" (originated from [89][2], revised and resubmitted to ICRA23) written in collaboration with Site Li, Seth Siriya, Ye Pu, and Mo Chen.

3.1 Chapter Overview and Related Work

Sequential decision-making is a fundamental problem faced by any autonomous agent interacting extensively with the environment [83]. RL and optimal control are two essential tools for solving such problems. RL trains an agent to choose actions that maximize its long-term accumulated reward through trial and error and can be divided into model-free and model-based variants [103]. Optimal control, on the other hand, assumes the perfect knowledge of system dynamics and produces control policy through analytical computation.

Model-free RL has been successful in many fields such as games and robotics [121, 118, 78, 132, 116, 74], and allows control policies to be learned directly from high-dimensional inputs by mapping observations to actions. Despite such advantages, model-free methods often require an impractically large number of trials to learn desired behaviors. Data inefficiency is a fundamental barrier impeding the adoption of model-free algorithms in real-world settings, especially in the context of robotics [99, 45, 97].

To address the problem of data inefficiency in model-free RL, various techniques have been proposed. "Deep exploration" [99] samples actions from randomized value function in order to induce exploration in the long term. Count-based exploration [138] extends near-optimal algorithms into high-dimensional state space. On the other hand, several recent papers refactor the structure of RL in order to utilize data more efficiently [146, 147, 97, 12, 36].

[1]Supplementary contents: https://sites.google.com/view/ttrpapersupp
[2]Supplementary contents: https://sites.google.com/view/mbbpapersupp

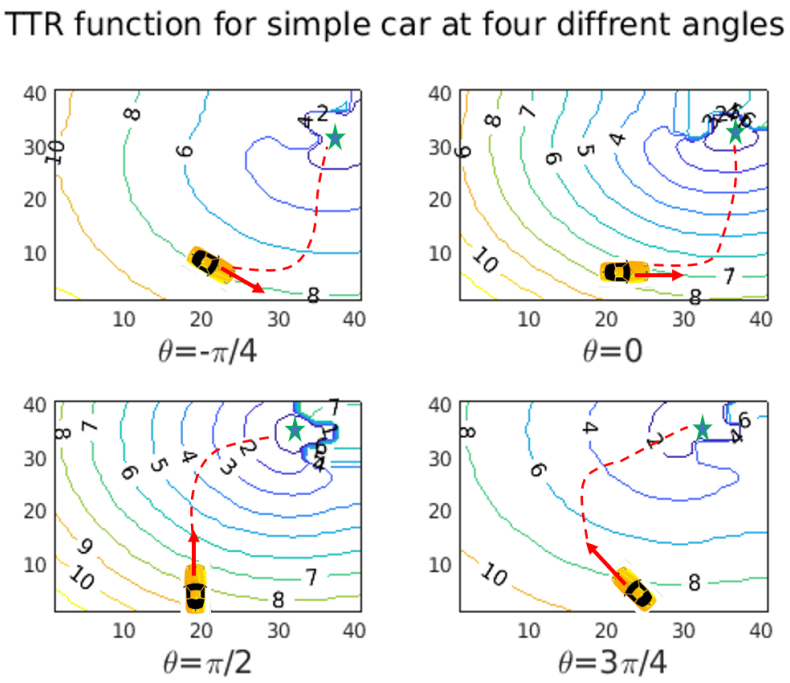


Figure 3.1: TTR functions at different heading angles for a simple car model. The TTR function describes the minimum arrival time under assumed system dynamics and is effectively used for reward shaping in robotic RL tasks.

In particular, curriculum-based approaches [12, 36] learn progressively over multiple subtasks where the initial task is used to guide the learner so that it will perform better on the final task. Hierarchical Reinforcement Learning (HRL) [146, 147, 97] involves decomposing problems into a hierarchy of sub-problems or sub-tasks such that higher-level parent tasks invoke lower-level child tasks as if they were primitive actions.

Model-based RL uses an internal model (given or learned) that approximates the full system dynamics [5, 1, 32]. A control policy is learned based on this model. This significantly reduces the number of trials in learning and leads to fast convergence. However, model-based methods are heavily dependent on the accuracy of the model itself, thus the learning performance can be easily affected by the model bias. This is challenging especially when one aims to map sensor inputs directly to control actions, since the evolution of sensor inputs over time can be very difficult to model.

Optimal control is a formal mathematical optimization technique that utilizes analytical or numerical methods to solve well-defined objectives and adhere to system constraints. It is considered "classical" as it has been substantially applied to control many autonomous systems for decades. For example, the authors in [80] apply optimal control on a two-joint robot manipulator in order to find a robust control strategy. The authors in [152] realize the real-time stabilization for a falling humanoid robot by solving a simplified optimal control

problem. In addition, there are numerous other applications in mobile robotics and aerospace [25, 21, 17, 22]. In general, optimal control does not require any data to generate an optimal solution if the system model is known, but cannot scale to models with high-dimensional state space due to modeling difficulty and computational complexity.

In this chapter, we propose methods that improve learning efficiency and exploration from the control-theoretic perspective. In Sec. 3.2, a Time-To-Reach (TTR) reward shaping approach is proposed to integrate optimal control into model-free RL to reduce the sample complexity of learning. This is accomplished by incorporating into the RL algorithm a TTR-based reward function, which is obtained by solving a Hamilton-Jacobi (HJ) partial differential equation (PDE), a technique that originated in optimal control. A TTR function maps a robot's internal state to the minimum arrival time to the goal, assuming a model of the robot's dynamics. In the context of reward function in RL, intuitively a smaller TTR value indicates a desirable state for many goal-oriented robotic problems.

To accommodate the computational intractability of computing the TTR function for a high-dimensional system such as the one used in the RL problem, an approximate, low-dimensional system model that still captures key dynamic behaviors is selected for the TTR function computation. As we will demonstrate, such an approximate system model is sufficient for improving the data efficiency of policy learning. Therefore, our method avoids the shortcomings of both model-free RL and optimal control. Unlike model-based RL, our method does not try to learn and use a full model explicitly. Instead, we maintain a looser connection between a known model and the policy improvement process in the form of a TTR reward function. This allows the policy improvement process to take advantage of model information while remaining robust to model bias.

Our approach can be modularly incorporated into any model-free RL algorithm. In particular, by effectively infusing system dynamics in an implicit and compatible manner with RL, we retain the ability to learn policies that map sensor inputs directly to actions. Our approach represents a bridge between traditional analytical optimal control approaches and the modern data-driven RL and inherits the benefits of both. We evaluate our approach on two common mobile robotic tasks and obtain significant improvements in learning performance and efficiency. We choose Proximal Policy Optimization (PPO) [118], Trust Region Policy Optimization (TRPO) [121], and Deep Deterministic Policy Gradient (DDPG) [78] as three representative model-free algorithms to illustrate its modularity and compatibility.

3.2 Time-To-Reach Value Function as Learning Reward

3.2.1 Preliminary

We briefly introduce the concepts of approximate system models and the time-to-reach function. Consider the following dynamical system in $\mathbb{R}^n$,

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \tag{3.1}$$

where $\mathbf{f}(\cdot)$ is the system dynamics of a TTR problem that can be extracted from a given RL problem. We call it *approximate* as it is usually a simplification of the true RL MDP model $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$. Here, $\mathbf{x}(t)$ and $\mathbf{u}(t)$ are the state and control of this approximate system model. The TTR problem involves finding the minimum time it takes to reach a goal from any initial state $\boldsymbol{x}$, subject to the system dynamics in Eq. (3.1). We assume that $\mathbf{f}(\cdot)$ is Lipschitz continuous. Under these assumptions, the dynamical system has a unique solution. The common approach for tackling TTR problems is to solve a Hamilton-Jacobi (HJ) partial differential equation (PDE) corresponding to system dynamics and this approach is widely applicable to both continuous and hybrid systems [175, 168, 136]. Mathematically, the time it takes to reach a goal $\Gamma \in \mathbb{R}^n$ using a control policy $\mathbf{u}(\cdot)$ is

$$\mathcal{T}_{\boldsymbol{x}}[\mathbf{u}] = \min\{t | \mathbf{x}(t) \in \Gamma\}, \tag{3.2}$$

and the TTR function with only one player taken into account is defined by

$$\phi(\mathbf{x}) = \min_{\mathbf{u}} \mathcal{T}_{\boldsymbol{x}}[\mathbf{u}]. \tag{3.3}$$

Through dynamic programming, we can obtain ϕ by solving the following HJ PDE:

$$\max_{\mathbf{u}}\{-\nabla\phi(\mathbf{x})^\top \mathbf{f}(\mathbf{x}, \mathbf{u}) - 1\} = 0 \tag{3.4}$$

$$\phi(\mathbf{x}) = 0 \ \forall \mathbf{x} \in \Gamma \tag{3.5}$$

Detailed derivations and discussions are presented in [8, 10]. Normally the computational cost of solving the TTR problem is too expensive for systems with higher than five dimensional state. However, model simplification and system decomposition techniques partially alleviate the computational burden in a variety of problem setups [93, 15]. Well-studied level set-based numerical techniques [168, 136, 93, 15] have been developed to solve Eq. (3.4).

3.2.2 Method

Model-free RL algorithms have the benefit of being able to learn control policies directly from high-dimensional state and observation; however, the lack of data efficiency is a well-known challenge. In this work, we alleviate this issue by implicitly utilizing a simplified

system model to provide a useful "model-informed" reward in an important subspace of the full MDP state. This way we produce policies that are as flexible as those obtained from model-free RL algorithms, and accelerate learning without altering the model-free pattern.

In this section, we explain the concrete steps of applying our method, as shown in Fig. 3.2. The RL problem under consideration is represented by an MDP given by $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$, as explained in Section 2.1.1; this MDP is in general unknown. Choosing an approximate system model $\mathbf{f}(\cdot)$ that captures key dynamic behavior is the first step; this step is explained in Section 3.2.2. Using this approximate system model, we compute the TTR function $\phi(\cdot)$, and then apply a simple transformation to it to obtain the reward function $r(\cdot)$ that is used in RL; this is fully discussed in Section 3.2.2. Finally, any model-free RL algorithm may be used to obtain a policy that maximizes the expected return.

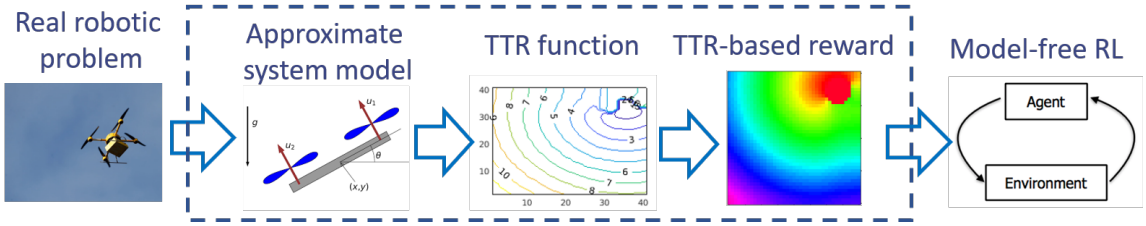


Figure 3.2: Overview of TTR-based reward shaping method

Model Selection

"Model selection"[3] here refers to the fact that we need to pick an (approximate) system model for the robotic task to compute the corresponding TTR function. This model should be relatively low-dimensional so that the TTR computation is tractable but still retains key behaviors in the dynamics of the system. Before the detailed description of model selection, it is necessary to clarify some terminology used in this section. First, we use the phrase "full MDP model" to refer to $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$, which drives the true state transitions in the RL problem. The full MDP model is often inaccessible since it is an underlying model behind the simulator or real world and captures the high-dimensional state inputs including both sensor data and the robot's internal state. Second, we will use the phrase "approximate system model" to refer to $\mathbf{f}(\cdot)$. It does not necessarily accurately reflect the real state transitions of the problem we are solving. The approximate system model should be low-dimensional to simplify the TTR computation while still capturing key robot physical dynamics.

The relation between the full MDP model and the approximate system model is formalized by their state spaces, as shown in Eq. (3.6), where the approximate system state $\mathbf{x}$ is a projection onto an axis-aligned subspace of the full MDP state $\mathbf{s}$. The projection may have various forms but in this work, we assume a simple case where $\mathbf{x}$ includes a subset of state

[3]Note that "model selection" here has a different meaning from machine learning validation.

components of $\mathbf{s}$.

$$\mathbf{s} = (\mathbf{x}, \hat{\mathbf{x}}) \tag{3.6}$$

Here the full state $\mathbf{s}$ refers to the entire high-dimensional state in the full MDP model $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$, and $\mathbf{x}$ refers to the state of the approximate system model $\mathbf{f}(\cdot)$ which evolves according to Eq. (3.1). For clarity, we also define $\hat{\mathbf{x}}$, which are state components in the full MDP model but are not part of $\mathbf{x}$. For example, in the simulated car experiment, the full state $\mathbf{s}$ contains the internal states of the car, including the position (x, y), heading θ, speed v, and turn rate ω. In addition, eight laser range measurements $d_1, \ldots, d_8$ are also part of the state s. These measurements provide distances from nearby obstacles. As one can imagine, the evolution of $\mathbf{s}$ can be very difficult if $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ is impossible to obtain, especially in a *priori* unknown environments. The state of the approximate system, denoted $\mathbf{x}$, contains a subset of the internal states $(x, y, \theta, v, \omega)$, and evolves according to Eq. (3.1). In particular, for the simulated results in this work, we choose the Dubins Car model to be the approximate system dynamics:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{3.7}$$

As we show in the simple car example, such simple dynamics are sufficient for improving the data efficiency of model-free RL. With this choice, the remaining states are denoted $\hat{\mathbf{x}} = (v, \omega, d_1, \ldots, d_8)$. Fig. 3.3 illustrates this example. Note that in general, the control $\mathbf{a}$ and $\mathbf{u}$ of the two systems do not have to be the same: $\mathbf{u}$ could be a projection of $\mathbf{a}$.

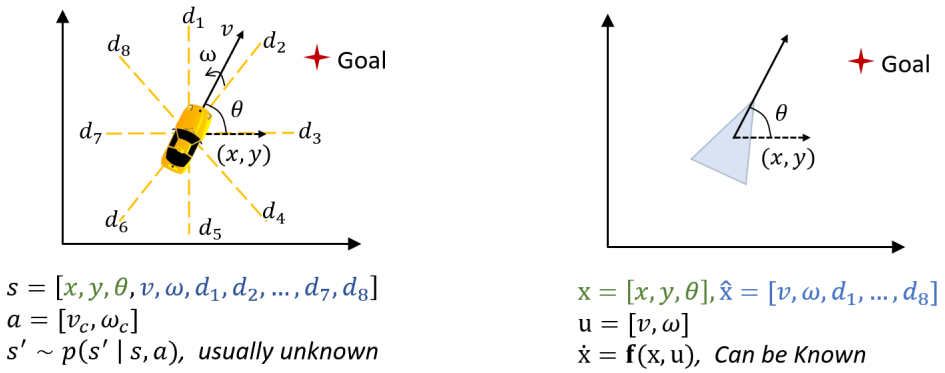


Figure 3.3: State definition of a simulated car. **Left**: full RL state with true MDP model. **Right**: 3D lo-dim state of the approximate model.

How does one choose a good approximate model $\mathbf{f}(\cdot)$? It is important to note that $\mathbf{f}(\cdot)$ needs not to be directly derived from the target RL full MDP model $p(\cdot)$. In addition, $f(\cdot)$ and $p(\cdot)$ do not necessarily need to have rigorous mathematical relations. In fact, $\mathbf{f}(\cdot)$ can be empirically selected from the available robot system models in classical control

Robot Type	Robot System Model
Car-like system	3D Dubins car [33] 5D extended Dubins car [112] 4D bicycle model [38]
Quadrotor	6D planar quadrotor [87] 6D rotation model [135] 3D point mass model [110]

Table 3.1: A list of available robot system models. A 3D Dubins car has three states – position (x, y) and its heading angle (θ), describing the dynamics of a simple differential-wheeled robot. Its 5D extended version involves speed v and turn rate ω as additional states. A 4D bicycle model describes the motion of a four-wheeled vehicle with positions, heading and specifically steering angle. For quadrotors, the 6D planar model focuses on 2D plane motion with states of positions, velocities, and 1-axis rotation. The 6D rotation model focuses on the 3-axis rotation without translation while a simple point-mass model focuses solely on 3D translation without any rotational dynamics.

theory, where various useful mathematical models are already well-defined and widely used, as shown in Table 3.1. We identify a proper $\mathbf{f}(\cdot)$ according to the robot type and state space of the RL problem on hand. Despite being abstracted and simplified from true dynamics, $\mathbf{f}(\cdot)$ usually allows efficient optimal control computation and the associated solution (e.g. TTR value function) can be useful in mitigating the sample inefficiency of pure RL search and exploration.

In general, we may choose $\mathbf{x}$ such that a reasonable explicit, closed-form ODE model $\mathbf{f}(\cdot)$ can be derived. Such a model should capture the evolution of the robotic internal state. One motivation for using an ODE is that the real system operates in continuous time, and computing the TTR function for continuous-time systems is a solved problem for sufficiently low-dimensional systems. Note that if a higher-fidelity model of the car is desired, one may also choose the following 5D ODE approximate system model instead:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \\ \alpha_v \\ \alpha_\omega \end{bmatrix} \tag{3.8}$$

In this case, we would have $\mathbf{x} = (x, y, \theta, v, \omega)$, and $\hat{\mathbf{x}} = (d_1, \ldots, d_8)$. Note that the choice of an ODE model representing the real robot may be very flexible, depending on what behavior one wishes to capture. In the 3D car example given in Eq. (3.7), we focus on modeling the position and heading of the car to be consistent with the goal. However, if speed and angular speed are deemed crucial for the task under consideration, one may also choose a more complex approximate system given by Eq. (3.8). To re-iterate, a good choice

of approximate model is computationally tractable for the TTR function computation and captures the system behaviors that are important for performing the desired task.

TTR Function as Approximate Reward

In this section, we discuss how the reward function $r(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ of the given RL problem can be shaped based on the TTR function. For simplicity, we ignore $\mathbf{a}$ and denote it as $r(\mathbf{s})$, although a simple modification to the TTR function can be made to incorporate actions into the reward function. Since $\mathbf{s}$ is often in high dimensional space with sensor measurements involved, it is often unclear how to determine a proper reward for $\mathbf{s}$. As a result, simple reward functions such as sparse and distance rewards are sometimes used.

However, this can be easily resolved in our approach by viewing $r(\cdot)$ as a function of $\mathbf{x}$, the state of the approximate system model we have chosen before. As mentioned earlier, $\mathbf{x}$ is a subset of full state $\mathbf{s}$. In our method, the TTR function $\phi(\mathbf{x})$ defined in Eq. (3.3) is transformed slightly to obtain the reward function for full MDP state $\mathbf{s}$:

$$r(\mathbf{s}) = r(\mathbf{x}, \hat{\mathbf{x}}) = \begin{cases} -\phi(\mathbf{x}) & \mathbf{s} \in \mathbf{I} \\ 1000 & \mathbf{s} \in \mathbf{G} \\ -400 & \mathbf{s} \in \mathbf{C} \end{cases} \tag{3.9}$$

By definition, $\phi(\mathbf{x})$ is non-negative and $\phi(\mathbf{x}) = 0$ if and only if $\mathbf{x} \in \Gamma$. Thus, we use $-\phi(\cdot)$ as the reward because the state with a lower goal-arrival time should be given a higher reward.

As shown in Eq. (3.9), we set positive reward for goal states $\mathbf{G}$ and negative reward for collision states $\mathbf{C}$. Note that the TTR-based reward can also be extended to have obstacles taken into account, or to satisfy any other design choices if required. For *intermediate* states that are neither obstacles nor goals, TTR function $\phi(\cdot)$ directly provides a useful reward signal in an important subspace of the full MDP state. This is significant since the associated rewards for these intermediate states are usually quite difficult to manually design, and the TTR reward does not require any manual fine-tuning. This way the RL agent learns faster compared to not having a useful reward in the subspace, and can quickly learn to generalize the subspace knowledge to the high-dimensional observations. For example, positions that are near obstacles correspond to small values in LiDAR readings, and thus the agent would quickly learn these observations correspond to bad states.

To further reduce the computational complexity of solving the PDE for more complicated system dynamics (e.g. a quadrotor), we may apply system decomposition methods established from the optimal control community [93, 15, 16] to obtain an approximate TTR function without significantly impacting the overall policy training time. Particularly, we first decompose the entire system into several sub-systems potentially with overlapping components of state variables, and then efficiently compute the TTR for each sub-system. We utilize Lax-Friedrichs sweeping-based [168] to compute the TTR function. As shown in

Table 3.2, the computation time of TTR functions is negligible compared to the time it takes to train policies.

3.2.3 Results

In order to illustrate the benefits of our TTR-based reward shaping method, we now present two goal-oriented tasks through two different mobile robotic systems: a simple car and a planar quadrotor. Each system is simulated in Gazebo [61], an open-source 3D physical robot simulator. Also, we utilize the Robot Operating System (ROS) for communication management between the robot and the simulator. For each task, we compare the performance between our TTR-based reward and two other conventional rewards: sparse and distance-based rewards. For each reward function, we use three representative model-free RL algorithms (DDPG, TRPO, and PPO) to demonstrate that TTR-based reward can be applied to augment any model-free RL algorithm.

We select sparse and distance-based rewards for comparison with our proposed TTR-based reward because they are simple, easy to interpret, and easy to apply to any RL problem. These reward functions are consistent across our two simulated environments, shown in Table 3.3. We formulate the distance-based reward as general Euclidean distance involving position and angle because the tasks we consider involve reaching some desired set of positions and angles, shown in Table 3.3. By choosing different weights λ, the angle is assigned different weights. For both examples, we choose four different weights, $\lambda \in \{0, 0.1, 1, 10\}$. The sparse reward is defined to be 0 everywhere except for goal states (1000 reward) or collision states (-400 reward).

Task	Model	Computation Time	Decomposed
Simple Car	Eq. (3.7)	5 sec	No
Planar Quadrotor	Eq. (3.10)	90 sec	Yes

Table 3.2: TTR function computational load. 'Decomposed' means we need to decompose the approximate model into subsystems in order to reduce computational cost.

Simple Car Example

The car model is widely used as a standard testbed in motion planning [61] and RL [155] tasks. Here we use a "turtlebot-2" ground robot to illustrate the performance of the TTR-based reward. The state and observation of this example are already discussed at 3.2.2. The car starts with randomly sampled initial conditions from the starting area and aims to reach the goal region without colliding with any obstacle along the trajectory. Specifically, we set goal states as $\mathbf{G} = \{(x, y, \theta) | 3.7\,\text{m} \leq x \leq 4.3\,\text{m}; 3.7\,\text{m} \leq y \leq 4.3\,\text{m}; 0.45\,\text{rad} \leq \theta \leq 1.05\,\text{rad}\}$. The TTR-based reward for this simple car task is derived from a lower-

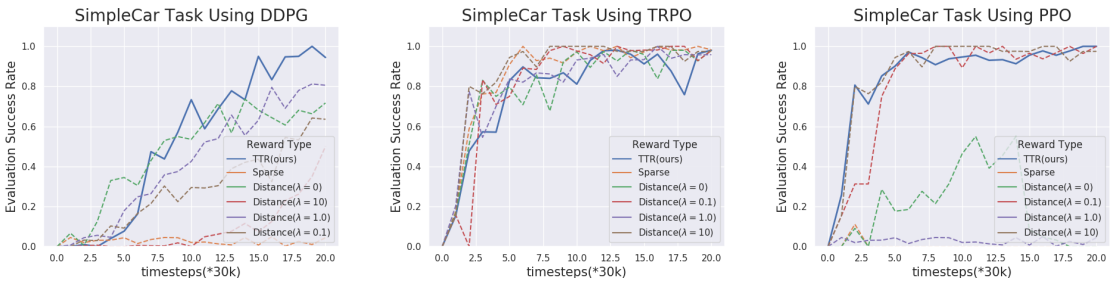


Figure 3.4: Performance comparison of three different reward functions on the car example under three model-free RL optimization algorithms: DDPG, TRPO, and PPO. All results are based on the mean of five runs.

dimensional approximate car system in Eq. (3.7) which only considers the 3D vector (x, y, θ) as state and angular velocity ω as control.

Fig. 3.4 compares the performance of TTR-based reward with sparse and distance-based rewards under three different model-free algorithms. The success rate after every fixed number of training episodes is considered a qualitative assessment. In general, the car system is simpler and more stable and thus obtains a relatively higher success rate among different reward settings compared to the quadrotor task (shown later in Fig. 3.6). In particular, TTR-based reward leads to a high success rate (mostly over 90%) consistently with all three learning algorithms. In contrast, sparse reward leads to poor performance with PPO, and distance-based reward leads to poor performance with DDPG.

Note that despite the better performance from certain distance-based rewards, the choice of appropriate weight for each variable is non-trivial and not transferable between different tasks. However, TTR-based reward requires little human engineering to design and can be efficiently computed once a low-fidelity model is provided.

The TTR function for the model in Eq. (3.7) is shown in Fig. 3.1 to convey the usefulness of TTR-based reward more intuitively. Here, we show the 2D slices of the TTR function at four heading angles, $\theta \in \{-\pi/4, 0, \pi/2, 3\pi/4\}$. The green star located at the upper-right of each plot is the goal area. The car starts moving from the lower-middle area. Note that the 2D slices look different for different heading angles according to the system dynamics, with the contours expanding roughly in the opposite direction to the heading slice.

Planar Quadrotor Example

A Quadrotor is usually considered difficult to control mainly because of its nonlinear and under-actuated dynamics. In the second experiment, we select a planar quadrotor model [42, 125], a popular test subject in the control literature, as a relatively complex mobile robot to validate that the TTR-based reward shaping method still works well even on the

Sparse	Distance	TTR
$r(\mathbf{s}) = \begin{cases} 0 \\ 1000 \\ -400 \end{cases}$	$r(\mathbf{s}) = \begin{cases} -d(\cdot)^* \\ 1000 \\ -400 \end{cases}$	$r(\mathbf{s}) = \begin{cases} -\phi(\mathbf{x})^* & \mathbf{s} \in \mathbf{I} \\ 1000 & \mathbf{s} \in \mathbf{G} \\ -400 & \mathbf{s} \in \mathbf{C} \end{cases}$
$^*d(\cdot) = \begin{cases} \sqrt{(x-x_g)^2 + (y-y_g)^2 + \lambda(\theta-\theta_g)^2} & \textbf{Simple Car} \\ \sqrt{(x-x_g)^2 + (z-z_g)^2 + \lambda(\psi-\psi_g)^2} & \textbf{Planar Quadrotor} \end{cases}$		
$^*\phi(\mathbf{x})$: TTR function defined in a subspace of $\mathbf{s}$		

Table 3.3: Reward functions tested in this work. **I**: set of intermediate states; **G**: set of goal states; **C**: set of collision states. $d(\cdot)$: generalized distance function involving angle.

highly dynamic and unstable system. "Planar" here means the quadrotor only flies in the vertical (x-z) plane by changing the pitch angle without affecting the roll and yaw angle.

The approximate system model has 6D internal state $\mathbf{x} = (x, v_x, z, v_z, \psi, \omega)$, where x, z, ψ denote the planar positional coordinates and pitch angle, and v_x, v_z, ω denote their time derivatives respectively. The dynamics used for computing the TTR function are given in Eq. (3.10). The quadrotor's movement is controlled by two motor thrusts, T_1 and T_2. The quadrotor has mass m, moment of inertia I_{yy}, and half-length l. Furthermore, g denotes the gravity acceleration, C_D^v the translation drag coefficient, and C_D^ψ the rotational drag coefficient. Similar to the car example, the full state s contains eight laser readings extracted from the "Hokoyu_utm30lx" ranging sensor for detecting obstacles, in addition to the internal state $\tilde{s}$. The objective of the quadrotor is to learn a policy, mapping from states and observations to thrusts, that leads it to the goal region.

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{z} \\ \dot{v}_z \\ \dot{\psi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ -\frac{1}{m} C_D^v v_x + \frac{T_1}{m} \sin\psi + \frac{T_2}{m} \sin\psi \\ v_z \\ -\frac{1}{m} \left(mg + C_D^v v_z\right) + \frac{T_1}{m} \cos\psi + \frac{T_2}{m} \cos\psi \\ \omega \\ -\frac{1}{I_{yy}} C_D^\psi \omega + \frac{l}{I_{yy}} T_1 - \frac{l}{I_{yy}} T_2 \end{bmatrix} \tag{3.10}$$

The environment for this task is shown in Fig. 3.5. The obstacles are fixed. The goal region is $\mathbf{G} = \{(x, z, \psi) | 3.5\,\text{m} \leq x \leq 4.5\,\text{m}; 8.5\,\text{m} \leq z \leq 9.5\,\text{m}; 0.45\,\text{rad} \leq \psi \leq 1.05\,\text{rad}\}$. The quadrotor's starting condition is uniformly-randomly sampled from $\{(x, z) | 2.5\,\text{m} \leq x \leq 3.5\,\text{m}; 2.5\,\text{m} \leq z \leq 3.5\,\text{m}\}$ (green area in Fig. 3.5) and the starting pitch angle is randomly sampled from $\{\psi | -0.17\,\text{rad} \leq \psi \leq 0.17\,\text{rad}\}$.

Fig. 3.6 shows the performance of TTR-based, distance-based, and sparse reward under optimization from DDPG, TRPO, and PPO. With TTR-based reward, performance is consistent over all model-free algorithms. In contrast, sparse and distance-based rewards often do not lead to quadrotor stability and consistency of performance. For example, under sparse and distance-based rewards, performance is relatively good under TRPO, but very

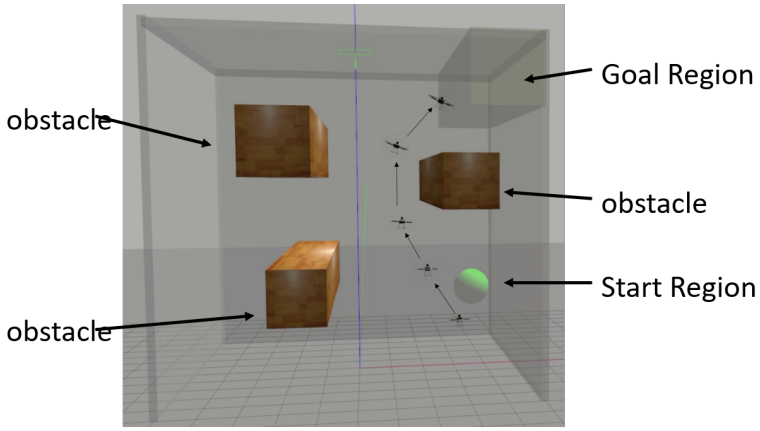


Figure 3.5: Visualization of quadrotor's sequential movement after learning from TTR-based reward. The trajectory is connected by a combination of the same quadrotor at a few different time snapshots. As shown in the picture, the quadrotor has learned to make use of physical dynamics (tilt) to reach the target as soon as possible

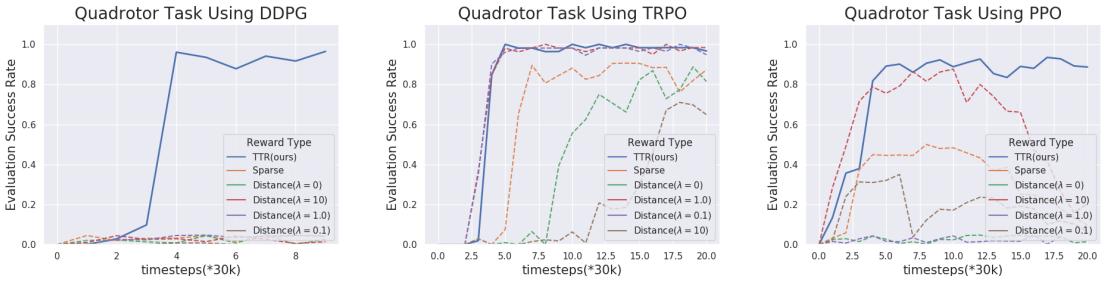
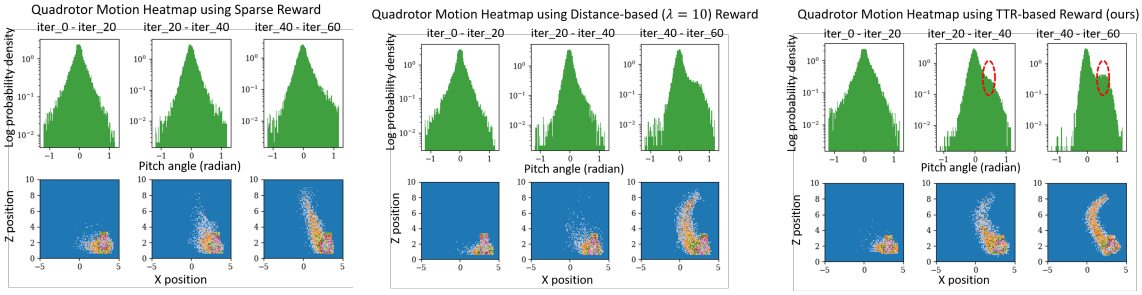


Figure 3.6: Performance comparison between TTR, distance, and sparse-based rewards on quadrotor using three different model-free algorithms. The results are based on an identical evaluation setting as a car example and are concluded from five runs as well. Our TTR-based reward achieves the best in terms of efficiency and performance. **Left**: success rate comparison under DDPG algorithm **Middle**: success rate comparison under TRPO algorithm **Right**: success rate comparison under PPO algorithm

poor under DDPG. In terms of learning efficiency, TTR-based reward achieves a success rate of greater than 90% after 3 iterations (approximately 90K time steps) regardless of the model-free algorithm. This is the best result among the three reward functions we tested.



(a) Sparse reward (b) Distance reward (the best λ) (c) TTR-based reward

Figure 3.7: Frequency histograms of (x, z, ψ) during different learning stages on quadrotor task. Top row: log probability density vs. ψ; bottom row: (x, z) heatmap. Only the TTR-based reward leads to near-complete trajectories in the (x, z) heatmap between iterations 20 and 40, when the other rewards still involve much exploration. Also, the shift (circled in red on Fig. 3.7c) of log probability density towards the target θ at 0.75 rad occurs only when TTR-based reward is used, which suggests TTR function is guiding learning effectively.

To further illustrate the effectiveness of our approach, Fig. 3.7 shows statistics of positional (x, z) and angular variables ψ during early learning stages using the three reward functions over three ranges of learning iterations. Note that we choose distance-based reward with $\lambda = 10$ since it's the best one among all tested distance-based reward variants. The 2D histograms show the frequency of (x, z) along trajectories in training episodes as heatmaps while the 1D histograms show the frequency of ψ as log probability densities. Note that the second column of each subplot (from iteration $20 \sim 40$) represents transitory behaviors before the quadrotor successfully learns to perform the task. Fig. 3.7c shows that the desired angular goal ($\psi = 0.75$ rad) has higher probability density (circled in red), which means TTR-based reward does provide effective angular local feedback. Furthermore, heat maps for TTR-based reward (Fig. 3.7c) are concentrated around plausible trajectories for reaching the goal, while heat maps for the other rewards are more spread out. This shows that TTR-based reward is providing dynamics-informed guidance.

3.3 Optimal Control Values as Policy Gradient Baseline

Deep reinforcement learning has achieved remarkable success in robotics [117, 120, 79, 58]. One of the key techniques behind the success is the policy gradient method. It uses gradient descent to directly optimize a parameterized control policy with sampled task data, enabling effective learning of high-dimensional and continuous policies but yielding gradient estimates with high variance.

The baseline function is a commonly used component in the policy gradient methods to mitigate its high variance of gradient estimates [43, 53]. A baseline is typically a state-dependent function subtracted from the observed total reward, also called return, resulting in a shifted return. This shifted return yields an unbiased estimate of the gradient with reduced variance. A baseline can have various choices, including the value-based [117, 120, 44], gradient norm-based [102], and trajectory-based forms [28].

Previous research on baseline [44, 102, 84, 161, 28] has primarily focused on mitigating gradient variance. While reducing variance is essential, it may not be adequate to ensure policy success, given the challenges of insufficient guidance and exploration in sparse feedback environments. Although other methods exist to tackle these challenges, there is limited research that addresses them specifically from the perspective of the baseline function. Therefore, we aim to investigate a novel aspect of baseline, particularly its role in addressing the problem of inefficient exploration in RL.

In this section, we introduce a novel, optimal control-based baseline to provide guided exploration for policy gradient methods. The baseline is obtained by solving a value function for an optimal control problem. This problem stems from the original RL task, from where a cost function and a coarse mathematical model are identified to approximately describe the objective and the robot system involved in the RL task. This allows the use of optimal control techniques for efficient value computation. Then the value function of the optimal control problem can be used as the baseline for the policy gradient method to solve the RL task. The key insight here is that the optimal control value function can offer essential prior information related to the RL task, such as the robot system dynamics, thus providing guided exploration for policy gradient RL.

We empirically evaluate our method using the standard policy gradient algorithm on a variety of robot learning tasks and provide a thorough analysis of the baseline's effect on guided exploration, particularly under sparse reward setups.

3.3.1 Related Work

Baselines in Policy Gradient RL

Various baselines have been proposed in policy gradient RL to mitigate its high variance of gradient estimate. A foundational study [43] established the theoretical bounds of estimated gradient variance induced by commonly used baselines such as the value function. To further lower the variance, recent studies have expanded the range of baseline forms, such as separate baselines for every coefficient of the gradient [102], Taylor expansion of the off-policy critic as baseline [44], state-action dependent baselines [84, 149], and a trajectory correlation-based baseline [28]. However, most baselines are designed for variance reduction while the potential role of baseline in other aspects, such as providing guided exploration for policy learning, is less studied.

Robot System Models from Control Theory

Classical control theory provides mathematical models for a range of robots based on their physical dynamics. For example, the Dubins car model [18, 23, 19] simplifies the motion of vehicles by assuming constant speed and position-steering. The extended Dubins car model [112, 87] allows variable speeds and turning rates, providing a more realistic representation. Both models are often used to describe differential-wheel robots and four-wheeled vehicles. In addition, the planar quadrotor model [20, 56] simplifies the quadrotor motion from 3D to 2D plane and the attitude model focuses solely on 3-axis rotational behavior [135] while point-mass model [110] only focuses on translation. Various other models, such as those for landing robots [73], and pendulum-like systems [100] are available. These models provide simplified abstractions of complex real-world robot models but still capture vital system priors with the potential to enhance modern robot learning methods, such as RL.

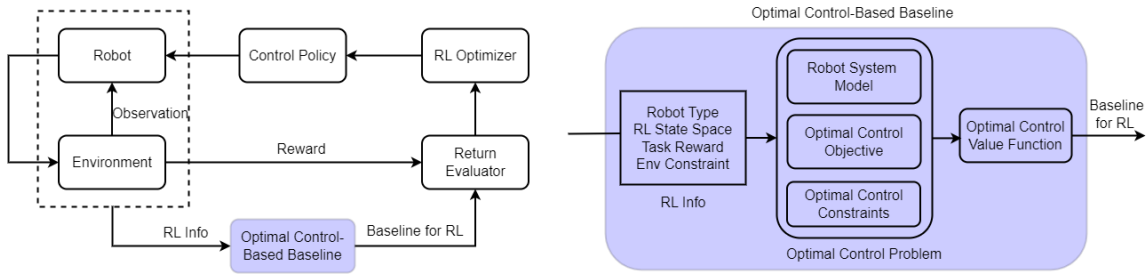


Figure 3.8: Overview of our method. **Left**: We propose a novel baseline function for policy gradient RL. Our method involves extracting RL info from the robot and environment, which is used to formulate an associated optimal control problem. Subsequently, we compute the optimal control value function and used it as a baseline for the policy gradient RL. **Right**: The RL info encompasses crucial aspects of the RL problem, including robot types, RL full state, task reward, and more. This RL info serves to form the key components of an optimal control problem, which include the robot system model, objectives, and constraints. Techniques like Model Predictive Control (MPC) can then be employed to compute the value function, which can be utilized as an RL baseline.

Relations to Our Work

Our work falls into the realm of introducing a novel baseline for policy gradient RL. Unlike prior work which focused on reducing gradient variance, this new baseline prioritizes providing valuable guidance in a subspace of the RL full state space during policy learning. Furthermore, our baseline is formed by utilizing the value function derived from an associated optimal control problem. It innovatively leverages well-developed robot system models from control theory to incorporate essential robot priors to provide guidance exploration for policy gradient RL.

This section describes the process of designing an optimal control-based baseline. It starts by extracting the key RL info from the RL task to form an associated optimal control

problem. Then the value function of this optimal control problem is computed and serves as a baseline for policy gradient RL to solve the original RL task, as shown in Fig. 3.8.

3.3.2 Method

Extraction of RL Info

A given RL problem often has key information characterizing its core aspects, which is denoted as RL Info in this work. There are in total four common components of RL info: robot type, state space, task reward, and environment constraints. This info can be extracted from the given robot and environment setup.

1. Robot Type. Robots can have diverse types, like cars, quadrotors, landing robots, etc.
2. RL State Space. RL's full state often includes the physical state of the robot such as its position and velocity as well as sensory observation such as pixels or LiDAR readings. This state could be high-dimensional.
3. Task Reward. It is numerical feedback provided to the robot at each time step to evaluate the robot's actions.
4. Environment Constraints. Typical constraints include factors such as maximum episode length, the valid range of state and action spaces, and physical limitations such as obstacles in the navigation task.

Formulation of Optimal Control Problem

Once the RL info mentioned above is obtained, an associated optimal control problem can be formed, which consists of three primary components: robot system model, optimal control objective, and related constraints.

Robot System Model. In control theory, there are many known models developed based on the physical dynamics of robot systems. Despite being abstracted and simplified from true dynamics, these models allow efficient optimal control computation and have the potential to be used in RL problems to mitigate its data inefficiency especially when the RL environment model is unknown and hard to learn. Thus, we aim to leverage those control theory models to calculate an optimal control value function, which can be used as a baseline function for policy gradient RL to provide guidance.

The robot system model is a mathematical equation describing robot physical dynamics and can be represented by a state-space Ordinary Differential Equation (ODE), denoted by $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ where $\mathbf{x}(t)$ and $\mathbf{u}(t)$ are the state and control of an optimal control problem at continuous time t and $\mathbf{f}(\cdot, \cdot)$ is the model function. Finite difference schemes such as the Forward Euler Method can be applied to this ODE to obtain its discrete version $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$, where k is the discrete time step.

This model can be heuristically selected from control theory based on RL info. To do that, we first categorize the robot system by its type (e.g., car-like, quadrotor, or others) to select a set of candidate models, referring to Table. 3.1. Then we assess candidate models with the following two criteria:

1. The components of optimal control state $\mathbf{x} \in \mathbb{R}^{\mathbf{x}_n}$ should be a subset from the components of RL full state $\mathbf{s} \in \mathbb{R}^{\mathbf{s}_n}$, where $\mathbb{R}^{\mathbf{x}_n}$ is the subspace of $\mathbb{R}^{\mathbf{s}_n}$.

2. The RL task reward $r(\cdot)$ can be written in terms of the components of $\mathbf{x}$, that is $r(\mathbf{s}) \coloneqq r(\mathbf{x})$. This criterion is typically met when the RL task reward is sparse and related to a limited set of states.

Optimal Control Objective. The optimal control objective specifies the desired outcome that an optimal control problem aims to achieve while satisfying the robot system model. It is often described in Eq. (3.11) as the minimization of cumulative cost subject to the system model constraint,

$$\begin{aligned} &\min_{\mathbf{u}_{0:T-1}} \sum_{k=0}^{T-1} c(\mathbf{x}_k, \mathbf{u}_k) \\ &\text{s.t. } \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \text{ for } k = 0, \dots, T-1 \end{aligned} \tag{3.11}$$

where $c(\mathbf{x}_k, \mathbf{u}_k)$ is the step cost function defined over optimal control state $\mathbf{x}_k$ and control input $\mathbf{u}_k$ at each discrete time step k, and $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ is the discrete-time robot system model. The key to deriving this objective is to find the appropriate cost function. Note that despite the presence of the RL reward, its direct utilization as the cost in Eq. (3.11) is generally impractical due to the potential difficulty in obtaining an optimal control solution.

The cost function $c(\mathbf{x}_k, \mathbf{u}_k)$ has multiple choices. One common choice is set-point tracking, where the cost can be defined by penalizing the error between the current and desired state whilst optionally minimizing the control effort

$$c(\mathbf{x}_k, \mathbf{u}_k) = \alpha e_{\mathbf{x}}(\mathbf{x}_k, \mathcal{G}) + \beta e_{\mathbf{u}}(\mathbf{u}_k), \tag{3.12}$$

where $e_{\mathbf{x}}(\cdot)$ and $e_{\mathbf{u}}(\cdot)$ are state and control error functions. To use this cost function, we assume the existence of a goal region $\mathcal{G}$ from the original RL problem. For instance, $e_{\mathbf{x}}$ can be expressed as the Euclidean distance between 2D positions in navigation tasks while $e_{\mathbf{u}}$ can be expressed as l^2-norm of control input $\|\mathbf{u}_k\|_2$. The cost function coefficients α, β can be flexibly adapted to obtain optimal control solutions.

Optimal Control Constraints. This includes additional limitations that must be obeyed when solving an optimal control problem. Typical constraints include the range of state space, the initial state, and the positions of obstacles in a collision-avoidance scenario. Those constraints can be added to the optimal control problem to rectify the solution.

Calculation of Optimal Control Value Function

We use Model Predictive Control (MPC) to solve the formulated optimal control problem and obtain the associated optimal control value function. MPC [105, 54] is a well-known control technique that repeatedly iteratively solves the optimization problem, and is used as a representative optimization method in this work.

We first uniformly generate many initial states from the optimal control state space. With these initial states and objective function Eq. (3.11), MPC produces a set of feasible trajectories, denoted by $\{\tau_j | j = 0, 1, 2, ..., N\}$, where N is the number of trajectories and each trajectory has a horizon of length H_j. From each trajectory τ_j, we select every state $\mathbf{x}_m^j$, which is the mth state of the jth trajectory and compute its value $V^{\text{oc}}(\mathbf{x}_m^j)$ following Eq. (3.13), where i is sum index.

$$V^{\text{oc}}\left(\mathbf{x}_m^j\right) = \sum_{i=m}^{H_j} \gamma^{i-m} r(\mathbf{x}_m^j) \tag{3.13}$$

This value sums up the discounted reward of each state along the whole trajectory span, where the γ^m means the discount factor of mth step of the trajectory. Given the states and their corresponding values, we form a discrete dataset $\mathcal{D} = \{(\mathbf{x}_m^j, V^{\text{oc}}(\mathbf{x}_m^j)) \mid \forall j, m\}$. Then, we can employ any regression model (e.g. neural network) to fit a continuous value function V^{oc} based on $\mathcal{D}$ for arbitrary states.

Optimal Control Baseline for Policy Gradient RL

The optimal control value function calculated above can be directly used in the policy gradient RL as the baseline function. Particularly, we consider a generic form of policy gradient RL that involves Generalized Advantage Estimation [119] given by Eq. (3.14). G_k^λ is the TD(λ) return [130] considering the weighted average of n-step returns for $n = 1, 2, ...\infty$ via parameter λ. π_θ is the RL parameterized policy that selecting action $\mathbf{a}_k$ based on the state $\mathbf{s}_k$ at each time step k. The baseline function, denoted as $b(\mathbf{s}_k)$, can take various forms, such as the widely used on-policy value function.

$$\nabla_\theta J\left(\pi_\theta\right) = \underset{\tau \sim \pi_\theta}{\mathbf{E}} \left[\sum_{k=0}^{T} \nabla_\theta \log \pi_\theta\left(\mathbf{a}_k \mid \mathbf{s}_k\right) \left(G_k^\lambda - b(\mathbf{s}_k) \right) \right] \tag{3.14}$$

In this work, we simply use the optimal control value function to be a novel form of baseline function: $b(\mathbf{s}_k) = V^{\text{oc}}(\mathbf{x}_k)$, where $\mathbf{x}_k$ is extracted from $\mathbf{s}_k$. We integrate it into Eq. (3.14) to estimate policy gradient $\nabla_\theta J\left(\pi_\theta\right)$ for optimizing policy π_θ to solve the robot learning task. This optimal control-based baseline can address challenging robot learning tasks with limited reward feedback, ensuring sufficient exploration.

3.3.3 Results

In this section, we present results on the performance of the proposed method and name our method as "OC baseline".

Demo Example: Differential-Drive Car Navigation

We first evaluate our method on a simulated TurtleBot [4], which resembles a differential drive car navigating in an unknown environment with multiple static obstacles. The car aims to learn a collision-free control policy via trial and error to move from a starting area to a goal area. We use the Gazebo simulator [62] to build the robotic system and environment for the target RL problem, as shown in Fig. 3.9.

Following the steps described in the methodology, the original RL problem provides info to form the optimal control baseline: (1) robot type is a car, (2) RL full state is $\mathbf{s} = (x, y, \theta, v, \omega, d_1, ..., d_8)$, which are 2D position, heading angle, current speed, current turning rate, and eight-dimensional lidar sensory data. (3) RL task reward is shown in Eq. (3.15), where $\mathcal{G}$ refers to the goal area, $\mathcal{C}$ the collision area and $\mathcal{I}$ intermediate area involving neither collision nor goal. All these areas are with regard to 2D position $\mathbf{p} = [x, y]^\top$.

$$r(\mathbf{s}) = \begin{cases} 0 & (x, y) \in \mathcal{I} \\ +1000 & (x, y) \in \mathcal{G} \\ -400 & (x, y) \in \mathcal{C} \end{cases} \tag{3.15}$$

Environment constraints include start area Γ, goal area $\mathcal{G}$, obstacle area $\mathcal{O}$, and map size. According to the robot type and key state variables of RL reward, we choose the 3D Dubins car model [33] as an abstraction of the true robot as it captures the 2D positional states and the simplest car motion which is computationally efficient. Its ODE is defined as:

$$\dot{\mathbf{x}} = g(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix},$$

where state $\mathbf{x} = [x, y, \theta]^\top$ and control input $\mathbf{u} = [\omega]^\top$, v is the constant speed.

Then we construct the objective function according to Eq. (3.11). As we discussed before, the key step here is designing a proper cost function. In the optimal control problem, the cost function should be designed based on the goal of the related RL problem and its reward function. In this task, the goal is to navigate the car to the goal area, so a typical cost function can be the 2D distance between the robot and the goal. As the distance to the goal decreases, the cost becomes smaller, otherwise, the cost becomes larger. A commonly used mathematical expression of such cost function can be in the quadratic form, written as follows, where $\mathbf{p}_k$ is the 2D position at step k and $\mathbf{p}^*$ is the goal position, $\mathbf{Q}$ and $\mathbf{R}$ are

coefficients of state and control to describe their relative importance weights.

$$\begin{aligned} \min_{\mathbf{u}_{0:H-1}} \sum_{k=0}^{H-1} & \left[(\mathbf{p}_k - \mathbf{p}^*)^\top \gamma^{k+1} \mathbf{Q} (\mathbf{p}_k - \mathbf{p}^*) + \omega_k^\top \mathbf{R} \omega_k \right] \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \\ & \text{initial state } \mathbf{x}_0 \in \Gamma \\ & \text{final state } \mathbf{x}_H \in \mathcal{G} \\ & \mathbf{p}_k \notin \mathcal{O} \\ & \forall \mathbf{p}_k \in \text{map area}. \end{aligned} \tag{3.16}$$

The value function can be calculated based on Eq. (3.16). Particularly, we use the IPOPT

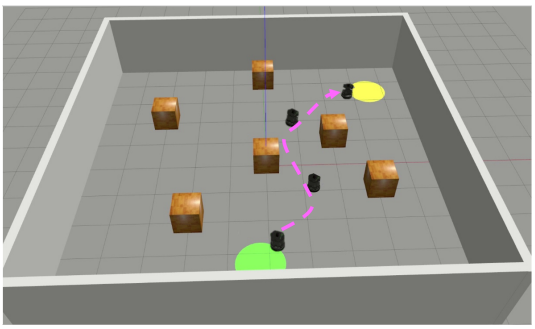


Figure 3.9: Car navigation environment

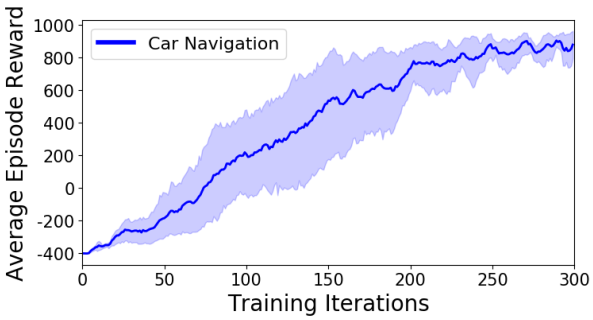


Figure 3.10: The car navigation reward performance

MPC solver [162] to solve 800 feasible trajectories with each trajectory having 140 step states, building associated state-value dataset $\mathcal{D} = \{(\mathbf{x}_i, V^{\text{oc}}(\mathbf{x}_i)) \mid i = 0, 1, 2, ..., 112000\}$ following Eq. (3.13). We then use a simple MLP neural network as a regression model to fit a continuous value function $V^{\text{oc}}(\mathbf{x})$. Finally, we utilize $V^{\text{oc}}(\mathbf{x})$ as the baseline function for a typical policy gradient algorithm PPO [120], following Eq. (3.14) to address the learning-based navigation task.

As depicted in Fig. 3.9, our method effectively learns a collision-free, goal-reaching policy. The policy is able to navigate cautiously through open spaces, skillfully avoid obstacles, and ultimately reach the desired goal along the trajectory in pink. Fig. 3.10 shows the car navigation training results that are summarized across 10 different trials. The curves show the mean reward and the shaded area represents the standard deviation of all trials. As depicted in the figure, as training iterations progress, the cumulative reward steadily increases and continues to rise until a gradual convergence towards a stable value. At 300 iterations the reward stabilizes around 800, which implies that the agent may have learned a relatively effective and stable policy.

Analysis of Guided Exploration

We conduct analysis on the primary benefit of our method in offering guided exploration for policy gradient RL. We illustrate a 2D plane value heatmap which is calculated from the Dubins Car optimal control problem. From Fig. 3.11, we can identify value distribution from any initial state to the goal. Regions with darker red colors correspond to higher values. The square concaves correspond to obstacles, denoting significantly lower values as they are to be avoided. With this value function as an RL baseline, it captures the state value priors related to the collision-free, goal-reaching areas in the 2D subspace of the RL full state, thereby providing valuable guidance for policy learning.

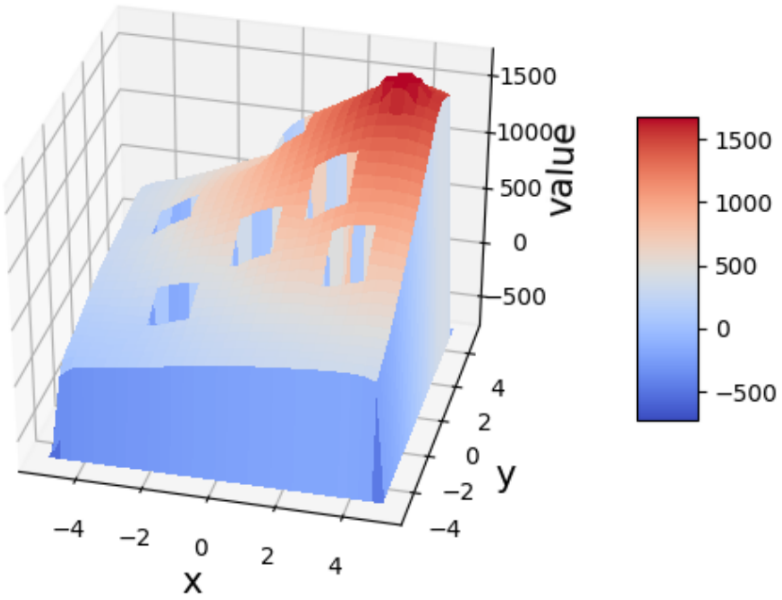


Figure 3.11: The optimal control value heatmap for car navigation

We also investigate the cause of exploration by analyzing the advantage estimation in the car example. The advantage $A^\pi(\mathbf{s}, \mathbf{a})$ is a function that drives the policy update in policy gradient RL by measuring how much better an action $\mathbf{a}$'s return than the baseline and is defined by $A^\pi(\mathbf{s}_k, \mathbf{a}_k) = G_k^\lambda - b(\mathbf{s}_k)$ in Eq. (3.14). In Fig. 3.12, we depict advantage estimation with/without the OC baseline (i.e. on-policy value function V^π as baseline), averaging every 10 iterations. With the OC baseline, policy gradient RL maintains a wider range of advantage estimation during the early learning stage. Conversely, without the OC baseline, the advantage estimation gradually narrows around zero. This indicates that the OC baseline consistently drives the exploration of diverse actions when initial actions are unfavorable ($A \leq 0$), while without OC baseline, the policy may get stuck in undesired local optima due to negligible advantage estimates, as $V^\pi(\mathbf{s}_k)$ will be gradually fitted to G_k^λ.

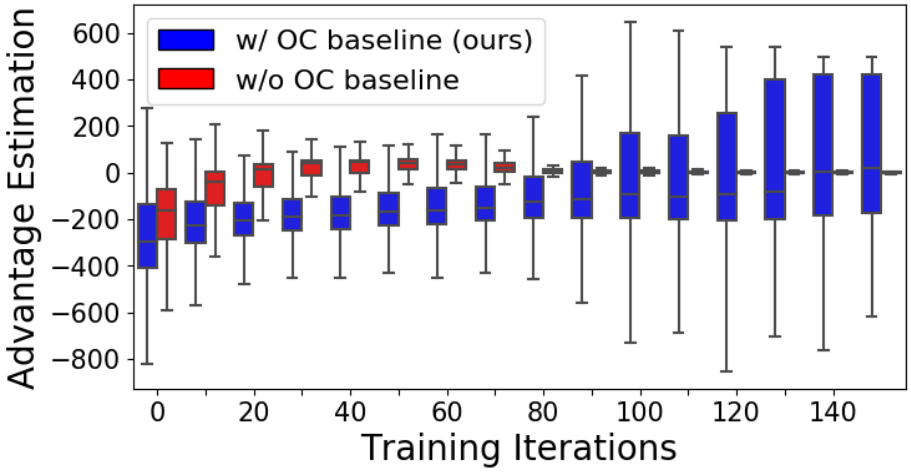


Figure 3.12: Policy advantage estimation w/ and w/o OC baseline. We halt at 150 iterations as it adequately demonstrates the substantial difference in advantage estimation between the two methods.

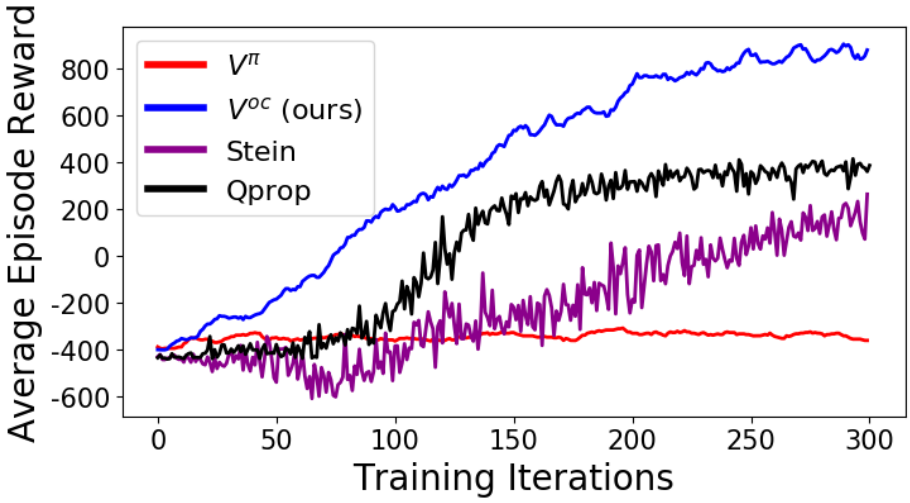


Figure 3.13: Comparison with other forms of baseline function. The curves show the mean value of episodic reward. Their standard deviations (not shown here) are within reasonable range without affecting the comparison.

Compare with Other methods

We compare with existing methods on the car navigation problem. Particularly, we select three other baseline functions and denote them by V^π [120], Stein [84] and Qprop [44]. V^π is the on-policy value function and is widely used as the baseline for actor-critic RL algorithms. Stein refers to a state-action dependent baseline based on the Stein Identity while Qprop refers to a baseline derived from a Taylor expansion of the off-policy critic. The comparative analysis, as depicted in Fig. 3.13, highlights the superior performance of our OC baseline, particularly in terms of achieving the highest average episodic reward. Our findings suggest that while baseline functions aimed at variance reduction are effective in many scenarios, they may not always provide feasible control solutions, especially when dealing with tasks with sparse rewards and requiring more guidance.

Ablation Study

We perform an ablation study to showcase the importance of the OC baseline by comparing the success rates of policies learned in the car navigation task with and without it. We choose PPO as the base policy gradient method and augment it with the OC baseline. By default, PPO uses an on-policy value function V^π as its baseline. In Fig. 3.14, with OC baseline, the robot continuously improves its policy and finally acquires a collision-free, goal-reaching policy with a success rate of around 100%. In contrast, the standard PPO, which is used without the OC baseline, struggles to develop effective behaviors. This indicates that the RL trial-and-error exploration is data-inefficient and the policy can get stuck to local optima when reward is sparse, but the OC baseline utilizes system and task priors from the associated optimal control problem to improve policy learning.

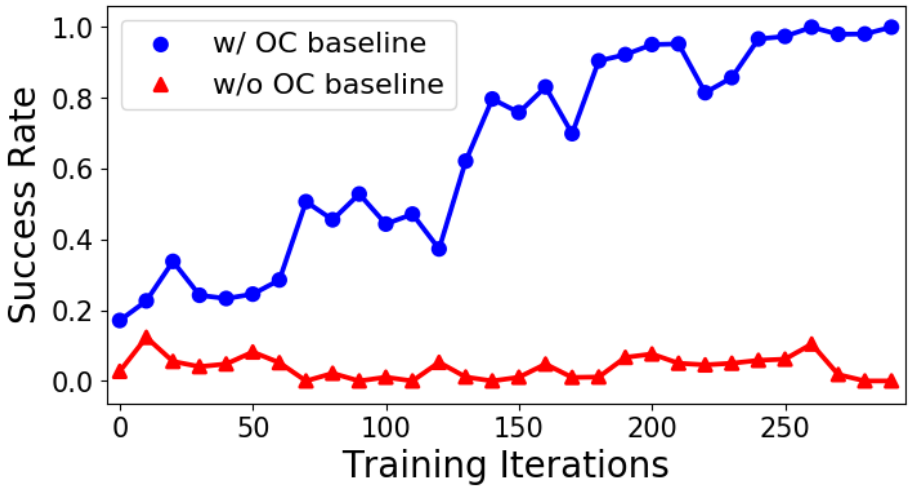


Figure 3.14: Ablation study of OC baseline method

From another perspective, it is worth noting that the control solution obtained from the associated optimal control problem is usually not directly applicable to the original RL problem. This gap arises because the chosen robot system model is typically an abstraction and simplification of the actual, unknown robot model, plus the optimal control input space may differ from the RL action space.

Applied OC baseline to Quadrotor Task

This method has also been extended to a more complex Quadrotor trap avoidance task, as shown in Fig. 3.15. We use a detailed planar quadrotor simulated by the Robot Operating System (ROS) [107] as a complex robot. Controlling a quadrotor is notoriously difficult due to its under-actuated nature. This experiment aims to confirm that our method remains effective even when the robot learning task involves a highly dynamic and unstable system.

The quadrotor receives RL full state $\mathbf{s} = (x, v_x, z, v_z, \psi, \omega, d_1, ..., d_8)$, where x, z, ψ are the planar positional coordinates and pitch angle, and v_x, v_z, ω denote their time derivatives respectively. It also contains eight sensor readings extracted from the laser rangefinder for obstacle detection. m is the mass of quadrotor, g is gravity, C_D^v and C_D^ϕ are drag coefficients and I_{yy} is the rigid body inertia based on y-axis. The quadrotor intends to learn a policy mapping from full RL states to its two thrusts F_1 and F_2 to reach a goal while avoiding the trap. We use a sparse reward function for this task, where Tr is the trap area.

$$r(\mathbf{s}) = \begin{cases} 0 & (x, y) \in \mathcal{I} \\ +1000 & (x, y) \in \mathcal{G} \\ -400 & (x, y) \in \mathcal{C} \\ +100 & (x, y) \in \text{Tr} \end{cases} \tag{3.17}$$

This task is challenging because, without effective guided exploration, the quadrotor tends to learn a policy that leads it to the trap instead of the goal, due to the positive trap reward and nearby position, which is easily reachable by gravity.

To compute the OC baseline, we choose a 6D planar model [87] as the simplified abstraction of the fully planar quadrotor, shown in Eq. (3.18). The corresponding MPC problem for this task is similar to Eq. (3.16), but adding an extra term $-(\mathbf{p}_k - \hat{\mathbf{p}}^*)^\top \gamma^{k+1} \mathbf{W} (\mathbf{p}_k - \hat{\mathbf{p}}^*)$ as the penalty of trap-reaching to the cost function, where $\hat{\mathbf{p}}^*$ is the center position of trap

and $\mathbf{W}$ is penalty coefficients.

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{z} \\ \dot{v}_z \\ \dot{\psi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ -\frac{1}{m} C_D^v v_x + \frac{F_1}{m} \sin\psi + \frac{F_2}{m} \sin\psi \\ v_z \\ -\frac{1}{m}\left(mg + C_D^v v_z\right) + \frac{F_1}{m} \cos\psi + \frac{F_2}{m} \cos\psi \\ \omega \\ -\frac{1}{I_{yy}} C_D^{\psi} \omega + \frac{l}{I_{yy}} F_1 - \frac{l}{I_{yy}} F_2 \end{bmatrix} \tag{3.18}$$

Fig. 3.15 depicts the environment and quadrotor's trajectories by executing policies learned with (blue dashed line) and without (red dashed line) our method. We use PPO as the base policy gradient method. With OC baseline, the robot learns the desired goal-reaching policy otherwise can easily lead to a trap-reaching policy without our baseline.

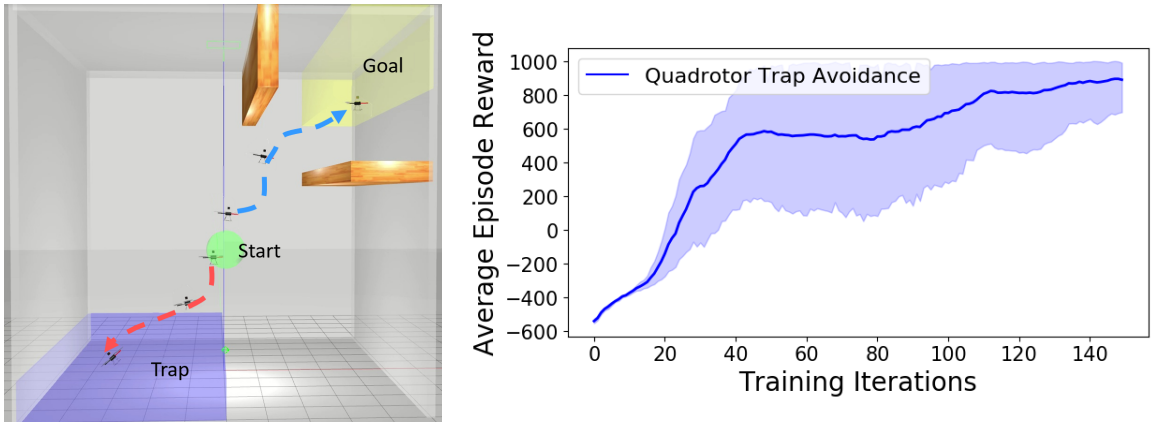


Figure 3.15: Quadrotor trap avoidance task environment and reward performance

The training results of this experiment are demonstrated in Fig. 3.15, indicating that in the quadrotor trap avoidance task, the final cumulative reward stabilizes at around 900. Table. 3.4 shows our method has an increasing goal-reaching rate as learning progresses while the normal RL without our baseline has an increasing trap-reaching rate. This result indicates that our method indeed provides guided exploration for the RL process. In this "Trap-Goal" example, although the "Trap" provides a positive reward as a good incentive for RL, The OC baseline perceives that the "Goal" region provides even better rewards so that the policy is optimized toward a global optimal direction towards the goal.

Method	Succ rate ↑	Trap rate ↓
PPO w/o OC baseline	0.13	0.65
PPO w/ OC baseline (ours)	**0.87**	**0.02**

Table 3.4: Comparison of the goal- and trap-reaching rate of quadrotor trap avoidance task.

Analysis: Robustness to Model Misspecification and State Mapping

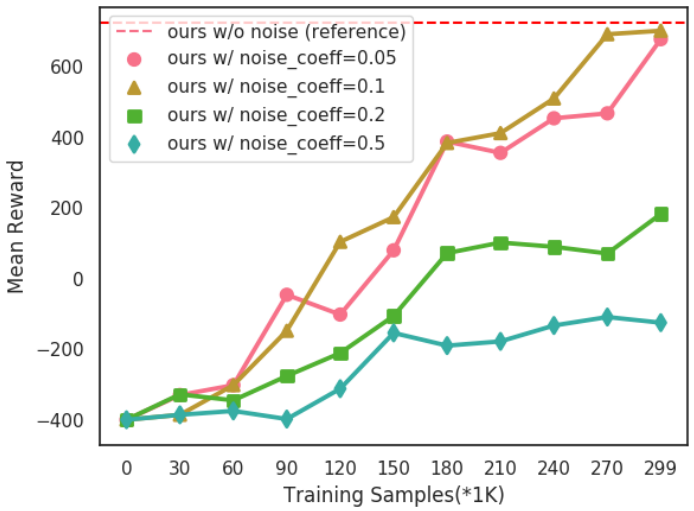


Figure 3.16: Mean reward of policy trained via our method but with different levels of noise between hi-dim and lo-dim state mapping. Data are averaged over 5 runs on the car navigation task.

We further test the robustness of our method to model misspecification via inaccurate mapping. In Fig. 3.16, we perturb the state mapping between hi-dim and lo-dim with different levels of random noise to include state uncertainty. This will potentially have the effect of increasing the model mismatch, thereby reducing the optimality of the lo-dim controller. Particularly, for each hi-dim observation, we add noise to its lo-dim corresponding state to make its lo-dim value prediction less accurate. The inaccurate lo-dim value is used for the hi-dim baseline term during the whole learning process. We show that our method can resist certain levels of state noise. In Fig. 3.16, a noise coefficient of 0.05 and 0.1 means we add a Gaussian noise with the specified standard deviations to the lo-dim state (x, y, θ). Even under such disturbances, we observe significant progress in policy performance (brown and red curves), close to the reference level. Moreover, with 0.2 radians, a relatively large random noise to heading angle $\theta \in [-\pi, \pi]$, our method still acquires around 200 rewards which are equivalent to around 50% success rate of the goal-navigation. As the noise level continues to increase, the policy training becomes worse due to very inaccurate value estimation.

Analysis: Model-Mismatch affects Low-Level Controller

The model mismatch between the selected mathematical model $\mathbf{f}(\mathbf{s}, \mathbf{a})$ and the target simulation model $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ (both over lo-dim state x) is a notable feature of our application scenarios. We investigate it quantitatively based on a quadrotor navigation task, and show its effect on the direct use of a lo-dim MPC controller. We first numerically measure the system state transitional error over multiple control steps between using $\mathbf{f}(\mathbf{s}, \mathbf{a})$ and $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$,

as shown in Table. 3.5. It turns out the state transitional error becomes significant after 10 control steps (i.e. quadrotor's positional state error increases to 1.14m, 2 times greater than the vehicle size 0.5m). In Fig. 3.17, we further show an actual control trajectory of an MPC controller running in target simulation but using selected model $\mathbf{f}(\mathbf{s}, \mathbf{a})$ for prediction. It turns out the model mismatch affects the control performance a lot and a good MPC controller designed based on lo-dim model $\mathbf{f}(\mathbf{s}, \mathbf{a})$ may not work well in the target environment with model $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ (green curve, collisions to obstacle). In contrast, our method trains an RL controller directly from the target environment in a model-free way and ends up with a successful trajectory towards the goal.

	1 Step	5 Steps	10 Steps
Car	$0.095 \pm 4.1\%$	$0.391 \pm 5.5\%$	$0.518 \pm 6.5\%$
Quadrotor	$0.542 \pm 3.8\%$	$0.836 \pm 7.2\%$	$1.144 \pm 5.3\%$

Table 3.5: Mean and standard deviation of system state transitional errors (over 10 roll-outs) between using the lo-dim control-theoretic model and target, simulation model after applying the same control sequences from the same initial state. The error is calculated based on Euclidean distance of 2D positions (unit: meters).

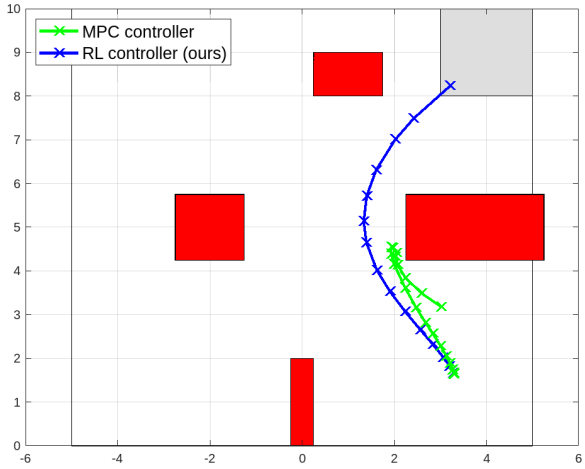


Figure 3.17: 2D positional trajectory of MPC and RL controller. MPC uses a lo-dim model for planning and executes its control in target simulation while RL controller uses our value-based warm-starting method to learn from pure simulated experiences. Red blocks are obstacles, green block is the goal.

Analysis: Better than Enhanced Exploration

To investigate whether a simple exploration strategy can work similarly well to our method, we compare our method with standard PPO with its exploration enhanced by different levels of policy entropy. We do this because entropy is easily generalizable to many methods, and has been shown to be effective in a variety of problems. We vary the ratio of the entropy term in the PPO loss function and choose its coefficient to be in the range of $[0, 0.1]$, a

	PPO w/ Entropy Coeff				PPO w/ MBB
	0.01	0.02	0.05	0.10	
Car	-12 ± 15	-52 ± 21	-32 ± 22	-24 ± 37	**678 ± 63**
Trap-Goal	44 ± 51	48 ± 33	65 ± 20	50 ± 40	**929 ± 77**

Table 3.6: Mean and standard deviation of evaluated return of final policy trained via our method and standard PPO with various levels of entropy-based exploration.

typical maximum range often used for PPO. We summarize results on car navigation and trap-goal examples in Table. 3.6. It clearly indicates that increasing exploration in a simple fashion does not significantly improve policy performance. This is because entropy only provides *undirected*, "uniformly directed" exploration, which sometimes is not sufficient to drive the policy update towards a specific goal-reaching direction. In contrast, our method uses the lo-dim value function that provides *directed* exploration via an advantage function that can inform correct policy gradient direction to the actual goal (see Fig. 3.11, 3.12 and Table. 3.7), rather than uniform exploration. This was clearly evident in the Trap-Goal example. The car navigation example with sparse reward has many obstacles and is constantly without effective exploration, which results in conservative behaviors (e.g. moving nowhere and obtaining 0 reward) when the agent only explores in an *undirected* manner.

Analysis: Correctness of Policy Gradient Estimation

Sample Size	Cosine Similarity	
	PPO	PPO w/ OC Baseline
2^7	0.238	**0.331**
2^9	0.422	**0.615**
2^{11}	0.674	**0.858**
2^{13}	0.855	**0.961**
2^{15}	0.914	**0.982**

Table 3.7: Cosine similarity is used to measure the closeness between the estimated policy gradient and the true gradient. Our method computes a more correct gradient direction (larger cosine similarity at any sample size level) w.r.t. true gradient, showing its effectiveness in providing better guidance on policy updates.

We further investigate policy gradient estimation, in particular the gradient's direction, to offer more insights into the cause of global guidance. Specifically, we quantify the closeness between the estimated and true policy gradient using cosine similarity. We use an early stage policy, one that visits the trap and goal roughly equally often, to compute the "true" policy gradient over a significantly large number of samples $N_{\max} = 102400 \approx \infty$ via Eq. (3.14) without any $b(\cdot)$. Then we estimate the policy gradients via Eq. (3.14) using two baselines: $V^{\pi}(\cdot)$ and $V^{\text{oc}}(\cdot)$ respectively over various sample sizes. Finally, we calculate cosine similarity

between estimated and "true" gradients and summarize results in Tab. 3.7. We find that with $b(\cdot) = V^{\text{oc}}(\cdot)$, our method can provide a better approximation of policy gradient, even when the number of samples is low, indicating its ability to implicitly direct the policy gradient towards the *globally-optimal* region in a data-efficient way.

3.4 Chapter Summary

In this chapter, we first propose TTR-based reward shaping to alleviate the data inefficiency of model-free RL on robotic tasks. The advantages of the TTR reward are: (1) It can be derived efficiently from the TTR value function within the RL subspace. (2) It encodes the minimization of goal-reaching time while taking into account essential robot dynamics, proving valuable in a wide range of robotic learning scenarios demanding rapid task completion. (3) It is easy to implement and can be used as a wrapper for any model-free RL algorithm as it does not alter the original RL structure. (4) It requires little human engineering. Moreover, by altering the Hamilton-Jacobi equations Eq. (3.4), TTR-based reward can easily involve the consideration of obstacles and disturbances. It can also be generalized to a reward predictor for various environments.

Secondly, we introduce a novel baseline function for policy gradient RL. The baseline is derived from the optimal control value function, which is computed based on an optimal control problem defined in the RL subspace and is formed to be closely related to the target RL task. We demonstrate that this control-based baseline can ensure guided exploration to improve policy gradient RL especially when the task has sparse and insufficient reward feedback. We further show this baseline function is a better alternative than entropy-based exploration and its guidance is robust to model misspecification. From this work, we open a new perspective on the utility of the baseline function.

Chapter 4

Scalable and Interpretable Learning-Based Control

This chapter is based on my paper "Task-Oriented Koopman-Based Control with Contrastive Encoder" [88][1] written in collaboration with Hanyang Hu, Seth Siriya, Ye Pu, and Mo Chen.

4.1 Chapter Overview and Related Work

Robot control is crucial in robotics and finds applications in various domains. Nonlinear and linear control are two primary approaches in robot control. Nonlinear control [126, 71, 31] is suitable for complex systems when a good nonlinear dynamical model is available. But such a model is not easy to obtain and the nonlinear computation can be sophisticated and time-consuming. Linear control [142, 76, 109] is relatively simple to implement and computationally efficient for systems with linear, simple dynamics, but can exhibit poor performance or instability in realistic systems with highly nonlinear behaviors. Based on Koopman operator theory [64], Koopman-based control [57, 104, 143, 51] offers a data-driven approach that reconciles the advantages of nonlinear and linear control to address complex robot control problems. It transforms the (unknown) nonlinear system dynamics into a latent space in which the dynamics are (globally) linear. This enables efficient control and prediction of nonlinear systems using linear control theory.

Numerous studies have been done on Koopman-based control and they typically follow a two-stage model-oriented process [51, 124, 69]. The first stage is to identify a Koopman model – that is, a globally linear model – from system data, which involves finding a Koopman operator and its associated embedding function to represent linearly evolving system dynamics in the latent space. Classical methods use matrix factorization or solve least-square regression via pre-defined basis functions, while modern methods leverage deep learning techniques [85, 124, 51, 69, 163, 144], such as deep neural networks (DNNs) and autoencoder frameworks, to enhance Koopman model approximation. In the second stage,

[1]Supplementary contents: https://sites.google.com/view/kpmlilatsupp

a linear controller is designed over the latent space based on the Koopman model. Various optimal control methods for linear systems, including Linear Quadratic Regulator (LQR) [91, 69, 124] and Model Predictive Control (MPC) [2, 59, 65, 144], have been employed.

The model-oriented approach in the aforementioned works prioritizes Koopman model accuracy for prediction rather than control performance. While it allows the model to be transferred and reused across different tasks, it has certain limitations. Firstly, it involves a sequential two-stage process, where the performance of the controller is highly dependent on the prediction accuracy of the Koopman model. Thus, slight prediction inaccuracies of the learned model can significantly degrade the subsequent control performance. Secondly, even if the model is perfect, the cost function parameters for the linear controller (e.g. Q and R matrices in LQR controller) need careful manual tuning in both *observed and latent space* in order to have good control performance. These challenges are particularly pronounced in problems with high-dimensional state spaces, thus restricting the applicability of Koopman-based control to low-dimensional scenarios.

In this work, we propose a task-oriented approach with a contrastive encoder for Koopman-based control of robotic systems. Unlike existing works that prioritize the Koopman model for prediction, our task-oriented approach emphasizes learning a Koopman model with the intent of yielding superior control performance. To achieve this, we employ an end-to-end reinforcement learning (RL) framework to *simultaneously* learn the Koopman model and its associated linear controller over latent space within a single-stage loop. In this framework, we set the minimization of the RL task cost to be the primary objective, and the minimization of model prediction error as an auxiliary objective. This configuration has the potential to alleviate the aforementioned limitations: (1) RL optimization provides a dominant, task-oriented drive for controller update, reducing its reliance on accurate model identification. (2) Manual tuning of cost function parameters is unnecessary as they can be learned implicitly along with the controller in the end-to-end loop.

More specifically, we adopt a contrastive encoder as the Koopman embedding function to learn the linear latent representation of the original nonlinear system. In contrast to the commonly-used autoencoder, we demonstrate the contrastive encoder as a preferable alternative, delivering latent embedding that is well-suited for our end-to-end learning, especially in high-dimensional scenarios such as pixel-based control. To design the Koopman controller, we develop a differentiable LQR solution process, which serves as the linear controller over the latent system derived from the Koopman operator. This process is gradient-optimizable, allowing us to integrate it into our end-to-end RL framework and optimize controller parameters through gradient backpropagation. We empirically evaluate our approach through simulations across various tasks, demonstrating superior control performance while maintaining accurate Koopman model prediction. We compare our approach with two-stage Koopman-based control and pure RL method, providing a comprehensive assessment.

4.1.1 Koopman-Based Control.

B.O. Koopman [64] laid the foundation for analyzing nonlinear systems through an infinite-dimensional linear system via the Koopman operator. Subsequent works proposed efficient computation algorithms such as dynamical mode decomposition (DMD) [114, 115] and extended DMD (EDMD) [158, 159] to approximate the Koopman operator from observed time-series data. Recent research has expanded the Koopman operator theory to controlled systems [104, 157], and explored its integration with various control techniques such as LQR [13], MPC [65, 2, 66, 59], pulse control [127]. The emergence of deep learning has further enhanced the learning of Koopman embedding and operator using neural networks and autoencoders [85, 163], enabling their integration with optimal control [51, 124, 144].

4.1.2 Contrastive Representation Learning.

Contrastive representation learning has emerged as a prominent approach in self-supervised learning in computer vision and natural language processing [29, 98, 27, 52, 67, 174, 77], where it employs an encoder to learn a latent space where the latent representation of similar sample pairs are proximate while dissimilar pairs are distant. Recent works have extended contrastive learning to RL for robot control. Particularly, CURL [72] learns a visual representation for RL tasks by matching embeddings of two data-augmented versions of the raw pixel observation in a temporal sequence. The use of a contrastive encoder on RL enables effective robot control directly from high-dimensional pixel observations.

4.1.3 Relations to Our Work.

Our work falls into the realm of using deep learning for Koopman-based control. In contrast to existing two-stage approaches [51, 124] involving model identification and controller design, we propose a single-stage, end-to-end RL loop that simultaneously learns the Koopman model and controller in a task-oriented way. We also draw inspiration from the use of contrastive encoder [72], and specifically tailor it as Koopman embedding function for nonlinear systems with physical states and pixel observations. Our approach enhances the Koopman-based control to be used in high-dimensional control tasks beyond traditional low-dimensional settings.

4.2 Task-Oriented Koopman-Based Control with Contrastive Encoder

4.2.1 Problem Formulation

Consider an optimal control problem over a nonlinear, controlled dynamical systems

$$\min_{\mathbf{u}_{0:T-1}} \sum_{k=0}^{T-1} c\left(\mathbf{x}_k, \mathbf{u}_k\right) \quad \text{subject to } \mathbf{x}_{k+1} = \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right), \tag{4.1}$$

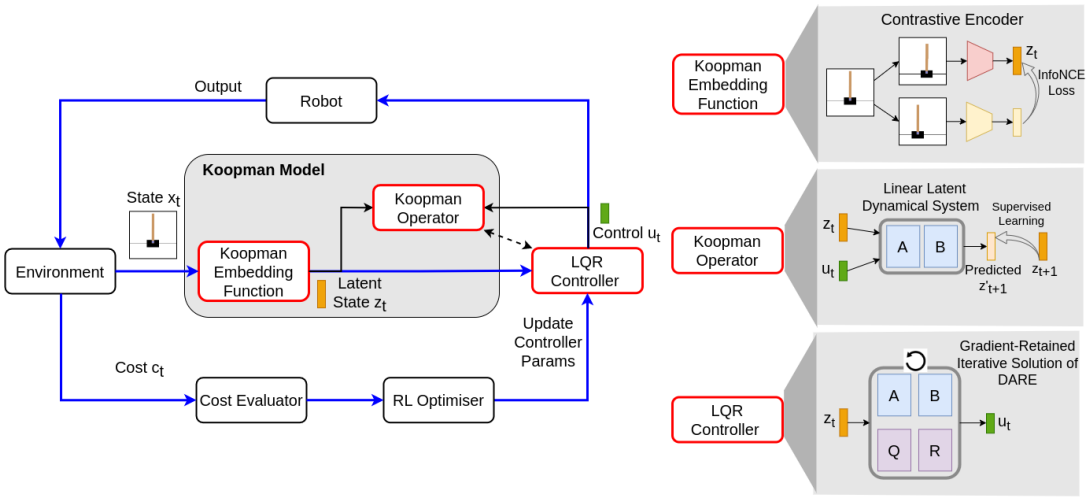


Figure 4.1: Overview of our method. We adopt an end-to-end RL framework to simultaneously learn a Koopman model and its associated controller. The Koopman model includes a contrastive encoder as the embedding function and a linear matrix as the operator. The Koopman controller is integrated into the loop as a differentiable LQR solution process to derive step optimal control and allow for the gradient-based update. We optimize the entire loop by considering the task cost as the primary objective and incorporating contrastive and model prediction losses as auxiliary objectives.

where state $\mathbf{x}$ evolves at each time step k following a dynamical model $\mathbf{f}$, and we aim to find a sequence of control $\mathbf{u}_{0:T}$ to minimise the cumulative cost $c(\mathbf{x}_k, \mathbf{u}_k)$ over T steps. Koopman operator theory [64, 104] allows the lifting of original state and input space $\mathbf{x} \in \mathbf{X}; \mathbf{u} \in \mathbf{U}$ to a *infinite-dimensional* latent embedding space $\mathbf{z} \in \mathbf{Z}$ via a set of scalar-valued embedding functions $\mathbf{\Psi} : (\mathbf{X}, \mathbf{U}) \to \mathbf{Z}$, where the evolution of latent embedding $\mathbf{z}_k = \mathbf{\Psi}(\mathbf{x}_k, \mathbf{u}_k)$ can be globally captured by a linear operator $\mathcal{K}$, as shown in Eq. (4.2).

$$\mathcal{K}\mathbf{\Psi}\left(\mathbf{x}_k, \mathbf{u}_k\right) \triangleq \mathbf{\Psi}\left(\mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right), \mathbf{u}_{k+1}\right) \tag{4.2}$$

Identifying the Koopman operator $\mathcal{K}$ and the embedding function $\mathbf{\Psi}$ is the key to Koopman-based control. In practice, $\mathcal{K}$ is often approximated using a finite-dimensional matrix $\mathbf{K}$, and the choice of $\mathbf{\Psi}$ is typically determined through heuristics or learning from data. Recent research [51, 124] has established the convention that employ neural networks $\psi(\cdot)$ to encode state $\mathbf{x}$, and define the Koopman embedding function $\mathbf{\Psi}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix}\psi(\mathbf{x}) & \mathbf{u}\end{bmatrix}$. Correspondingly, $\mathbf{K}$ is decoupled into state and control components, denoted by matrices $\mathbf{A}$ and $\mathbf{B}$, to account for $\psi(\mathbf{x})$ and $\mathbf{u}$ respectively. This results in a linear time-invariant system with respect to $\psi(\mathbf{x})$ and $\mathbf{u}$, shown in Eq. (4.3), facilitating linear control analysis and synthesis.

$$\mathbf{K}\mathbf{\Psi}(\mathbf{x}_k, \mathbf{u}_k) = \begin{bmatrix}\mathbf{A} \\ \mathbf{B}\end{bmatrix} \begin{bmatrix}\psi(\mathbf{x}_k) & \mathbf{u}_k\end{bmatrix} = \mathbf{A}\psi(\mathbf{x}_k) + \mathbf{B}\mathbf{u}_k = \psi(\mathbf{x}_{k+1}) \tag{4.3}$$

The goal of Koopman-based control is to identify the Koopman operator $\mathbf{K} = \begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix}^\top$, the embedding function $\psi(\mathbf{x})$ and a linear controller $\mathbf{u} = \pi(\mathbf{x})$ to minimise the total cost.

4.2.2 Contrastive Encoder as Koopman Embedding Function

Deep neural networks are extensively employed as flexible and expressive nonlinear approximators for learning Koopman embeddings in a latent space. Inspired by the success of contrastive learning, we adopt a contrastive encoder to parameterize the embedding function $\psi(\cdot)$. Specifically, for each state $\mathbf{x}_i$ in the data batch $\mathcal{B} = \{\mathbf{x}_i \mid i = 0, 1, 2, ...\}$, we create its associated query sample $\mathbf{x}_i^q$ and a set of key samples $\mathbf{x}_i^k$ that include positive and negative samples $\mathbf{x}_i^+$ and $\{\mathbf{x}_j^- \mid j \neq i\}$. $\mathbf{x}_i^+$ is generated by using different versions of augmentations on $\mathbf{x}_i$, while $\{\mathbf{x}_j^- \mid j \neq i\}$ are generated by applying similar augmentations for all the other states: $\mathcal{B}\backslash\{\mathbf{x}_i\} = \{\mathbf{x}_j \mid j \neq i\}$.

Following [52, 72], we use two separate encoders ψ_{θ_q} and ψ_{θ_k} to compute the latent embeddings: $\mathbf{z}_i^q = \psi_{\theta_q}(\mathbf{x}_i^q)$, $\mathbf{z}_i^+ = \psi_{\theta_k}(\mathbf{x}_i^+)$ and $\mathbf{z}_j^- = \psi_{\theta_k}(\mathbf{x}_j^-)$. Then we compute the InfoNCE loss over data batch $\mathcal{B}$ based on Eq. (4.4) as the contrastive loss $\mathcal{L}_{\text{cst}}$ to update encoders parameters θ_q, θ_k, as well as W which is a learnable parameter matrix to measure the similarity between query and key samples. Two encoders ψ_{θ_q} and ψ_{θ_k} are used for contrastive loss computation, but eventually only ψ_{θ_q} serves as the Koopman embedding function, and we simplify it notation as ψ_θ. We use $\mathbf{t} = (\mathbf{z}, \mathbf{u}, \mathbf{z}', r, d)$ to denote a tuple with current and next latent state $\mathbf{z}, \mathbf{z}'$, action $\mathbf{u}$, reward $r(\cdot) = -c(\cdot)$ and done signal d.

$$\mathcal{L}_{\text{cst}} = \mathbb{E}_{\mathbf{t}\sim\mathcal{B}} \log \left(\frac{\exp(\mathbf{z}_i^{q\top} W \mathbf{z}_i^+)}{\exp(\mathbf{z}_i^{q\top} W \mathbf{z}_i^+) + \sum_{j\neq i} \exp(\mathbf{z}_i^{q\top} W \mathbf{z}_j^-)} \right) \tag{4.4}$$

Different encoder structures and augmentation strategies are required to handle system states depending on how they are represented. For pixel-based states, we adopt convolutional layers as the encoder structure and apply random cropping for augmentation [52, 72]. For physical states, we utilize fully connected layers as the encoder structure and augment the states by adding uniformly distributed, scaled random noise as defined in Eq. (4.5). $\mathbf{x}^{|\cdot|}$ refers to element-wise absolute of $\mathbf{x}$.

$$\Delta\mathbf{x} \sim \mathrm{U}(-\eta\mathbf{x}^{|\cdot|}, \eta\mathbf{x}^{|\cdot|}); \quad \mathbf{x}^+ = \mathbf{x} + \Delta\mathbf{x} \tag{4.5}$$

4.2.3 Linear Matrices as Koopman Operator

Koopman operator describes linear-evolving system dynamics over the latent embeddings and is denoted by a matrix $\mathbf{K}$. Following Eq. (4.3), we decompose $\mathbf{K}$ into two matrices $\mathbf{A}$ and $\mathbf{B}$, representing the state and control coefficient of a linear latent dynamical system.

$$\mathbf{z}_{k+1} = \mathbf{A}\mathbf{z}_k + \mathbf{B}\mathbf{u}_k \tag{4.6}$$

To learn $\mathbf{A}$ and $\mathbf{B}$, we optimise a model prediction loss $\mathcal{L}_{\mathrm{m}}$, which is described by Mean-Squared-Error (MSE) as defined in Eq. (4.7). $\hat{\mathbf{z}}_{k+1}$ is the latent embedding obtained through contrastive encoder at $k+1$ step. It supervises the predicted latent embedding at $k+1$ step from Eq. (4.6).

$$\mathcal{L}_{\mathrm{m}} = \mathbb{E}_{\mathbf{t}\sim\mathcal{B}}\|\hat{\mathbf{z}}_{k+1} - \mathbf{A}\mathbf{z}_k - \mathbf{B}\mathbf{u}_k\|^2; \;\; \hat{\mathbf{z}}_{k+1} = \psi_\theta(\mathbf{x}_{k+1}) \tag{4.7}$$

Given Koopman embeddings $\mathbf{z} = \psi_\theta(\mathbf{x})$ and its associated linear latent system parameterized by $\mathbf{K} = \begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix}^\top$ shown in Eq. (4.6), Koopman-based approaches allow for linear control synthesis over latent space $\mathbf{Z}$.

Algorithm 1: Iterative solution of DARE

Set the total number of iterations M
Prepare current $\mathbf{A}, \mathbf{B}$, $\mathbf{Q}, \mathbf{R}$; initialise $\mathbf{P}_M = \mathbf{Q}$.
for $m = M, M-1, M-2, ..., 1$ **do**
    $\mathbf{P}_m = \mathbf{A}^\top\mathbf{P}_{m+1}\mathbf{A} - \mathbf{A}^\top\mathbf{P}_{m+1}\mathbf{B}(\mathbf{R} + \mathbf{B}^\top\mathbf{P}_{m+1}\mathbf{B})^{-1}\mathbf{B}^\top\mathbf{P}_{m+1}\mathbf{A} + \mathbf{Q}$
end for
Compute linear gain: $\mathbf{G} = (\mathbf{B}^\top\mathbf{P}_1\mathbf{B} + \mathbf{R})^{-1}\mathbf{B}^\top\mathbf{P}_1\mathbf{A}$
Generate optimal control for latent embedding $\mathbf{z}$:
$\mathbf{u}^* = -\mathbf{G}\mathbf{z}$

Formally, consider the LQR problem in Koopman latent space that can be formulated as Eq. (4.8) where $\mathbf{Q}$ and $\mathbf{R}$ are state and control cost matrices. In practice, we choose to represent $\mathbf{Q}$ and $\mathbf{R}$ as diagonal matrices to maintain their symmetry and positive definiteness. The LQR latent reference, denoted as $\mathbf{z}_{\mathrm{ref}}$, can be obtained from $\psi(\mathbf{x}_{\mathrm{ref}})$ if $\mathbf{x}_{\mathrm{ref}}$ is provided. Alternatively, $\mathbf{z}_{\mathrm{ref}}$ can be set to $\mathbf{0}$ in the latent space if $\mathbf{x}_{\mathrm{ref}}$ is not available. This is particularly useful in cases where the LQR problem does not have an explicit, static goal reference, such as controlling the movement of a cheetah.

$$\min_{\mathbf{u}_{0:T-1}} \sum_{k=0}^{T-1} \left[(\mathbf{z}_k - \mathbf{z}_{\mathrm{ref}})^\top \mathbf{Q} (\mathbf{z}_k - \mathbf{z}_{\mathrm{ref}}) + \mathbf{u}_k^\top \mathbf{R}\mathbf{u}_k \right] \;\text{ subject to }\; \mathbf{z}_{k+1} = \mathbf{A}\mathbf{z}_k + \mathbf{B}\mathbf{u}_k, \tag{4.8}$$

Solving the LQR problem in Eq. (8) involves solving the Discrete-time Algebraic Riccati Equation (DARE). One way this can be done is to take a standard iterative procedure to recursively update the solution of DARE until convergence, as shown in Algo. 1. In practice, we find performing a small number of iterations, typically $M < 10$, is adequate to obtain a satisfactory and efficient approximation for the DARE solution. Thus, we build a LQR control policy π_{LQR} over Koopman latent embedding $\mathbf{z}$ while dependent on a set of parameters $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}$, as described by Eq. (4.9).

$$\mathbf{u} \sim \pi_{\mathrm{LQR}}(\mathbf{z}|\mathbf{G}) \triangleq \pi_{\mathrm{LQR}}(\mathbf{z}|\mathbf{P}_1, \mathbf{A}, \mathbf{B}, \mathbf{R}) \triangleq \pi_{\mathrm{LQR}}(\mathbf{z}|\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}) \tag{4.9}$$

Together with $\mathbf{z} = \psi_\theta(\mathbf{x})$, Eq. (4.9) implies that the Koopman control policy π_{LQR} is differentiable with respect to the parameter group $\mathbf{\Omega} = \{\mathbf{Q}, \mathbf{R}, \mathbf{A}, \mathbf{B}, \psi_\theta\}$ over the input $\mathbf{x}$. Therefore, this process can be readily used in our gradient-based, end-to-end RL framework. During learning, we follow Algo. 1 to dynamically solve an LQR problem (4.8) at each step k with current parameters $\mathbf{\Omega}$ to derive a control $\mathbf{u}_k$ for the robot. To optimize the controller π_{LQR} towards lowering the task-oriented cost, we adopt a efficient off-policy RL algorithm, soft actor-critic (SAC) [47], to maximize the objective of Eq. (4.10) via off-policy gradient ascent over data sampled from batch buffer $\mathcal{B}$. Q_1, Q_2 are two Q-value approximators used in SAC. In principle, any other RL algorithms can also be utilized.

$$\mathcal{L}_{\text{sac}} = \mathbb{E}_{\mathbf{t}\sim\mathcal{B}}\left[\min_{i=1,2} Q_i(\mathbf{z}, \mathbf{u}) - \alpha \log \pi_{\text{sac}}(\mathbf{u} \mid \mathbf{z})\right]; \;\; \mathbf{z} = \psi_\theta(\mathbf{x}) \tag{4.10}$$

4.2.4 End-to-End Learning for Koopman Control

Algorithm 2: End-to-End Learning for Koopman Control

1: Initialise Koopman control parameters $\mathbf{Q}, \mathbf{R}, \mathbf{A}, \mathbf{B}, \psi_\theta$
2: Reset task environment $\mathcal{E}$.
3: Initialise a data replay buffer $\mathcal{D}$.
4: **for** $i = 0, 1, 2, ...$ **do**
5: Collect new roll-outs τ from $\mathcal{E}$ by running policy π_{LQR} following **Algo. 1**.
6: Save τ to $\mathcal{D}$ and sample a batch of data $\mathcal{B}$ from $\mathcal{D}$.
7: Compute $\mathcal{L}_{\text{sac}}, \mathcal{L}_{\text{cst}}, \mathcal{L}_{\text{m}}$ based on $\mathcal{B}$.
8: Update $\mathbf{\Omega} = \{\mathbf{Q}, \mathbf{R}, \mathbf{A}, \mathbf{B}, \psi_\theta\}$ based on $\mathcal{L}_{\text{sac}}$.
9: Update ψ_θ and $\mathbf{A}, \mathbf{B}$ based on $\mathcal{L}_{\text{cst}}$ and $\mathcal{L}_{\text{m}}$ respectively.
10: **end for**

We summarise the previous discussions and present the end-to-end learning process for task-oriented Koopman control with contrastive encoder, as illustrated in Fig. 4.1 and Algo. 2. The Koopman learning process is off-policy for better data efficiency.. We repeatedly collect batches of trajectory data from task environment $\mathcal{E}$ and utilise three objectives to update the parameter group $\mathbf{\Omega} = \{\mathbf{Q}, \mathbf{R}, \mathbf{A}, \mathbf{B}, \psi_\theta\}$ at each iteration. We take the RL task loss $\mathcal{L}_{\text{sac}}$ as defined in Eq. (4.10) to be the primary objective to optimise all parameters in $\mathbf{\Omega}$ for achieving better control performance on the task. Meanwhile, we use contrastive learning $\mathcal{L}_{\text{cst}}$ and model prediction $\mathcal{L}_{\text{m}}$ losses as two auxiliary objectives, as defined in Eq. (4.4) and Eq. (4.7), to regularise the parameter learning. $\mathcal{L}_{\text{cst}}$ is used to update $\psi_\theta(\cdot)$ to ensure a contrastive Koopman embedding space, while $\mathcal{L}_{\text{m}}$ is used to update $\mathbf{A}, \mathbf{B}$ to ensure an accurate Koopman model in the embedding space.

4.3 Results

We present simulated experiments to mainly address the following questions: (1) Can our method achieve desirable Koopman control performance for problems involving different state spaces with different dimensionalities? (2) Are we able to obtain a well-fitted globally linear model in the latent space? For all control tasks, we assume the true system models are unknown.

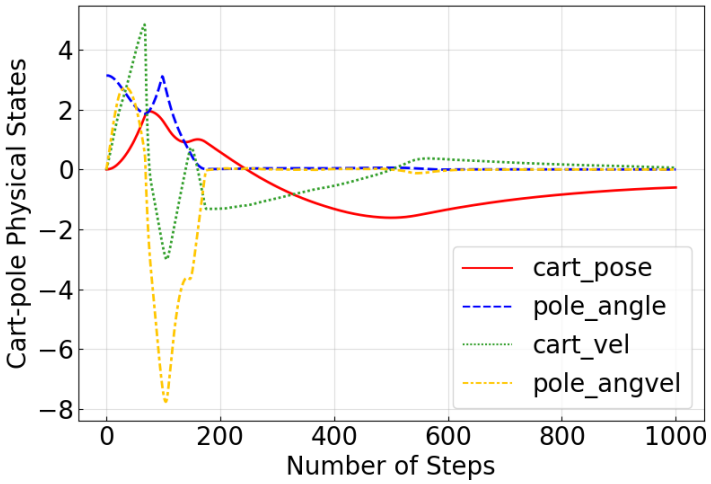
4.3.1 Task Environments

We include three robotic control tasks with varying dimensions in their state and control spaces from DeepMind Control Suite Simulator [139]: (1) **4D CartPole Swingup.** The objective of this task is to swing up a cart-attached pole that initially points downwards and maintain its balance. To achieve this, we need to apply proper forces to the cart. This task has 4D physical states of cart-pole kinematics as well as 1D control. (2) **18D Cheetah Running.** The goal of this task is to coordinate the movements of a planar cheetah to enable its fast and stable locomotion. It has 18D states describing the kinematics of the cheetah's body, joints, and legs. The 6D torques are used as a control to be applied to the cheetah's joints. (3) **Pixel-Based CartPole Swingup.** The CartPole swingup task with third-person-view images as the robot states.

4.3.2 Result Analysis

We report the results in Fig. 4.2, 4.3, 4.4 to demonstrate the effectiveness of our method. Fig. 4.3 shows the Koopman controller's performance by comparing its evaluation cost with the reference cost at various learning stages. The reference cost, obtained from [72], is considered the optimal solution to the problem. All experiments are tested over 5 random seeds. Across all three tasks, our method can eventually reach within 10% of the reference cost and continues to make further processes. This indicates our method is generally applicable to both simple, low-dimensional systems and very complex systems involving high-dimensional physical and pixel states. Fig. 4.2 showcases dynamical system behaviors by running a learned Koopman controller. The state evolution and temporal visual snapshots of the three tasks illustrate the successful control achieved by our method.

Fig.4.4 shows the Koopman model's prediction accuracy in the latent space. We employ t-SNE [145] to project the latent trajectories from 50D latent space onto a 2D representation for improved visualization. The plot in Fig. 4.4 shows the true and predicted states from trajectories consisting of 1000 steps. Significant overlapping and matching patterns are observed in the distribution of the data points for the 4D CartPole and 18D cheetah systems. This, plus the model prediction error, indicates the potential of utilizing a globally linear latent model to capture the state evolution in both simple and highly complex nonlinear systems. However, for pixel-based CartPole control, the projected states do not

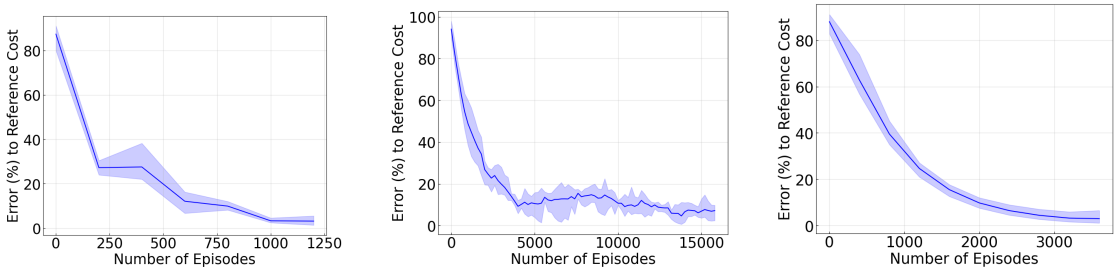


(a) Controlled states of 4D CartPole system

(b) Visualization of pixel-based CartPole control.

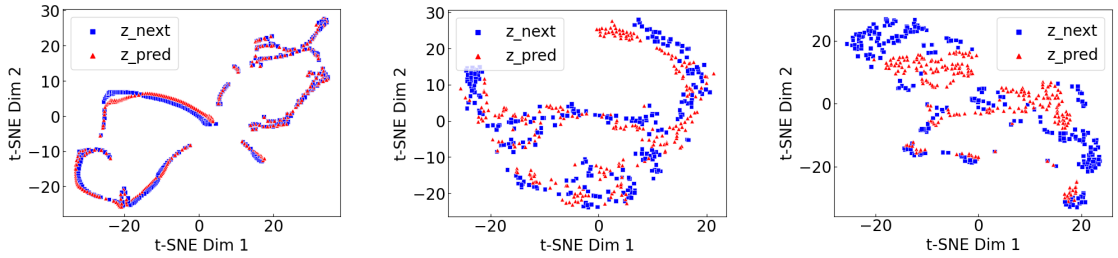
(c) Visualization of cheetah 18D control.

Figure 4.2: Dynamical system behaviors obtained by learned Koopman controller.



(a) 4D CartPole swingup (b) 18D cheetah running (c) Pixel-based CartPole

Figure 4.3: Mean and standard deviation of error between reference cost and our controller cost during learning.



(a) 4D CartPole swingup with mean model error: 2.72×10^{-3}

(b) 18D cheetah running with mean model error: 8.10×10^{-2}

(c) Pixel-based CartPole with mean model error: 3.71×10^{-1}

Figure 4.4: Distribution maps of 2D data points projected via tSNE from latent trajectories. **z_next** denotes true trajectories while **z_pred** denotes predicted trajectories using learned Koopman model.

perfectly match, suggesting difficulties in accurately modeling the pixel space. Nevertheless, our method still achieves good control performance, even with slight modeling inaccuracies. This highlights the advantage of our approach where the controller is less affected by the model.

4.3.3 Comparison with Other Methods

Ours vs. Model-Oriented Koopman Control

We compare our method with model-oriented Koopman control (MO-Kpm), which often requires a two-stage process of Koopman model identification and linear controller design. We compare with the most recent work [124] and conduct analysis through the 4D CartPole-swingup task.

Model Error	MO-Kpm		TO-Kpm (Ours)	
	Total Cost	Cost Variation	Total Cost	Cost Variation
$\sim 10^{-4}$	-188.10	-	**-872.18**	-
$\sim 10^{-3}$	-107.67	42.75%	**-846.88**	**2.90%**
$\sim 10^{-2}$	-64.32	40.27%	**-784.01**	**7.42%**

Table 4.1: Total control cost and its variation under different levels model error using MO-Kpm and our method.

Controller More Robust to Model Inaccuracy. Table. 4.1 presents the performance of the Koopman controller under varying levels of Koopman model accuracy. MO-Kpm experiences a rapid increase in total control cost with slightly increasing model error. In contrast, our method demonstrates superior and consistent control performances, indicating its better control quality as well as less dependency on the model's accuracy. This advantage arises from designing the controller primarily based on task-oriented costs rather than relying heavily on the model. Thus, our method is applicable not only to low-dimensional

systems but also to more complex and high-dimensional scenarios, such as the cheetah and pixel-based CartPole, where MO-Kpm cannot obtain a reasonable control policy.

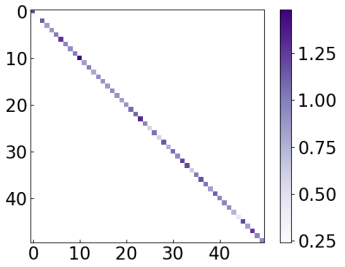


Figure 4.5: Learned Q matrix. Horizontal and vertical axes represent the rows and columns of a 50×50 diagonal matrix.

MO-Kpm	Total Cost
$\mathbf{Q}_1 = \mathbf{Diag}(84.12, 62.07, 65.79, 0.04, 0.04, 0, ...)$	-188.10
$\mathbf{Q}_2 = \mathbf{Diag}(0.01, 10, 10, 0.01, 0.01, 0, 0, ...)$	-109.23
$\mathbf{Q}_3 = \mathbf{Diag}(10, 60, 60, 0.01, 0.01, 0, 0, ...)$	-70.65
$\mathbf{Q}_4 = \mathbf{Diag}(10, 60, 60, 10, 10, 0.1, 0.1, ...)$	-124.80
TO-Kpm (Ours)	**Total Cost**
$\mathbf{Q}$ is shown on the right	**-846.88**

Table 4.2: Manually tuned and learned Q matrices for latent LQR, and their associated control costs.

Automatic Learning of Q Matrix in Latent Space. One major challenge of MO-Kpm is the difficulty in determining the state weight matrix $\mathbf{Q}$ for the latent cost function (Eq. (4.8)), especially for latent dimensions that may not have direct physical meanings. This challenge can lead to poor control performance, even when the identified model is perfect. Table. 4.2 compares the control costs obtained from several manually tuned $\mathbf{Q}$ matrices under the best-fitted Koopman model (10^{-4} level) with the learned $\mathbf{Q}$ using our method. Our approach enables automatic learning of $\mathbf{Q}$ over latent space and achieves the best control performance.

Ours vs. CURL

We compare our method with CURL [72], a model-free RL method that uses a contrastive encoder for latent representation learning and a neural network policy for control.

System Analysis using Control Theory. Our method differs from CURL in that we learn a linear Koopman model, whereas CURL does not. The presence of a Koopman model (parameterized by $\mathbf{A}, \mathbf{B}$ in Eq. 4.6) allows us to analyze the system using classical control theory and provides insights for optimizing the controller design. For the CartPole system, we perform stability analysis on both the 50D latent and the 4D true systems, and draw the pole-zero plots in Fig. 4.6. We find that the learned system demonstrates the same inherent instability as the true system, with the true system's poles accurately reflected in the poles of the latent system (overlapping blue and red dots).

We also analyze the controllability of the learned latent system and find its matrix rank as 6, which indicates that a latent dimension of 50 results in excessive uncontrollable states. Using this information, we apply our method with a lower-dimensional 6D latent space and can maintain the same control and model performance. Further decreasing the latent dimension to 4 leads to degraded control performance, suggesting that the controllability

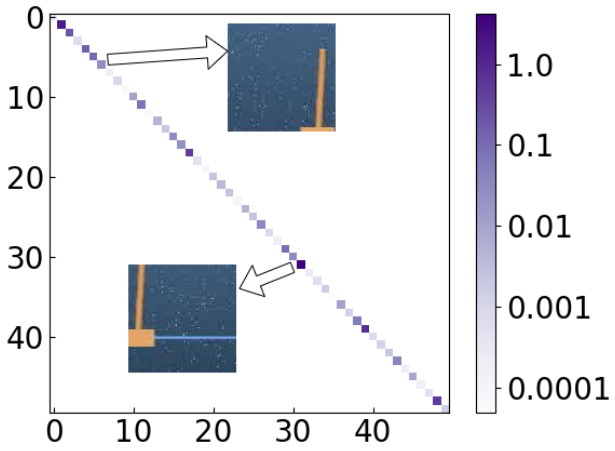


Figure 4.7: Relations of learned weights in latent **Q** matrix and original pixel states

matrix rank is a valuable clue for controller design. This demonstrates the benefit of having an interpretable representation of the state space.

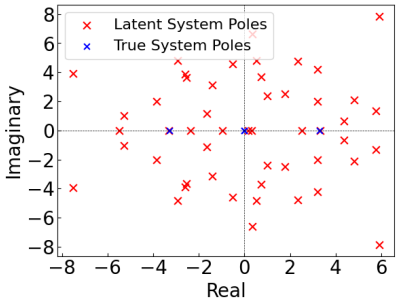


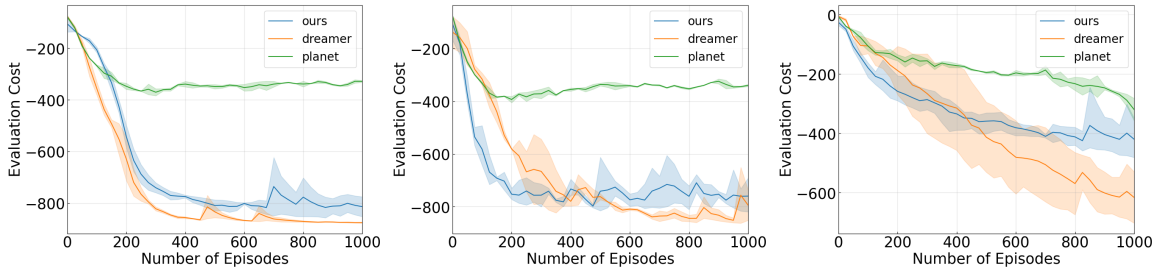
Figure 4.6: Pole-zero plot of true and learned latent CartPole systems.

Latent System Dimensions	Total Cost	Model Error
$\mathbf{Dim}(\mathbf{Z}) = 50$	-846.88	7.76×10^{-3}
$\mathbf{Dim}(\mathbf{Z}) = \text{rank}(W_{\mathbf{Z}}) = 6$	**-834.18**	$\mathbf{6.3 \times 10^{-3}}$
$\mathbf{Dim}(\mathbf{Z}) = 4$	-253.80	5.4×10^{-2}
CURL Control Performance	-841	-

Table 4.3: Our method achieves comparable control cost to CURL while providing more interpretable information about the system.

Interpretable Q Matrix in Latent Space. One key distinction of our method from CURL is the utilization of a structured LQR policy in the latent space. In Fig. 4.7, we illustrate that the LQR policy parameters, especially the Q matrix, can capture the relative significance weights of latent embedding and their relationship to the original pixel states.

We take the pixel-based CartPole task as an example. The larger diagonal elements in the learned Q matrix correspond to visual patches that contain the CartPole object, which provides interpretable information that captures useful latent information related to the CartPole object's area in the image is crucial for achieving a good controller.



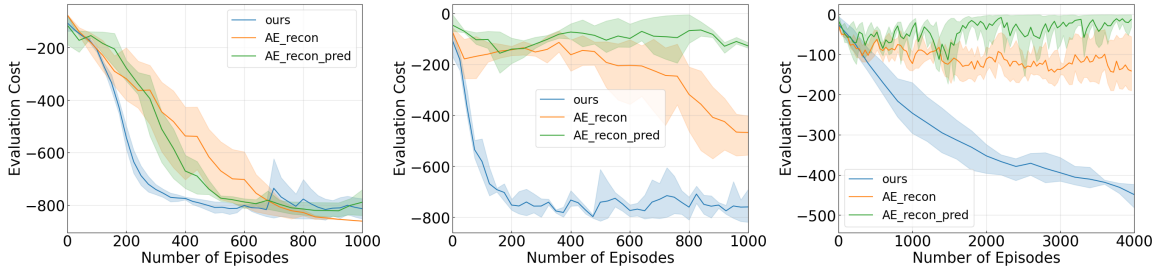
(a) Cart-Pole with Pixel Obsv (b) Cart-Pole with 4D state (c) Cheetah with 18D state

Figure 4.8: Comparison with well-known model-based RL methods.

4.3.4 Comparison with Other MBRL Methods

To benchmark our approach against established model-based RL methods, we select two widely recognized methods: PlaNet [48] and Dreamer (Version 2) [49]. This comparison spans three tasks. Remarkably, across all three tasks, our method demonstrates a clear superiority over PlaNet in terms of both data efficiency and peak performance. Additionally, our method achieves competitive performance with Dreamer-v2 in two Cart-Pole experiments, along with comparable data efficiency in cheetah running tasks. It's important to note that this comparable performance is achieved while our method learns a globally linear model, whereas Dreamer employs a much larger nonlinear network to approximate a world model. Therefore, our approach achieves a substantial reduction in both computational demands and structural complexity while not compromising control performance too much. It is also worth noting that our approach stands out from commonly used Model-Based RL methods as it's rooted in Koopman theory, offering the advantage of enabling control theory analysis for the system.

4.3.5 Comparison with Other Encoders and Losses



(a) Cart-Pole with Pixel Obsv (b) Cart-Pole with 4D state (c) Cheetah with 18D state

Figure 4.9: Comparison with autoencoder and varying losses.

We perform experiments to validate our selection of the contrastive embedding function. To achieve this, we replace the contrastive encoder with a canonical autoencoder (AE), a

commonly used method for learning condensed representations from high-dimensional observations. Specifically, we utilize the AE along with two types of loss functions: one involving only reconstruction loss, and another involving a combination of reconstruction and one-step prediction loss. Importantly, we keep all other aspects of our method unchanged. Results are based on 3 random seeds.

As shown in Figure. 4.9, our approach achieves comparable control performance with the use of AE. This aligns with the notion that autoencoder excels in reconstructing and representing pixel-based observations. However, when tasks involve non-pixel observations, such as normal states, our approach still maintains significant control efficiency and performance, while the AE-based structure struggles to learn a useful policy even with ample data. Particularly, we observed that the AE with only reconstruction loss slightly outperforms the one employing the combined loss, but still falls short of achieving the performance obtained by our method using the contrastive encoder. These results provide validation for our choice of contrastive embedding function within our approach.

4.3.6 Ablations Study of Hyper-parameters

We undertook an ablation study involving key parameters of the LQR solving iteration and Koopman embedding dimension. Results are the mean over 3 random seeds and summarized in Table. 4.4.

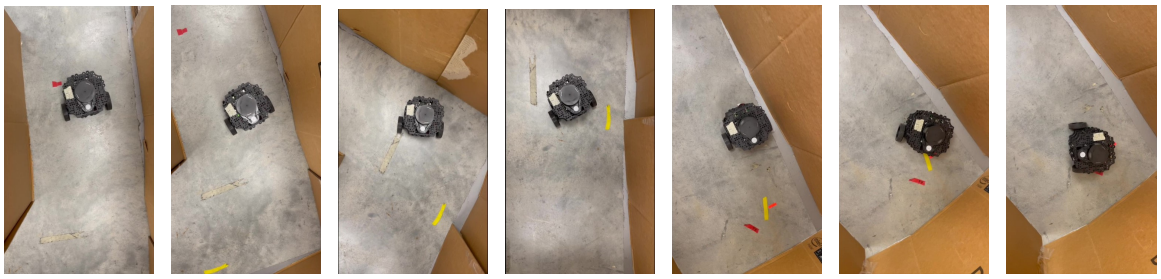
We found that our approach remains robust regardless of the specific number of iterations used for the LQR solution, as long as it falls within a reasonable range. This suggests that achieving a certain level of precision in solving the Riccati Equation contributes positively to both policy and linear model learning. We also studied the impact of varying the latent embedding dimensions for the encoder. Our findings indicate that using a smaller dimension that aligns with the encoder's intermediate layers yields consistently good results, while excessively increasing the embedding dimension (d=100) can diminish control performance. One reason could be the reduced approximation capabilities of the encoder due to inappropriate high latent dimension. It might also be because the latent state becoming overly sparse and failing to capture crucial information for effective control.

4.3.7 Real-World Evaluation

In this real robot task, we deploy our algorithm trained from the Gazebo simulator to the turtlebot3 burger ground robot. We use 2D Lidar measurements as well as the odometry information as observation, and the linear LQR policy generates the linear and angular velocities as control. We aim to control the robot to navigate through a narrow curved path without any collisions. We directly transfer the trained policy (which is trained with only 40 episodes and each episode contains around 700 steps) to the hardware without any fine-tuning, indicating the applicability of our approach to a real robot.

		LQR iteration			Latent Dimension		
		iter=3	iter=5*	iter=10	d=30	d=50*	d=100
CartPole pixel	control cost	-863	-873	-835	-848	-873	-813
	model error	0.075	0.058	0.269	0.165	0.058	0.038
CartPole state	control cost	-751	-762	-769	-777	-762	-667
	model error	0.00047	0.00031	0.00042	0.00043	0.00031	0.0002
cheetah run	control cost	-436	-464	-448	-477	-464	-235
	model error	0.732	0.862	0.842	0.653	0.862	0.046

Table 4.4: Ablation results of final cost and model-fitting error regarding LQR solving iteration and Koopman latent state dimension. The asterisk refers to the parameter used in this work.



(a) 4s (b) 9s (c) 15s (d) 20s (e) 26s (f) 29s (g) 30s

Figure 4.10: Snapshots of real robot curved trajectory at different time stamps (seconds).

4.4 Chapter Summary

In this work, we propose task-oriented Koopman-based control with a contrastive encoder to enable simultaneous learning of the Koopman embedding, model, and controller in an iterative loop which extends the application of Koopman theory to high-dimensional, complex systems. The significance of this work lies in (1) It reduces the reliance of controller design on a well-identified model by prioritizing the task cost as the main objective for controller learning, which, for the first time to the best of our knowledge, extends the classical Koopman control from low to high-dimensional, complex nonlinear systems, including pixel-based tasks and a real robot with lidar observations. (2) It enhances the interpretability of the learning process, which is usually not fully addressed. Specifically, it integrates a linear classical controller and a linear latent model into the RL loop, enabling the theoretical analysis (e.g. stability, controllability) of the acquired system model via control theory. It also allows for the interpretability of part of the learning process, such as the relations between latent control coefficients $\mathbf{Q}$ and the learned behaviors).

Chapter 5

Efficient Multi-Agent Centralized Learning with Asynchronous Options

This chapter is based on my paper "Asynchronous, Option-Based Multi-Agent Policy Gradient: A Conditional Reasoning Approach" [86][1] written in collaboration with Amin Banitalebi-Dehkordi, Mo Chen, and Yong Zhang.

5.1 Chapter Overview and Related Work

Cooperative multi-agent problems are common in the real world. For example, a team of warehouse agents must coordinate their behaviors for efficient cargo handling, or a group of robots to achieve a common goal in a team-based game. In these cases, agents' actions and outcomes affect each other, thus a centralized controller is often needed to learn a policy that takes the joint observation of all agents as input and outputs a joint action for all agents [46, 40, 167].

To efficiently learn a cooperative, centralized policy, multi-agent policy gradient (MAPG) methods have been widely applied and have many successful applications [108, 37, 171, 156, 154, 151]. One key feature of MAPG methods is that they normally require robots to perform low-level actions at a low-level time scale: an action only lasts one time step and multiple agents choose their actions synchronously at every time step. Although low-level actions can be useful, they are not always practical for solving complex problems especially those with large state and action space. This is because relying solely on low-level actions may lead to a combinatorial explosion, where there are too many possible actions to search, making it difficult to find an optimal policy.

To improve policy search efficiency, hierarchical reinforcement learning (HRL) has been developed and widely adopted [90, 68, 134, 39, 166]. HRL breaks down complex tasks into

[1]Supplementary contents: https://sites.google.com/view/mahrlsupp

smaller subtasks to reduce the problem's complexity, making it easier to search for policies. This is especially helpful for long-term reasoning tasks with sparse reward feedback. One typical approach of HRL is the options framework [134], which employs options to optimize policies. Options are higher-level combinations of primitive actions that enable agents to perform actions over an extended duration. The use of options effectively reduces search space size and improves policy learning efficiency and generalisability [169, 148, 128].

In order to effectively address complex cooperative problems, it is advantageous to employ the option framework in MAPG methods for centralized learning. However, a significant challenge arises from the inconsistency between asynchronous option execution and centralized decision-making. This inconsistency results from the fact that options have varying low-level time lengths, leading to their selection and completion across all agents occurring at different time steps, that is, "asynchronicity" of multi-agent option execution. While preserving asynchronicity can enable a realistic and efficient option execution strategy, as illustrated in Fig. 5.1, it also poses an obstacle to MAPG methods in determining when and how to optimize a joint control policy and sample new options from it for all agents. Specifically, under asynchronicity, the joint option sampled from a centralized policy may not be fully executed. For example, at a given low-level time step, only a subset of agents may need to choose new options, while others do not. Therefore, it is unclear how to properly evaluate policy gradients for policy training.

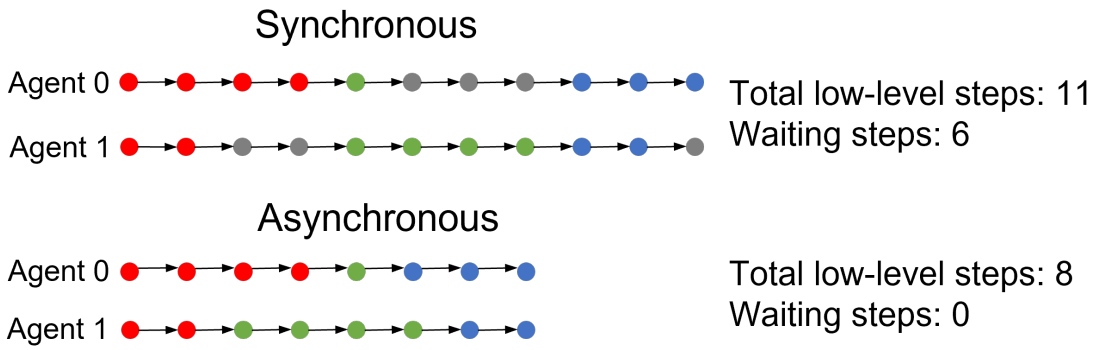


Figure 5.1: An illustration of synchronous and asynchronous option execution. Each node refers to a state. Each arrow refers to a low-level action. A sequence of nodes in red, green, or blue refers to executing one option. The gray nodes are waiting states for option synchronization. An asynchronous strategy is more efficient and requires much fewer low-level waiting steps.

Several attempts [165, 169, 50, 3, 164] have been made to address the aforementioned challenge, but they have limitations. Specifically, these attempts can be classified into two categories: synchronous and asynchronous approaches. Synchronous approaches [169, 50] force all agents' options to be synchronized at certain low-level time steps. For example, they either use η_{any} strategy that interrupts unfinished options when *any* one of the agents

finishes its option, or use η_{all} strategy that waits until *all* agents' options to finish. Then the standard MAPG method can be applied to make centralized decisions as if they are working on synchronous actions with an "extended step". However, this type of approach destroys asynchronicity and eliminates the option completeness as well as its benefits of low-level temporal abstraction. In contrast, Christopher et al. [3, 164] proposed an alternative, asynchronous solution. It allows the use of η_{continue} strategy: a subset of agents choose new options while the others continue their ongoing options. They achieve this by aligning multi-agent options via a specially designed sequence filtering process. This method preserves asynchronicity but is only compatible with value-based algorithms (e.g., Deep Q-Network (DQN)[95]), not policy gradient algorithms.

In this work, we propose a novel, conditional reasoning approach to enable centralized learning of MAPG over asynchronous options. To achieve this, we first introduce a special type of trajectory named "option-level joint trajectory", which combines multi-agent independent trajectories into one single joint trajectory. This trajectory allows multiple robots to reason and plan at an option level while preserving efficient, asynchronous option execution.

To generate such a trajectory for policy training, we formalize a conditional centralized policy that only selects a subset of options for those agents who require new options but conditions its policy distribution on the currently executing options. In this way, we address the inconsistency between option asynchronicity and centralized decision-making. As a result, our method can seamlessly adapt MAPG from action to option-wise while fitting it into centralized learning. Our method is simple and can be easily implemented as an add-on to standard, action-based policy gradient algorithms, such as MAPPO [171], to extend its use over asynchronous options. We validate the effectiveness of our method on two option-based multi-agent cooperative learning tasks.

5.2 A Conditional-Reasoning Approach for Multi-Agent Centralized Learning over Asynchronous Options

5.2.1 Problem Formulation

Following option framework [134], we formulate a two-level, multi-agent HRL problem for cooperatively training N agents. In the higher level, at the option time step κ (its corresponding low-level time step k), an option-based joint policy $\pi(\mathbf{o}_\kappa|\mathbf{s}_\kappa) \in [0,1]$ needs to be trained to determine a joint option $\mathbf{o}_\kappa$ based on a joint observation $\mathbf{s}_\kappa$ (under true state $\mathbf{s}_\kappa$) for N agents. $\mathbf{o}_\kappa = (\mathbf{o}_\kappa^0, \mathbf{o}_\kappa^1, ...\mathbf{o}_\kappa^{N-1})$ and $\mathbf{s}_\kappa = (\mathbf{s}_\kappa^0, \mathbf{s}_\kappa^1, ...\mathbf{s}_\kappa^{N-1})$ are collections of each agent i's option and observation, where i is indexed by $i = \{0, 1, 2, ..., N-1\}$. In the lower level, each agent observes the local observation $\mathbf{s}_k^i$ under the true state $\mathbf{s}_k^i$ and takes an action $\mathbf{a}_k^i$ which is determined by the inner policy $\pi_{\mathbf{o}_\kappa^i}(\mathbf{a}_k^i|\mathbf{s}_k^i)$ of current-executing option $\mathbf{o}_\kappa^i$. After all agents take their actions, the environment transits from $\mathbf{s}_k$ to the next state $\mathbf{s}_{k+1}$ according to $p(\mathbf{s}_{k+1}|\mathbf{s}_k, \mathbf{a}_k)$ and achieve a joint reward $r(\mathbf{s}_k, \mathbf{a}_k, \mathbf{s}_{k+1})$. In our setup, the

transition function $p(\mathbf{s}_{k+1}|\mathbf{s}_k, \mathbf{a}_k)$ is based on a low-level time step and is determined by the simulation.

Our objective is to find the higher-level, option-based policy $\pi(\mathbf{o}|\mathbf{s})$, rather than the inner policy $\pi_{\mathbf{o}}$ over lower-level action space. Therefore, we assume the inner, action-based policies $\pi_{\mathbf{o}}$ of available options are given or pre-defined that are appropriate for potential task completion. This is to say, we are not learning $\pi_{\mathbf{o}}$ at the low level.

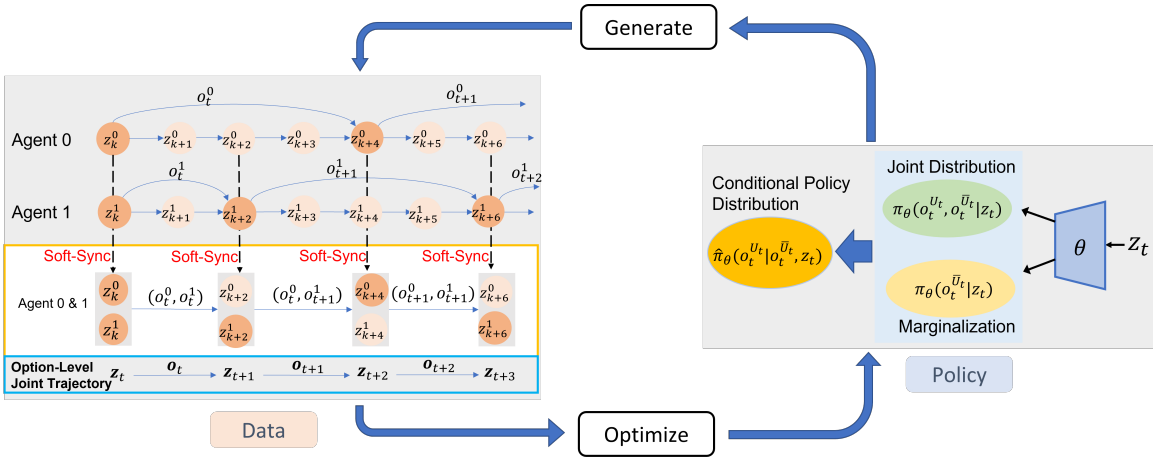


Figure 5.2: The structure of our method. **Left:** an example of the option-level joint trajectory where two agents take their options asynchronously. Observations $\mathbf{s}_\kappa, \mathbf{s}_{\kappa+1}, ...$ are joint observations of all agents at option step $\kappa, \kappa+1, ...$ when at least one agent's option is terminated. Option $\mathbf{o}_\kappa, \mathbf{o}_{\kappa+1}, ...$ are joint options at each option step. Batches of these trajectories are used as training data to optimize the conditional centralized policy. **Right:** The formation of centralized conditional policy $\hat{\pi}_\theta$ where θ is the policy parameter. This policy is used to generate the required joint options for a subset of agents U_κ to form the option-level trajectories.

5.2.2 Centralized MAPG over Asynchronous Options

In centralized learning, asynchronous option execution presents a problem for policy gradient optimization. To understand the problem, let us consider Generalized Advantage Estimation (GAE)-based policy gradient [122] shown in Eq. (5.1). It describes a centralized controller selecting the joint action a_k for all agents at the low-level time scale indexed by k. It calculates the policy gradient of the RL objective $J(\pi_\theta)$ over a batch of trajectories τ. G_k^λ estimates the Temporal-Difference TD(λ) [130] return and V_{π_θ} estimates the value function which is used as a baseline. Our question is: *how to adapt Eq. (5.1), which is based on actions, to the use of options, and apply it to learn a centralized policy?*

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{k=0}^{K} \nabla_\theta \log \pi_\theta(a_k \mid \mathbf{s}_k)\left(G_k^\lambda - V_{\pi_\theta}(\mathbf{s}_k)\right)\right]. \tag{5.1}$$

One trivial way is to simply substitute action-based policy $\pi(\mathbf{a}_k|\mathbf{s}_k)$ with option-based policy $\pi(\mathbf{o}_\kappa|\mathbf{s}_\kappa)$ and view an option as an action with "one extended step". In this way, Eq. (5.1) still holds, and can be converted to base on option time κ defined by the option termination, rather than action time k. However, this only works when multi-agent options are forced to be synchronized (e.g. η_{any} or η_{all}) such that a joint policy can always select new options for all agents at the same time. However, it is not applicable to many realistic situations where the asynchronous option execution η_{continue} is preferred. In fact, the calculation for policy term $\pi(\mathbf{o}_\kappa|\mathbf{s}_\kappa)$ remains a challenge when η_{continue} strategy is used, as there is no "synchronous timing" for a joint option selection anymore. Therefore, we present a novel solution to address it in the next section.

5.2.3 Option-Level Joint Trajectory

To train a multi-agent option-based policy in a centralized way, we require a collection of agents' trajectories as training data. Unlike action-level trajectory that is often defined at a sequence of low-level time steps, here we introduce a special option-level trajectory in Fig. 5.2. To form it, we determine specific low-level timings at when we collect joint states and observations, joint options, and the shared reward for all agents. Those timings are when *at least one agent* terminates its previous option and selects a new option, and they will serve as option time steps in the trajectory. We refer to this process as "soft-sync" because it synchronizes multi-agent state transitions and option selections in a soft way — it does not actually interrupt agents' ongoing options.

In contrast to strategies like η_{any} or η_{all} that actually interrupt robot-environment interactions to have forced synchronization, "soft-sync" allows the formation of an "interruption-free" trajectory over asynchronous options, and allows the use of a centralized policy to jointly select options. The trajectory is formally described by Definition 1.

Definition 1. We denote $\tau_{\mathbf{o}}$ as an option-level trajectory which is the joint trajectory for N agents. $\tau_{\mathbf{o}} = (\mathbf{s}_\kappa, \mathbf{o}_\kappa, r_\kappa, \mathbf{s}_{\kappa+1}, \mathbf{o}_{\kappa+1}, r_{\kappa+1}, ..., \mathbf{s}_{\kappa+T})$ where $\kappa, \kappa+1, ..., \kappa+T$ are consecutive option steps when "soft-sync" happens. In particular, $\mathbf{s}$ and $\mathbf{o}$ are joint observation and option: $\mathbf{s}_\kappa = (\mathbf{s}_\kappa^0, \mathbf{s}_\kappa^1, ..., \mathbf{s}_\kappa^{N-1})$ and $\mathbf{o}_\kappa = (\mathbf{o}_\kappa^0, \mathbf{o}_\kappa^1, ...\mathbf{o}_\kappa^{N-1})$. r_κ is the shared step reward.

5.2.4 Conditional Centralized Policy

To achieve centralized MAPG over asynchronous options, we not only require option-level joint trajectories $\tau_{\mathbf{o}}$ but also a special form of policy to generate these trajectories and get updated from them. This is because many policy gradient methods [122, 117, 171] update policy in an *on-policy* way: the policy being updated is also used to generate sequential data for training itself.

Although sampling new options for all agents from a joint policy $\pi(\mathbf{o}_\kappa|\mathbf{s}_\kappa)$ at each time step κ is the easiest approach, it is infeasible. The reason is that only a subset of agents need

to select new options at each option step κ of an option-level joint trajectory, while the others need to continue with their ongoing options to maintain "interruption-free" asynchronicity. For example, in Fig. 5.2, at option step $\kappa + 1$, agent 1 requires a new option, but agent 0 does not and continues with its ongoing option. Thus, a policy would be desirable if it could generate joint options at each option step only for those agents who need new options while not affecting the other agents with ongoing options.

Formally, we assume the existence of a centralized policy, denoted by $\pi(\mathbf{o}_\kappa|\mathbf{s}_\kappa)$, which takes the joint observation $\mathbf{s}_\kappa$ of all agents as input and outputs a probabilistic distribution of joint option $\mathbf{o}_\kappa$. For mathematical convenience, we introduce a new notation, U_κ, to denote the set of agents that need to choose new options at option step κ, and $\overline{U}_\kappa$ to denote the set of agents that do not need to choose new options. The corresponding joint option for U_κ and $\overline{U}_\kappa$ are denoted as $\mathbf{o}_\kappa^{U_\kappa}$ and $\mathbf{o}_\kappa^{\overline{U}_\kappa}$ respectively.

It turns out that we can take advantage of the concept of *conditional dependency* to derive the policy distribution we need. More specifically, we formalize a joint policy $\pi(\mathbf{o}_\kappa|\mathbf{s}_\kappa)$ in an equivalent way of conditioning the option distribution of the agents in U_κ based on the currently executing options of agents in $\overline{U}_\kappa$, as is shown in Eq. (5.2).

$$\pi(\mathbf{o}_\kappa \mid \mathbf{s}_\kappa) = \pi(\mathbf{o}_\kappa^{U_\kappa}, \mathbf{o}_\kappa^{\overline{U}_\kappa} \mid \mathbf{s}_\kappa) \triangleq \hat{\pi}(\mathbf{o}_\kappa^{U_\kappa} \mid \mathbf{o}_\kappa^{\overline{U}_\kappa}, \mathbf{s}_\kappa) \tag{5.2}$$

In Eq. (5.2), we denote $\hat{\pi}$ as a conditional policy to distinguish it from joint policy π, but they share the same parameters. The use of $\hat{\pi}(\mathbf{o}_\kappa^{U_\kappa} \mid \mathbf{o}_\kappa^{\overline{U}_\kappa}, \mathbf{s}_\kappa)$ allows us to sample options only for those agents who need to select new options while maintaining uninterrupted option executions for the others. Thus, it allows the generation of "true" asynchronous trajectory $\tau_\mathbf{o}$. Moreover, the generated trajectories $\tau_\mathbf{o}$ will be further used as training data to update the policy itself, we thus achieve the on-policy update.

5.2.5 Algorithm

In practice, Eq. (5.2) is computationally feasible. The term $\hat{\pi}(\mathbf{o}_\kappa^{U_\kappa} \mid \mathbf{o}_\kappa^{\overline{U}_\kappa}, \mathbf{s}_\kappa)$ is a conditional distribution thus can be reformed via Bayes' rule by a joint distribution $\pi(\mathbf{o}_\kappa^{U_\kappa}, \mathbf{o}_\kappa^{\overline{U}_\kappa} \mid \mathbf{s}_\kappa)$ and its marginalization $\pi(\mathbf{o}_\kappa^{\overline{U}_\kappa} \mid \mathbf{s}_\kappa)$, as shown in Eq. (5.3).

$$\mathbf{o}_\kappa^{U_\kappa} \sim \hat{\pi}(\mathbf{o}_\kappa^{U_\kappa} \mid \mathbf{o}_\kappa^{\overline{U}_\kappa}, \mathbf{s}_\kappa) = \frac{\pi(\mathbf{o}_\kappa^{U_\kappa}, \mathbf{o}_\kappa^{\overline{U}_\kappa} \mid \mathbf{s}_\kappa)}{\pi(\mathbf{o}_\kappa^{\overline{U}_\kappa} \mid \mathbf{s}_\kappa)} \tag{5.3}$$

The joint and marginalized terms can be effortlessly represented and obtained. In many deep RL cases, neural networks are often used to represent policy and value functions. The joint policy $\pi_\theta(\mathbf{o}_\kappa^{U_\kappa}, \mathbf{o}_\kappa^{\overline{U}_\kappa} \mid \mathbf{s}_\kappa)$ can be represented by a neural network parameterized by θ, and its output usually stands for the main parameters of commonly-used probabilistic distributions such as categorical or Gaussian distribution. From these standard distributions, the marginalization $\pi_\theta(\mathbf{o}_\kappa^{\overline{U}_\kappa} \mid \mathbf{s}_\kappa)$ can also be efficiently computed. For example, the

marginalization of categorical distribution can be done by summing out the variables corresponding to the categorical distribution to obtain a new distribution over the remaining variables. This process is illustrated in Fig. 5.2's policy stage.

By utilizing $\hat{\pi}_\theta(\mathbf{o}_\kappa^{U_\kappa} \mid \mathbf{o}_\kappa^{\overline{U}_\kappa}, \mathbf{s}_\kappa)$ as policy and running its optimization over option-level joint trajectory $\tau_{\mathbf{o}}$, we can seamlessly alternate standard MAPG method in Eq. (5.1) to support the centralized training over asynchronous options, as shown in Eq. (5.4). Specifically, the centralized critic $V_{\pi_\theta}(\mathbf{s}_\kappa)$ and TD(λ) return G_κ^λ are computed in the same way as usual, but only based on option-level trajectory. We summarize the entire process in Algo. 3.

$$\nabla_\theta J\left(\hat{\pi}_\theta\right) = \mathop{\mathrm{E}}_{\tau_o \sim \hat{\pi}_\theta}\left[\sum_\kappa \nabla_\theta \log \hat{\pi}_\theta\left(\mathbf{o}_\kappa \mid \mathbf{s}_\kappa\right)\left(G_\kappa^\lambda - V_{\hat{\pi}_\theta}(\mathbf{s}_\kappa)\right)\right] \tag{5.4}$$

Algorithm 3: Asynchronous Option-Based MAPG

Initialize N agents. Initialize a joint policy $\pi_\theta(\mathbf{o}|\mathbf{s})$ and a value network $V_\phi(\mathbf{s})$. Randomly sample initial observation $\mathbf{s}_0$.
for iteration $i = 0, 1, 2, ...$ **do**
  Training trajectory dataset $\tau_{\mathbf{o}} = \{\}$.
  for $\kappa = 0, 1, 2, ..., T$ **do**
    if $U_\kappa \neq \varnothing$ (some agents choose new options) **then**
      Obtain $\mathbf{o}_\kappa^{U_\kappa}$ and $\mathbf{o}_\kappa^{\overline{U}_\kappa}$, and alternate $\pi_\theta(\mathbf{o}_\kappa \mid \mathbf{s}_\kappa)$ to be $\hat{\pi}_\theta(\mathbf{o}_\kappa^{U_\kappa} \mid \mathbf{o}_\kappa^{\overline{U}_\kappa}, \mathbf{s}_\kappa)$ via Eq. (5.2) and (5.3).
      Sample $\mathbf{o}_\kappa$ from $\hat{\pi}_\theta(\mathbf{o}_\kappa^{U_\kappa} \mid \mathbf{o}_\kappa^{\overline{U}_\kappa}, \mathbf{s}_\kappa)$ and execute $\mathbf{o}_\kappa$.
      Add $(\mathbf{s}_\kappa, \mathbf{o}_\kappa, r_\kappa)$ to $\tau_{\mathbf{o}}$.
    else
      all agents continue their ongoing options.
    end if
  end for
  Update policy parameter θ via Eq. (5.4) using data from $\tau_{\mathbf{o}}$.
end for

5.2.6 An Extended Use-Case

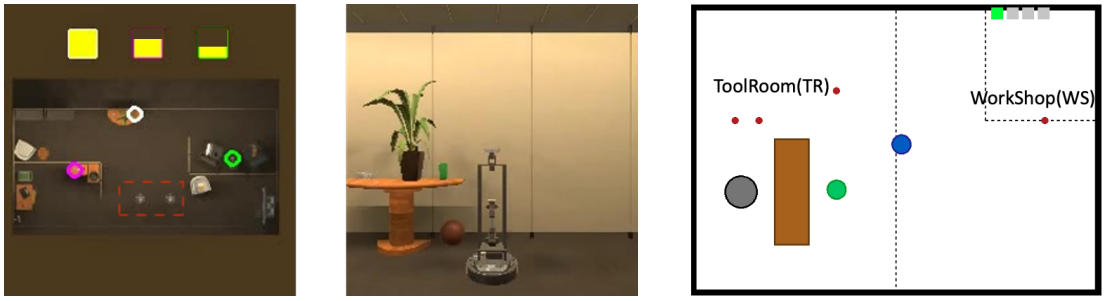
In addition to centralized learning, our method can also be applied to another important use case called partially centralized learning. In contrast to (fully) centralized scenarios where global states and observations are needed to make global decisions, partially centralized learning describes agents choosing their options based on their local observations as well as conditioned by other agents' options.

Partially centralized learning is useful in some realistic situations when having a fully centralized system is impracticable. For example, broadcasting global information for all agents can sometimes be costly and restricted by software or hardware capabilities such as communication bandwidth and latency, especially when global information includes high-

dimensional observations (e.g. images or point cloud). However, by only sharing options instead of observations, partially centralized learning can be more efficient since the option space is usually low-dimensional. In addition, knowing the options of other agents as conditions is potentially important for local decision-making.

Formally, we have multiple policies π^i, one for each agent i for option selection. Similar to centralized learning, at each option step κ, we formulate the option selection probability of each agent in U_κ by conditioning it on its local observation $\mathbf{s}^i_\kappa$ as well as the currently executing options of agents in $\overline{U}_\kappa$. For agents in the set $\overline{U}_\kappa$, we can simply evaluate its option selection probability to be deterministic and equal to 1. This is because the agent does not select a new option but naturally continues the ongoing option, thus not contributing to the policy gradient at this option step. This is shown in Eq. (5.5).

$$\hat{\pi}^i(\mathbf{o}^i_\kappa \mid \mathbf{o}^{\overline{U}_\kappa}_\kappa, \mathbf{s}^i_\kappa) = \begin{cases} 1, & \text{if } i \in \overline{U_\kappa} \\ \frac{\pi(\mathbf{o}^i_\kappa, \mathbf{o}^{\overline{U}_\kappa}_\kappa \mid \mathbf{s}^i_\kappa)}{\pi(\mathbf{o}^{\overline{U}_\kappa}_\kappa \mid \mathbf{s}^i_\kappa)}, & \text{otherwise.} \end{cases} \tag{5.5}$$



(a) (b)

Figure 5.3: (a) Water Filling task, top and front view. Water levels are in yellow. The positions of the three jars are circled in white, pink, and green. Two robots are in the middle space (red dashed box). (b) Tool Delivery task with one Fetchbot (gray) and two Turtlebots (green and blue). Red dots are available at middle waypoints for transitions, and the brown rectangle is a desk for passing tools. The progress bar on the top right represents the current task stage.

5.3 Results

5.3.1 Task Specifications

We empirically evaluate our method on two cooperative multi-robot tasks featuring in asynchronous options and long-term reasoning, following previous work [165, 164].

Water Filling (WF). As shown in Fig. 5.3a, two heterogeneous robots (a slower vehicle and a faster drone) navigate in a large indoor room and cooperate to refill water timely for three jars at different locations. This task is challenging as (1) the water levels in three jars

decrease randomly and rapidly following parameter-varying Gaussian distributions at each low-level step k; (2) two robots have different but limited abilities: vehicle can scout and fill water but move slowly; the drone is faster but can only scout water levels. So, robots have to develop "real" cooperation to make use of their abilities rather than repeating rote behaviors. For example, the drone is expected to scout the room quickly and share the most recent water levels with the vehicle. The vehicle can choose the closest or empty jar to fill.

We use ai2thor simulator [63] to model this task and use RGB raw images as well as important auxiliary information[2] as observations. Both agents can take one-step options: {up; down; left; right} or multi-step options: NavTo(j) where j is jar index. A vehicle can take an extra option NavToFill(j) to reach and fill water. The low-level time steps of NavTo(j) and NavToFill(j) depend on the robot's speed and its distance to the target jar: we use 0.5m/s for the vehicle and 2m/s for the drone. The reward function is defined over the global water level of all jars in Eq. (5.6) where $w(\mathbf{s}; j)$ is the water level of jar indexed by j under true state $\mathbf{s}$.

$$r(\mathbf{s}) = \sum_{j=0}^{|J|}(-\frac{1.0}{w(s,j)+0.001}+1), \mathrm{j} = 0, 1, 2 \tag{5.6}$$

Tool Delivery (TD). To validate the usefulness of our method on more complex problems, we use Tool Delivery [164], a three-agent task that requires complicated multi-stage collaboration (Figure. 5.3b). It involves a human working on a four-stage task, requiring assistance from a Fetchbot and two Turtlebots to search, pass, and deliver proper tools to him at the proper stage. Only Fetchbot can search and pass tools to Turtlebots, and only Turtlebots can deliver tools to a human. Turtlebots choose multi-step options from {GoToWS; GoToTR; GetTool} to go to one of the middle waypoints (TR or WS, red dots in Fig. 5.3b) or pick one tool. Fetchbot either searches specific tool or passes one of the found tools to one of the waiting Turtlebots. More constraints of this task can be found in [164].

Turtlebot observes its own location, human working stage, index of carried tools, and the number of the tools on the desk. Fetchbot observes the number of tools waiting to be passed and the index of waiting Turtlebot. Note that neither Fetchbot nor Turtlebots is aware of the correct tool required by a human at each stage, they have to reason about this information via training. The agents receive a -1 reward at each low-level step, -10 when the Fetchbot passes a tool but no Turtlebot around, and 100 for a good tool delivery to a human. Available options for all tasks are shown in Table 5.1.

[2]In particular, we use (a) global water levels recently updated by any robot; (b) elapsed time steps since water levels get updated last time.

Task	Agent	Available Options
WF	Drone	Up; Down; Left; Right; NavTo(j);
WF	Vehicle	Up; Down; Left; Right; Fill(j); NavTo(j); NavToFill(j);
TD	Turtlebot	GoToWS; GoToTR; GetTool
TD	Fetchbot	SearchTool(h); PassTo(w)

Table 5.1: Available agent options for two tasks. Options are discrete. j is the index of jars from the WF task. $h \in \{0,1,2\}$ is the tool index and $w \in \{0,1\}$ is the index of waiting Turtlebot from TD task. Options have varying low-level time lengths depending on their termination conditions and properties.

5.3.2 Implementation and Baselines

This section outlines the methods employed in our study, which includes our proposed approach, baselines, and a state-of-the-art method. In all methods, a joint policy network is used for centralized policy learning across all agents.

Our method and baselines are both adapted from MAPPO [171], a well-known, action-based MAPG algorithm. Our method alters MAPPO to allow the use of a conditional centralized policy and use it to sample asynchronous, option-level joint trajectories for training. Baseline methods also use option-level trajectories as training data, but the trajectories are collected synchronously. They apply use synchronous strategies such as η_{any} and η_{all} to forcibly select options for all agents, and collect $\{(\mathbf{s}_\kappa, \mathbf{o}_\kappa, r_\kappa) \mid \kappa = 0, 1, ..., T\}$ at every option step. We denote these methods as *"sync-cut"* and *"sync-wait"* respectively.

We include an *"end2end"* baseline which uses MAPPO to learn a centralized policy purely over low-level actions rather than options. Additionally, we include a state-of-the-art approach that is also applicable to "interruption-free", asynchronous option-based data, but mainly uses Q-value-based off-policy algorithms. We use mean reward over a batch of training samples per iteration as performance criteria.

For all methods, we set the training batch size to be based on a low-level action step instead of a high-level option step. During each training epoch, a fixed batch size of low-level samples is collected. For the WF task, the batch size is 2000 while for the TD task, the batch size is 200. Despite the variation in the number of option steps in a batch due to different option termination strategies. we ensure a fair comparison of "convergence rate v.s. sample size" (Fig. 5.4) between different methods, particularly between option-based and action-based methods, as an option often contains more than one action.

5.3.3 Performance Comparison with Baselines

Fig. 5.4a shows the actual training progress of ours and baseline methods on the Water Filling task. Our approach, denoted by *"async(ours)"*, achieves consistently increasing mean reward and outperforms other baselines from the start to the end. This indicates the

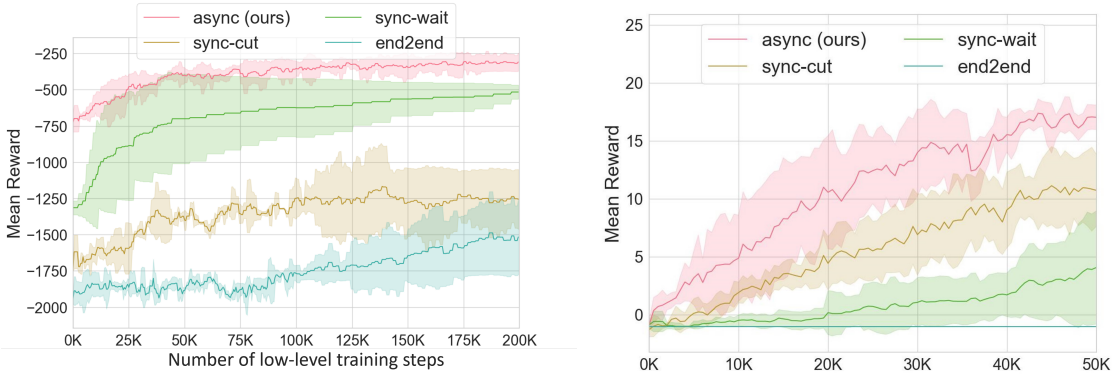


Figure 5.4: Centralized policy learning using asynchronous option-based (ours), synchronous option-based as well as low-level action-based methods on (a) Water Filling and (b) Tool Delivery task. All experimental results are summarized over 5 runs with random seeds. During each training epoch, we gather a fixed number of low-level samples and extract option-based trajectory data from them to update the policy. The variation of starting reward levels in (a) is attributed to the use of different option termination strategies, which can impact the number of option steps included in a trajectory. Thus, the variation in trajectory length influences the cumulative reward attained.

effectiveness of our centralized, conditional-based policy gradient on using complete asynchronous options, which is beneficial to long-term planning, especially in the large indoor environment, otherwise the interrupted options (e.g. using *"sync-cut"*) often give rise to navigation failure in the middle of path and waiting other agents' options can lead to very inefficient exploration process (e.g. using *"sync-wait"*, drone has to wait for slower vehicle and cannot scout the environment information quickly and frequently).

In Fig. 5.4b, we show the training results on the Tool Delivery task. Clearly, our approach still acquires the highest peak reward as well as the fastest convergence rate in contrast to the other baselines, despite more agents, stages, and cooperative structures involved. This again indicates the advantage of being capable of running MAPG on complete, asynchronous options over forced synchronized options. What even worse is, that without option-level operation, the "end2end" method cannot learn any useful policy at all.

5.3.4 Performance Comparison with a state-of-the-art method

To show the effectiveness of our method not only from running complete options but also the method itself, we present a Table. 5.2, which summarizes the results of the final policy trained by ours and a state-of-the-art method "Mac-DQN" [164], in terms of its peak performance and sample efficiency. For both methods, we use mean reward as a performance criterion. It indicates that despite both methods using the "true" asynchronous option with-

out any forced synchronization, we achieve not only a comparable level of peak performance but also much better sample efficiency than "Mac-DQN" on both tasks (3∼8x faster).

	Peak Performance		Training Samples	
	Mac-DQN	Ours	Mac-DQN	Ours
Water Filling	-287 ± 41	$\mathbf{-304 \pm 27}$	600K	**200**K
Tool Delivery	14.2 ± 0.8	$\mathbf{17.1 \pm 0.7}$	400K	**50**K

Table 5.2: Performance comparison with Mac-DQN, a state-of-the-art asynchronous option-based MAPG method using Q-value-based, off-policy optimization.

5.3.5 Centralized Learning: Fully v.s. Partially

We evaluate the performance of policies trained by fully and partially centralized learning, as shown in Table. 5.3. It was found that partially centralized learning achieved relatively close performance to centralized learning in the Water Filling task, but performed significantly worse in the Tool Delivery task. This may be because in the Water Filling task, important task states such as the global water level can be effectively communicated between robots and encoded in their local observations, and thus having only local observations is sufficient for robots to navigate and search effectively. However, in the multi-stage Tool Delivery task, more complex cooperation and global observation sharing is necessary to obtain important task states for all agents. Therefore, using partially centralized learning is not sufficient for making good decisions based solely on local observations. Overall, partially centralized learning may be useful in cases where observation sharing is less critical but option-dependency is more important, as it can reduce computational load.

	Partially Centralized	Centralized (Ours)
Water Filling	-349 ± 51	$\mathbf{-304 \pm 27}$
Tool Delivery	4.4 ± 0.3	$\mathbf{17.1 \pm 0.7}$

Table 5.3: Performance comparison between partially centralized and centralized learning. Both use asynchronous options.

5.4 Chapter Summary

In this work, we propose a new approach to facilitate centralized learning in the context of MAPG with asynchronous options while preserving the asynchronous execution of these options. To accomplish this, we introduce a unique type of trajectory known as the "option-level joint trajectory". This trajectory combines the individual trajectories of multiple agents into a single, unified trajectory. To generate such a trajectory for policy training, we formalize a conditional centralized policy that selects a specific subset of options for agents in

need of new options but conditions its policy distribution on the currently executing options. This approach effectively reconciles the inconsistency between the asynchronicity of options and centralized decision-making. Consequently, our method seamlessly adapts off-the-shelf MAPG (e.g. MAPPO [171]) from an action-wise to an option-wise perspective. In the current work, we use pre-defined, perfect options, but alternatively, the options can be pre-trained offline ahead as a set of subtasks, or they can use low-level classical controllers.

Bibliography

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[2] Ian Abraham, Gerardo De La Torre, and Todd D Murphey. Model-based control using koopman operators. *arXiv preprint arXiv:1709.01568*, 2017.

[3] Christopher Amato, George Konidaris, Leslie P Kaelbling, and Jonathan P How. Modeling and planning with macro-actions in decentralized pomdps. *Journal of Artificial Intelligence Research*, 64:817–859, 2019.

[4] Robin Amsters and Peter Slaets. Turtlebot 3 as a robotics education platform. In *International Conference on Robotics in Education (RiE)*, 2019.

[5] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997.

[6] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings, 2012.

[7] Martino Bardi, Italo Capuzzo Dolcetta, et al. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*, volume 12. Springer, 1997.

[8] Martino Bardi, Italo Capuzzo Dolcetta, et al. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*, volume 12. Springer, 1997.

[9] Martino Bardi and Pierpaolo Soravia. Hamilton-jacobi equations with singular boundary conditions on a free boundary and applications to differential games. *Transactions of the American Mathematical Society*, 325(1):205–229, 1991.

[10] Martino Bardi and Pierpaolo Soravia. Hamilton-jacobi equations with singular boundary conditions on a free boundary and applications to differential games. *Transactions of the American Mathematical Society*, 325(1):205–229, 1991.

[11] Richard Bellman. Dynamic programming and stochastic control processes. *Information and control*, 1(3):228–239, 1958.

[12] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proc. Annual Int. Conf. Machine Learning*, 2009.

[13] Steven L Brunton, Bingni W Brunton, Joshua L Proctor, and J Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PloS one*, 11(2):e0150171, 2016.

[14] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[15] M. Chen, S. Herbert, and C. J. Tomlin. Fast reachable set approximations via state decoupling disturbances. In *Proc. IEEE Conf. Decision and Control*, 2016.

[16] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin. Decomposition of reachable sets and tubes for a class of nonlinear systems. *IEEE Transactions on Automatic Control*, 63(11):3675–3688, Nov 2018.

[17] Mo Chen, Jaime F Fisac, Shankar Sastry, and Claire J Tomlin. Safe sequential path planning of multi-vehicle systems via double-obstacle hamilton-jacobi-isaacs variational inequality. In *2015 European Control Conference (ECC)*, pages 3304–3309. IEEE, 2015.

[18] Mo Chen, Jaime F Fisac, Shankar Sastry, and Claire J Tomlin. Safe sequential path planning of multi-vehicle systems via double-obstacle hamilton-jacobi-isaacs variational inequality. In *2015 European Control Conference (ECC)*, pages 3304–3309. IEEE, 2015.

[19] Mo Chen, Sylvia Herbert, and Claire J Tomlin. Exact and efficient hamilton-jacobi-based guaranteed safety analysis via system decomposition. *arXiv preprint arXiv:1609.05248*, 2016.

[20] Mo Chen, Sylvia L. Herbert, Haimin Hu, Ye Pu, Jaime F. Fisac, Somil Bansal, SooJean Han, and Claire J. Tomlin. Fastrack: a modular framework for real-time motion planning and guaranteed safe tracking, 2021.

[21] Mo Chen, Qie Hu, Jaime F Fisac, Kene Akametalu, Casey Mackin, and Claire J Tomlin. Reachability-based safety and goal satisfaction of unmanned aerial platoons on air highways. *Journal of Guidance, Control, and Dynamics*, 40(6):1360–1373, 2017.

[22] Mo Chen, Jennifer C Shih, and Claire J Tomlin. Multi-vehicle collision avoidance via hamilton-jacobi reachability and mixed integer programming. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 1695–1700. IEEE, 2016.

[23] Mo Chen, Jennifer C Shih, and Claire J Tomlin. Multi-vehicle collision avoidance via hamilton-jacobi reachability and mixed integer programming. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 1695–1700. IEEE, 2016.

[24] Mo Chen and Claire J Tomlin. Hamilton–jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:333–358, 2018.

[25] Mo Chen and Claire J Tomlin. Hamilton–jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:333–358, 2018.

[26] Mo Chen, Zhengyuan Zhou, and Claire J Tomlin. Multiplayer reach-avoid games via pairwise outcomes. *IEEE Transactions on Automatic Control*, 62(3):1451–1457, 2016.

[27] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[28] Ching-An Cheng, Xinyan Yan, and Byron Boots. Trajectory-wise control variates for variance reduction in policy gradient methods. In *Conference on Robot Learning*, pages 1379–1394. PMLR, 2020.

[29] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[30] Gabriele Ciaramella and Giulia Fabrini. Multilevel techniques for the solution of hjb minimum-time control problems. *Journal of Systems Science and Complexity*, 34(6):2069–2091, 2021.

[31] Charles Dawson, Zengyi Qin, Sicun Gao, and Chuchu Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pages 1724–1735, Auckland, New Zealand, December 2022. PMLR.

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

[33] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[34] Lawrence C Evans and Panagiotis E Souganidis. Differential games and representation formulas for solutions of hamilton-jacobi-isaacs equations. *Indiana University mathematics journal*, 33(5):773–797, 1984.

[35] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[36] Carlos Florensa, David Held, Markus Wulfmeier, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. *CoRR*, 2017.

[37] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[38] Bruce A Francis, Manfredi Maggiore, Bruce A Francis, and Manfredi Maggiore. Models of mobile robots in the plane. *Flocking and rendezvous in distributed robotics*, pages 7–23, 2016.

[39] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. In *International Conference on Learning Representations*, 2018.

[40] Wei Fu, Chao Yu, Zelai Xu, Jiaqi Yang, and Yi Wu. Revisiting some common practices in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2206.07505*, 2022.

[41] Mihalis Giannakis and Michalis Louis. A multi-agent based system with big data processing for enhanced supply chain agility. *Journal of Enterprise Information Management*, 29(5):706–727, 2016.

[42] Jeremy H Gillula, Haomiao Huang, Michael P Vitus, and Claire J Tomlin. Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice. In *2010 IEEE International Conference on Robotics and Automation*, pages 1649–1654. IEEE, 2010.

[43] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004.

[44] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016.

[45] Arthur Guez, David Silver, and Peter Dayan. Efficient bayes-adaptive reinforcement learning using sample-based search. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1025–1033. Curran Associates, Inc., 2012.

[46] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*, pages 66–83. Springer, 2017.

[47] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*, 2018.

[48] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, Long Beach, California, US, June 2019. PMLR.

[49] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[50] Dongge Han, Wendelin Boehmer, Michael Wooldridge, and Alex Rogers. Multi-agent hierarchical reinforcement learning with dynamic termination. In *Pacific Rim International Conference on Artificial Intelligence*, pages 80–92. Springer, 2019.

[51] Yiqiang Han, Wenjian Hao, and Umesh Vaidya. Deep learning of koopman representation for control. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1890–1895, Jeju Island, Republic of Korea, December 2020. IEEE.

[52] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[53] Thomas Hofmann, Aurelien Lucchi, Simon Lacoste-Julien, and Brian McWilliams. Variance reduced stochastic gradient descent with neighbors. *Advances in Neural Information Processing Systems*, 28, 2015.

[54] H. Hu, Y. Pu, M. Chen, and C. J. Tomlin. Plug and play distributed model predictive control for heavy duty vehicle platooning and interaction with passenger vehicles. In *2018 IEEE Conference on Decision and Control (CDC)*, 2018.

[55] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[56] Boris Ivanovic, James Harrison, Apoorva Sharma, Mo Chen, and Marco Pavone. Barc: Backward reachability curriculum for robotic reinforcement learning. In *Proc. IEEE Int. Conf. Robotics and Automation*, 2019.

[57] Brian Edward Jackson, Jeong Hun Lee, Kevin Tracy, and Zachary Manchester. Data-efficient model learning for control with jacobian-regularized dynamic-mode decomposition. In *Conference on Robot Learning*, pages 2273–2283, Atlanta, GA, November 2023. PMLR.

[58] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[59] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Data-driven discovery of koopman eigenfunctions for control. *Machine Learning: Science and Technology*, 2(3):035023, 2021.

[60] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254, 2019.

[61] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2004.

[62] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[63] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

[64] Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

[65] Milan Korda and Igor Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149–160, 2018.

[66] Milan Korda and Igor Mezić. Optimal construction of koopman eigenfunctions for prediction and control. *IEEE Transactions on Automatic Control*, 65(12):5114–5129, 2020.

[67] Klemen Kotar, Gabriel Ilharco, Ludwig Schmidt, Kiana Ehsani, and Roozbeh Mottaghi. Contrasting contrastive self-supervised representation learning pipelines. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9949–9959, virtually, October 2021. IEEE.

[68] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29:3675–3683, 2016.

[69] Philippe Laferrière, Samuel Laferrière, Steven Dahdah, James Richard Forbes, and Liam Paull. Deep koopman representation for control over images (dkrci). In *2021 18th Conference on Robots and Vision (CRV)*, pages 158–164, Burnaby, British Columbia, May 2021. IEEE.

[70] Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

[71] Béla Lantos and Lőrinc Márton. *Nonlinear control of vehicles and robots*. Springer Science & Business Media, 2010.

[72] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650, The Baltimore Convention Center, July 2020. PMLR.

[73] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[74] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, May 2015.

[75] Ernest Bruce Lee and Lawrence Markus. *Foundations of optimal control theory*, volume 87. Wiley New York, 1967.

[76] Yingying Li, Xin Chen, and Na Li. Online optimal control with linear dynamics and predictions: Algorithms and regret analysis. *Advances in Neural Information Processing Systems*, 32, 2019.

[77] Lilian Weng. Contrastive Representation Learning. Blog article, 2021-05-31. Accessed on 28 May 2023.

[78] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[79] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016.

[80] Feng Lin and Robert D Brandt. An optimal control approach to robust control of robot manipulators. *IEEE Transactions on Robotics and Automation*, 14(1):69–77, 1998.

[81] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1774–1783, 2018.

[82] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[83] Michael Lederman Littman. *Algorithms for sequential decision-making*. Brown University, 1996.

[84] Hao Liu, Yihao Feng, Yi Mao, Dengyong Zhou, Jian Peng, and Qiang Liu. Action-depedent control variates for policy optimization via stein's identity. *arXiv preprint arXiv:1710.11198*, 2017.

[85] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, November 2018.

[86] Xubo Lyu, Amin Banitalebi-Dehkordi, Mo Chen, and Yong Zhang. Asynchronous, option-based multi-agent policy gradient: A conditional reasoning approach. *arXiv preprint arXiv:2203.15925*, 2022.

[87] Xubo Lyu and Mo Chen. Ttr-based reward for reinforcement learning with implicit model priors. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5484–5489. IEEE, 2020.

[88] Xubo Lyu, Hanyang Hu, Seth Siriya, Ye Pu, and Mo Chen. Task-oriented koopman-based control with contrastive encoder. In *7th Annual Conference on Robot Learning*, 2023.

[89] Xubo Lyu, Site Li, Seth Siriya, Ye Pu, and Mo Chen. Mbb: Model-based baseline for global guidance of model-free reinforcement learning via lower-dimensional solutions, 2021.

[90] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on Autonomous agents*, pages 246–253, 2001.

[91] Giorgos Mamakoukas, Maria Castano, Xiaobo Tan, and Todd Murphey. Local koopman operators for data-driven control of robotic systems. In *Robotics: science and systems XV*, Freiburg im Breisgau, Germany, June 2019.

[92] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011.

[93] Ian M. Mitchell. The flexible, extensible and efficient toolbox of level set methods. *J. Scientific Computing*, 35(2):300–329, Jun 2008.

[94] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, February 2015.

[95] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[96] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A. Preiss, Nora Ayanian, and Gaurav S. Sukhatme. Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 59–66, 2019.

[97] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018.

[98] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[99] Ian Osband, Benjamin Van Roy, Daniel Russo, and Zheng Wen. Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*, 2017.

[100] Jyoti Ranjan Pati. *Modeling, identification and control of cart-pole system.* PhD thesis, ETH, 2014.

[101] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[102] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[103] Athanasios S. Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *J. Intelligent & Robotic Systems*, 86(2):153–173, May 2017.

[104] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[105] Y. Pu, M. N. Zeilinger, and C. N. Jones. Inexact fast alternating minimization algorithm for distributed model predictive control. In *53rd IEEE Conference on Decision and Control*, 2014.

[106] Ye Pu, Melanie N Zeilinger, and Colin N Jones. Complexity certification of the fast alternating minimization algorithm for linear mpc. *IEEE Transactions on Automatic Control*, 62(2):888–893, 2016.

[107] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, may 2009.

[108] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[109] Filippo Rinaldi, Sergio Chiesa, and Fulvia Quagliotti. Linear quadratic control for quadrotors uavs dynamics and formation flight. *Journal of Intelligent & Robotic Systems*, 70:203–220, 2013.

[110] Angel Romero, Sihao Sun, Philipp Foehn, and Davide Scaramuzza. Model predictive contouring control for time-optimal quadrotor flight. *IEEE Transactions on Robotics*, 38(6):3340–3356, 2022.

[111] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[112] Atefeh Sahraeekhanghah and Mo Chen. Pa-fastrack: Planner-aware real-time guaranteed safe planning. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2129–2136, 2021.

[113] Sosale Shankara Sastry and Alberto Isidori. Adaptive control of linearizable systems. *IEEE Transactions on Automatic Control*, 34(11):1123–1131, 1989.

[114] Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

[115] Peter J Schmid. Application of the dynamic mode decomposition to experimental data. *Experiments in fluids*, 50:1123–1130, 2011.

[116] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.

[117] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning (ICML)*, 2015.

[118] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proc. Annual Int. Conf. Machine Learning*, 2015.

[119] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[120] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[121] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[122] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[123] Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

[124] Haojie Shi and Max Q-H Meng. Deep koopman operator with control for nonlinear systems. *IEEE Robotics and Automation Letters*, 7(3):7700–7707, 2022.

[125] Sumeet Singh, Anirudha Majumdar, Jean-Jacques Slotine, and Marco Pavone. Robust online motion planning via contraction theory and convex optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5883–5890. IEEE, 2017.

[126] Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991.

[127] Aivar Sootla, Alexandre Mauroy, and Damien Ernst. Optimal control formulation of pulse-based control using koopman operator. *Automatica*, 91:217–224, 2018.

[128] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[129] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[130] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[131] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[132] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, USA, 2018.

[133] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 2000.

[134] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[135] Zaid Tahir, Waleed Tahir, and Saad Ali Liaqat. State space system modelling of a quad copter uav. *arXiv preprint arXiv:1908.07401*, 2019.

[136] Ryo Takei and Richard Tsai. Optimal trajectories of curvature constrained motion in the hamilton–jacobi formulation. *J. Scientific Computing*, 54(2):622–644, Feb 2013.

[137] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[138] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pages 2753–2762, 2017.

[139] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[140] Russ Tedrake. *Underactuated Robotics*. 2023.

[141] Emanuel Todorov et al. Optimal control theory. *Bayesian brain: probabilistic approaches to neural coding*, pages 268–298, 2006.

[142] Harry L Trentelman, Anton A Stoorvogel, and Malo Hautus. *Control theory for linear systems*. Springer Science & Business Media, 2012.

[143] Jonathan H Tu. *Dynamic mode decomposition: Theory and applications*. PhD thesis, Princeton University, 2013.

[144] Bas van der Heijden, Laura Ferranti, Jens Kober, and Robert Babuška. Deepkoco: Efficient latent planning with a task-relevant koopman representation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 183–189, Prague, Czech Republic, September 2021. IEEE.

[145] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[146] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic attentive writer for learning macro-actions. In *Advances in neural information processing systems*, pages 3486–3494, 2016.

[147] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org, 2017.

[148] Florian Voigt, Lars Johannsmeier, and Sami Haddadin. Multi-level structure vs. end-to-end-learning in high-performance tactile robotic manipulation. In *Conference on Robot Learning*, pages 2306–2316. PMLR, 2021.

[149] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[150] Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015.

[151] Rose E Wang, J Chase Kew, Dennis Lee, Tsang-Wei Edward Lee, Tingnan Zhang, Brian Ichter, Jie Tan, and Aleksandra Faust. Model-based reinforcement learning for decentralized multiagent rendezvous. *arXiv preprint 2003.06906*, 2020.

[152] S. Wang and K. Hauser. Realization of a real-time optimal control strategy to stabilize a falling humanoid robot with hand contact. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3092–3098, May 2018.

[153] S. Wang and K. Hauser. Realization of a real-time optimal control strategy to stabilize a falling humanoid robot with hand contact. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3092–3098, May 2018.

[154] Woodrow Zhouyuan Wang, Andy Shih, Annie Xie, and Dorsa Sadigh. Influencing towards stable multi-agent interactions. In *Conference on robot learning*, pages 1132–1143. PMLR, 2022.

[155] D. J. Webb and J. van den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *Proc. IEEE Int.Conf. Robotics and Automation*, 2013.

[156] Julian Wiederer, Arij Bouazizi, Marco Troina, Ulrich Kressel, and Vasileios Belagiannis. Anomaly detection in multi-agent trajectories for automated driving. In *Conference on Robot Learning*, pages 1223–1233. PMLR, 2022.

[157] Matthew O Williams, Maziar S Hemati, Scott TM Dawson, Ioannis G Kevrekidis, and Clarence W Rowley. Extending data-driven koopman analysis to actuated systems. *IFAC-PapersOnLine*, 49(18):704–709, 2016.

[158] Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25:1307–1346, 2015.

[159] Matthew O Williams, Clarence W Rowley, and Ioannis G Kevrekidis. A kernel-based approach to data-driven koopman spectral analysis. *arXiv preprint arXiv:1411.2260*, 2014.

[160] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[161] Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M Bayen, Sham Kakade, Igor Mordatch, and Pieter Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines. In *International Conference on Learning Representations*, 2018.

[162] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, apr 2005.

[163] Yongqian Xiao, Xin Xu, and QianLi Lin. Cknet: A convolutional neural network based on koopman operator for modeling latent dynamics from pixels. *arXiv preprint arXiv:2102.10205*, 2021.

[164] Yuchen Xiao, Joshua Hoffman, and Christopher Amato. Macro-action-based deep multi-agent reinforcement learning. In *Conference on Robot Learning*, pages 1146–1161. PMLR, 2020.

[165] Yuchen Xiao, Weihao Tan, and Christopher Amato. Asynchronous multi-agent actor-critic with macro-actions, 2022.

[166] Chengguang Xu, Christopher Amato, and Lawson LS Wong. Hierarchical robot navigation in novel environments using rough 2-d maps. *arXiv preprint arXiv:2106.03665*, 2021.

[167] Ping Xuan and Victor Lesser. Multi-agent policies: from centralized ones to decentralized ones. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, pages 1098–1105, 2002.

[168] Insoon Yang, Sabine Becker-Weimann, Mina J. Bissell, and Claire J. Tomlin. One-shot computation of reachable sets for differential games. In *Proc. ACM Int. Conf. Hybrid Systems: Computation and Control*, 2013.

[169] Jiachen Yang, Igor Borovikov, and Hongyuan Zha. Hierarchical cooperative multi-agent reinforcement learning with skill discovery. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1566–1574, 2020.

[170] Hang Yin, Michael C Welle, and Danica Kragic. Policy learning with embedded koopman optimal control. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic, September 2021.

[171] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

[172] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[173] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE international conference on robotics and automation (ICRA)*, 2016.

[174] Xuyang Zhao, Tianqi Du, Yisen Wang, Jun Yao, and Weiran Huang. Arcl: Enhancing contrastive learning with augmentation-robust representations. *arXiv preprint arXiv:2303.01092*, 2023.

[175] Z. Zhou, R. Takei, H. Huang, and C. J. Tomlin. A general, open-loop formulation for reach-avoid games. In *Proc. IEEE Conf, Decision and Control*, 2012.