When Models Meet Data: Pragmatic Robot Learning with Model-based Optimization

Mohak Bhardwaj

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

Doctor of Philosophy

University of Washington

2024

Reading Committee:

Byron Boots, Chair

Dieter Fox

Siddhartha Srinivasa

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

When Models Meet Data: Pragmatic Robot Learning with Model-based Optimization

Mohak Bhardwaj

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

Autonomous robots operating in complex and dynamic real-world scenarios must exhibit fast and reactive behaviors to adapt to environment changes, and learn to improve their performance over time. While recent advances in reinforcement learning (RL) have shown remarkable progress in learning complex decision-making directly from experience, the sample inefficiency of existing methods limits their application to robotics domains where data collection can be dangerous, time-consuming and expensive. In contrast, traditional model-based optimization approaches for robot control can leverage prior knowledge of physics and task structure to efficiently plan a robot's motion, and have a rich history of successful application in safety-critical systems. However, the complexity of real-world environments can be hard to model precisely, and assumptions made by practical approaches can significantly limit performance. In this thesis, we propose principled methods to combine data-driven approaches with model-based optimization that enable efficient, adaptive and self-improving robots. We address key challenges in designing such methods with varying degrees of prior model knowledge.

First, we show that even with access to perfect models, the real-time performance of existing motion planning algorithms can be sensitive to environment changes. We propose adaptive planning frameworks that leverage past experience to directly optimize the performance of motion planners across the distribution of environments the robot encounters. Second, for tasks where our prior models are approximate, long horizon planning can be

error prone. We show how model-predictive control (MPC), offers an efficient solution for generating adaptive behaviors via finite horizon optimization. We present a GPU accelerated MPC framework for real-world reactive manipulation that can rapidly optimize complex task objectives while ensuring qualitative behavior requirements. Further, we present a general framework that improves over MPC using model-free RL to overcome the effects of model-bias over time. Finally, we consider situations without access to prior models, and explore policy learning from static datasets of interactions. We show how learning predictive models from such datasets in an adversarial manner can enable learning policies that can improve over arbitrary reference policies regardless of data coverage, with formal guarantees.

This thesis contributes general algorithmic frameworks that are broadly applicable across robotics domains, as well as efficient open-source implementations of practical systems that can be leveraged by the wider community. The methods we present are supported by strong theoretical guarantees and empirical performance across a wide variety of benchmark tasks, and real-world manipulator control in dynamic environments.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

This dissertation would not have been possible without the unwavering support I have received from my family, friends, mentors and collaborators throughout my PhD journey.

I would like to start by thanking my advisor, Byron Boots. Byron has been an amazing mentor and his invaluable support, guidance and honest feedback have been instrumental in my growth as a researcher. I am constantly inspired by his breadth of knowledge and keen eye for practical research problems. I am thankful to him for providing me the freedom to pursue different research ideas, for always being patient, and being a voice of reason when I was overwhelmed by the uncertainties of research.

I would also like to thank Dieter Fox, Siddhartha Srinivasa, Sanjiban Choudhury and Aaron Marburg for serving on my thesis committee and providing their valuable time and feedback on my work. During my time as a PhD student I have had the opportunity to work with some awesome mentors. A special thanks to Sanjiban, for being an exceptional research mentor, friend and guide. Working with Sanjiban has not only taught me a lot about robotics, but also the virtue of thinking deeply and critically about scientific ideas. I am grateful for having stumbled into your office 7 years ago as a master's student at CMU, which marked the start of my research journey. I would like to extend my gratitude to Bala for his guidance through different research projects as well as many technical presentations. Thank you for teaching me the virtues of patience and carefully scoping research problems. Also, I wish to thank Ching-An for teaching me about RL theory, for the insightful discussions and invaluable advice.

My internships at NVIDIA and Google DeepMind have been foundational in my development as a researcher, providing me unique perspectives and opportunities for collaborations. I would like to express my sincere thanks to all my collaborators, especially, Ankur Handa, Fabio Ramos, Balakumar Sundaralingam and Dieter Fox at NVIDIA and Jonas Buchli, Mar-

tin Riedmiller, Thomas Lampe, Markus Wulfmeier and Arunkumar Byravan at DeepMind.

During my PhD, I have had the unique opportunity to live in three different cities, and am deeply grateful for the friendships I have formed over the years. I want to thank my friends at the Robot Learning Lab (in Atlanta and Seattle) - Jake, Sandesh, Carolina, Sid, Sasha and Anqi for their support and camaraderie. Thank you for helping me through my journey. I am grateful for Cristina and Michela for making my stay in London really memorable, for the great adventures, and the hours spent at pubs. I also want to thank friends I made outside academia in Seattle, especially Steve for painting the town red with me, and Jo for being the best Spanish learning partner and my biggest cheerleader.

I am extremely lucky to have a few precious long term friends who have been a constant source of emotional support and inspiration to me. Especially, I want to mention Shivam and Vandan for being the best of friends for over a decade. Brothers, thank you for seeing me through thick and thin. I also want to thank Abhinav for were it not for him and Shivam in Seattle, navigating the challenges of the Covid-19 pandemic with the pressures of research would have been insurmountable. I am grateful to Vaibhav and Jatin for being invaluable friends, and for patiently listening when I needed someone to talk to. During my time at CMU, I also found some great friends for life in Rushat, Shivam and Pranav. Thank you for being my family away from home, for the long nights spent working together at RI, and for being there when it mattered most.

Finally, my deepest gratitude goes to my loving family. Their support and sacrifices have been the foundation of my achievements. I am eternally thankful to my mother, Vandana Bhardwaj, for being the source of my emotional strength, my father Vijay Kumar Bhardwaj, for always being a pillar of support in my life and my sister, Ingita Bhardwaj, for always looking out to me. You are not only the best big sister, but also my best friend.

DEDICATION

to my family

Chapter 1

INTRODUCTION

As autonomous robots transition from controlled laboratory settings into the real world, they must demonstrate fast and adaptive behaviors to handle the complexities of dynamically changing environments. For instance, consider an autonomous car navigating through a crowded intersection or an industrial robot tasked with stacking boxes from one pallet to another. In both cases, the robot must produce smooth and safe motions under uncertainty, reason over long horizons to ensure task success, and satisfy several task and robot-specific constraints. For reliable operation across such challenging scenarios, there is a need for flexible decision-making algorithms that enable robots to rapidly optimize complex behaviors and importantly, learn from experience to improve their performance in data-efficient manner.

In this thesis, we propose pragmatic techniques for learning-based robot control that efficiently utilize experience by exploiting various sources of prior knowledge that are available in the robotics domain. This includes prior knowledge of physics (such as black-box simulators or analytical models of dynamics), task structure and even static datasets collected by human teleoperation or safe reference controllers.

Traditional behavior generation approaches based on model-based planning and optimal control offer a promising way forward. Model-based approaches have a rich history in control of complex, safety critical systems [141, 163, 393, 505] owing to their ability to exploit knowledge of physics and task structure to optimize complex objectives and nonlinear constraints. These algorithms work by constructing predictive models of the robot and the environment, and using principled techniques to optimize the robot's long-term behaviors. However, real-time requirements necessitate the use of approximate models that can significantly limit performance. Furthermore, these approaches generally don't have a mechanism to incorporate experience, so the robot does not overcome errors to improve performance over time, or generalize to novel situations.

Model-free RL offers a general-purpose framework for learning control laws for complex dynamical systems directly from experiential data, without the need for domain expertise [455]. Indeed, modern *deep* RL approaches that leverage expressive neural networks for function approximation have led to breakthroughs in a variety of domains, such as game-playing [328, 413], protein folding [225] , and simulated robot control [189]. Since these approaches make minimal assumptions about problem structure, it is attractive to explore their applicability to real-robot control as well, and there have been some successful demonstrations in controlled settings on complex tasks like contact-rich dexterous manipulation [19]. However, RL methods achieve their flexibility by discovering optimal behaviors through trial and error learning. This typically requires a large amount of data to succeed, and fundamentally limits their broad adoption to the control of safety critical robotic systems where data collection can be dangerous (both to the robot and the environment), time-consuming and expensive.

A successful approach for adaptive robot decision-making must incorporate the best elements of both model-based and model-free paradigms - exploiting prior knowledge and structure to achieve good performance initially and continuously improving with experience. This motivates the primary research statement of this thesis

A principled combination of fast, model-based optimization and learning from experience is key for enabling efficient, adaptive and self-improving robots.

Guided by key research questions that arise from this overarching principle, this thesis proposes general algorithmic frameworks and practical systems aimed at enabling robots to perform a wide range of tasks in dynamically changing environments. Next, we provide an overview of the work presented in this thesis that can be broadly categorized based on the nature of prior knowledge that we have access to.

Data-driven Motion Planning

First, we study the problem of motion planning with access to **perfect models** of the robot and the environment, where the task is to find a collision free motion for a robot from a start to goal configuration. There is a rich history of literature on efficient algorithms [163, 543]

that provide strong theoretical guarantees, however, their real-time performance can be sensitive to the actual distribution of environments the robot encounters. To address this issue, we develop frameworks that leverage past experience to directly improve the expected performance of planners over a distribution of problems.

Chapter 2: Leveraging Experience in Lazy Search. Search based algorithms formulate motion planning as finding the shortest feasible free path on a graph. In robot motion planning, collision checking edges is the computational bottleneck affecting real-time performance. Lazy search algorithms are efficient at solving such problems, and work by lazily computing the shortest potentially feasible path, evaluating edges along that path, and repeating until a feasible path is found. In order to minimize total edge evaluations, a good selector must choose edges that are not only likely to be invalid, but also eliminate future paths from consideration. We present an approach to learn such a selector by leveraging prior experience. We cast optimal edge selection to a Markov Decision Process (MDP) on search state and compute oracular selectors that can efficiently solve the MDP during training. Such oracles enable us to use imitation learning to train effective policies, that can then solve similar problems quickly. We evaluate our algorithm on a wide range of problems and show that the learned selector outperforms baseline commonly used heuristics. We further provide a novel theoretical analysis of lazy search in a Bayesian framework and regret guarantees on our imitation learning approach to motion planning.

Chapter 3: Differentiable Gaussian Process Motion Planning. In this chapter, we study the class of gradient-based trajectory optimization approaches for motion planning. While fast and effective on a wide range of robotics tasks, these algorithms have parameters that are typically set in advance (and rarely discussed in detail). Setting these parameters properly can have a significant impact on the practical performance, sometimes making the difference between finding a feasible plan or failing at the task entirely. We propose a method for leveraging past experience to learn how to automatically adapt the parameters of Gaussian Process Motion Planning (GPMP) algorithms. Specifically, we propose a differentiable extension to the GPMP2 algorithm [131], so that it can be trained end-to-end from data. Through several experiments we validate our algorithm and illustrate the benefits of our proposed learning-based approach to motion planning.

Model-Predictive Control and Reinforcement Learning

Next, we study the common scenario in real-world robot control where we only have access to prior **approximate models**. Here, long horizon planning algorithms from previous chapters can suffer from compounding errors due to model-bias. Instead, we focus on the framework of model-predictive control (MPC), that generates feedback controllers by repeatedly optimizing approximate models over finite horizons. We show how we can design scalable MPC frameworks that enable real-world reactive manipulation and how MPC can be improved via model-free RL.

Chapter 4: Fast Joint-Space Model-Predictive Control for Reactive Manipulation. Sampling-based MPC is a promising tool for feedback control of robots with complex, non-smooth dynamics, and cost functions [506]. However, the computationally demanding nature of these algorithms has been a key bottleneck in their application to high-dimensional robotic manipulation problems in the real world. To contend with this, we have developed a system for fast, joint space sampling-based MPC for manipulators that is efficiently parallelized using GPUs. Our approach handles task and joint space constraints while computing control commands at upwards of 125Hz. Further, the method tightly integrates perception into the control problem by utilizing learned cost functions from raw sensor data, and is deployed on a Franka Panda robot for a variety of dynamic manipulation tasks. We study the effects of different cost formulations and MPC parameters on the synthesized behavior and provide key insights for the principled application of sampling-based MPC for manipulators. We also provide highly optimized, open-source code to be used by the wider robot learning and control community.

Chapter 5: Blending MPC & Value Function Approximation for Efficient Reinforcement Learning. Biased dynamics models and short planning horizons can significantly limit the performance of MPC in practice. In this chapter, we present a general framework for improving on MPC with model-free RL. We leverage a key insight that MPC can be viewed as constructing a series of local Q-function approximations, and show how we can systematically trade-off learned value estimates against the local Q-function approximations. We present a theoretical analysis that shows how error from inaccurate models

in MPC and value function estimation in RL can be balanced, and propose an algorithm that smoothly reduces the dependence on MPC as our value function estimate improves. We validate our approach on challenging high dimensional manipulation tasks with biased models in simulation, and show that it obtains performance comparable with MPC with access to true dynamics while being more sample efficient as compared to model-free RL.

Model-based Offline Reinforcement Learning

Finally, we consider the case where we have **no prior model knowledge**, and study learning policies from **static datasets** (collected by safe reference policies). We show how learning predictive models from such datasets can robustly optimize policies without degradation over the reference.

Chapter 6 Adversarial Model-based Offline Reinforcement Learning This chapter presents a novel model-based offline RL framework, called Adversarial Model for Offline Reinforcement Learning (ARMOR), which can robustly learn policies to improve upon an arbitrary reference policy regardless of data coverage. ARMOR is designed to optimize policies for the worst-case performance relative to the reference policy through adversarially training an MDP model. We theoretically prove that ARMOR, with a well-tuned hyperparameter, can compete with the best policy covered by data, when the reference policy is within data support. Simultaneously, ARMOR is also robust to hyperparameter choices: the learned policy, with *any* admissible hyperparameter, will never degrade the performance of the reference policy, even when the reference policy is not covered by the dataset. We empirically validate these properties by designing a scalable implementation of ARMOR and testing it on several benchmark tasks, showing it achieves competent performance with both state-of-the-art offline model-free and model-based RL algorithms, and can robustly improve diverse reference policies over a range of hyperparameter choices.

Chapter 2

LEVERAGING EXPERIENCE IN LAZY SEARCH

2.1 Introduction

This chapter focuses on the paradigm of search-based motion planning that formulates robot motion planning as finding a shortest feasible path on a graph where the vertices represent robot states and edges represent possible motions between them [241]. We develop algorithms that leverage past experience to find the shortest path on a graph while minimizing planning time. In the domain of robot motion planning, planning time is dominated by *edge evaluation* [183]. Therefore, the goal here is to check the minimal number of edges, invalidating potential shortest paths along the way, until we discover the shortest feasible path – this is the central tenet of lazy search [50, 121]. In this chapter, we explore how we can *learn within this framework* which edges to evaluate (Fig. 2.1).

How should we leverage experience? Consider the "Piano Mover's Problem" [422] where the goal is to plan a path for a piano from one room in a house to another. Collision checking all possible motions of the piano can be quite time-consuming. Instead, what can we infer if we were given a database of houses and edge evaluations results?

1. *Check doors first* - these edges serve as bottlenecks for many paths which can be eliminated early if invalid.

2. *Prioritize narrow doors* - these edges are more likely to be invalid and can save checking other edges.

3. *Similar doors, similar outcomes* - these edges are correlated, checking one reveals information about others.

Intuitively, we must consider all past discoveries about edges to make a decision. While this has been explored in the Bayesian setting [89, 97], we show that more generally the problem

can be mapped to a Markov Decision Process (MDP). However, the size of the MDP grows exponentially with the size of the graph. Even if we use approximate dynamic programming, we still need to explore an inordinate number of states to learn a reasonable policy.

Interestingly, if we were to reveal the status of all the edges during training, we can conceive of a *clairvoyant oracle* [95] that selects the optimal sequence of edges to invalidate. In fact, we show that the oracular selector is equivalent to *set cover*, for which greedy approximations exist. By imitating clairvoyant oracles [95], we can drastically cut down exploration and focus learning on the relevant portion of the state space [451]. This leads to a key insight: use imitation learning to bootstrap the selector to match oracular performance.

We propose a new algorithm, StROLL, that deploys an interactive imitation learning framework [402] to train the edge selector (Fig. 2.2). At every iteration, it samples a world (validity status for all edges) and executes the learner. At every timestep, it queries the clairvoyant oracle associated with the world to select an edge to evaluate. This can be viewed as a classification problem where the goal is to map features extracted from edges to the edge selected by the oracle. This datapoint is aggregated with past data, which is then used to update the learner.

We also theoretically analyze the problem in the Bayesian setting to characterize lower bounds. We show that the problem can be mapped to an instance of Bayesian Active Learning, and hence is NP-Hard. However, this mapping allows us to harness the theory of adaptive submodularity to derive analytic, near-optimal policies for edge evaluation. To the best of our knowledge, this is the first such bound on lazy shortest path.

2.1.1 Contributions

To summarize, this chapter makes the following key contributions

1. In Section 2.2, we show how edge selection in lazy search can be mapped to an MDP and in Section 2.3 we show how it can be challenging solve even for small graphs.

2. In Section 2.4, we show that larger MDPs can be efficiently solved by imitating clairvoyant oracles.

3. In Section 2.7, we perform extensive experiments on a wide range of planning datasets show that the learned policy can outperform competitive baselines.

4. In Section 2.6, we introduce the paradigm of Bayesian Lazy Shortest Path and present a theoretical analysis that maps it to the Bayesian Decision Region Determination (DRD) problem for which near-optimal policies can be derived.

5. In Section 2.5, we provide theoretical bounds on the performance StrOLL by first deriving the optimal edge selector under the assumption of independent bernoulli edges followed by a regret analysis that exploits a connection between imitation learning of clairvoyant oracles to hindsight optimization. We also provide theoretical bounds on the performance StrOLL via a regret analysis that exploits a connection between imitation learning of clairvoyant oracles to hindsight optimization. We also show that in the special case of independent Bernoulli edges, the optimal edge selector lies within the policy class of StrOLL.

2.2 Problem Formulation

2.2.1 The Shortest Path (SP) Problem

Let $G = (V, E)$ be an explicit graph where V denotes the set of vertices and E the set of edges. Given a start and goal vertex $(v_s, v_g) \in V$, a path ξ is represented as a sequence of vertices $(v_1, v_2, \ldots, v_l)$ such that $v_1 = v_s, v_l = v_g, \forall i,\ \ (v_i, v_{i+1}) \in E$. We define a *world* $\phi : E \to \{0, 1\}$ as a mapping from edges to valid (1) or invalid (0). A path is said to be *feasible* if all edges are valid, i.e. $\forall e \in \xi, \phi(e) = 1$. Let $\ell : E \to \mathbb{R}^+$ be the length of an edge. The length of a path is the sum of edge lengths, i.e. $\ell(\xi) = \sum_{e \in \xi} \ell(e)$. The objective of the SP problem is the find the shortest feasible path:

$$\min_{\xi} \ \ell(\xi) \quad \text{s.t.} \quad \forall e \in \xi, \phi(e) = 1 \tag{2.1}$$

We now define a family of shortest path algorithms. Given a SP problem, the algorithms evaluate a set of edges $E_{\text{eval}} \subset E$ (verify if they are valid) and return a path ξ^* upon halting. Two conditions must be met:

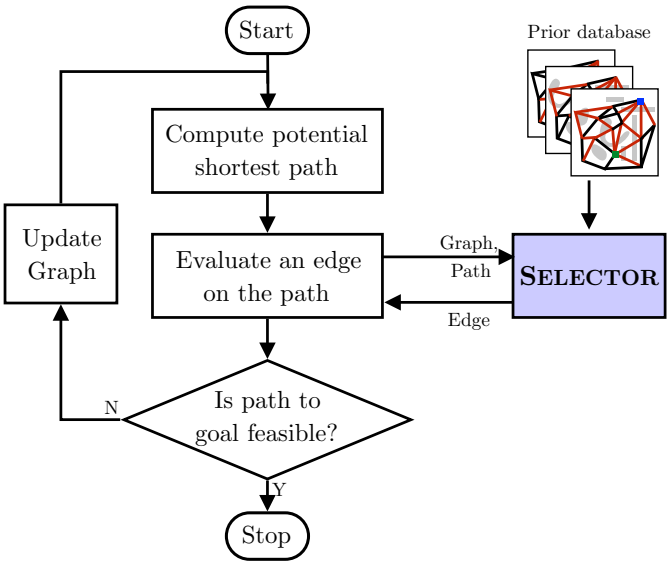
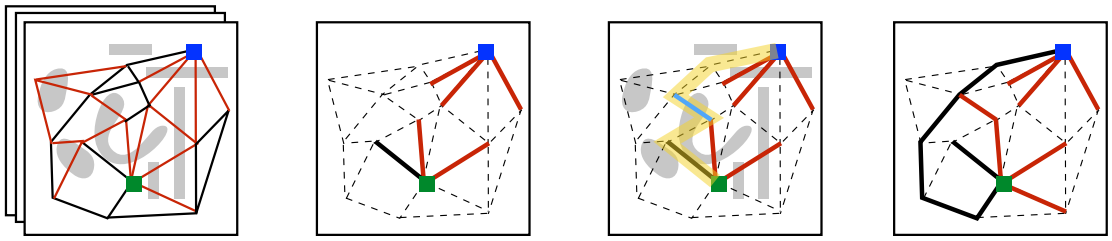


Figure 2.1: The LazySP [121] framework. LazySP iteratively computes the shortest path, queries a Selector for an edge on the path, evaluates it and updates the graph until a feasible path is found. The number of edges evaluated depends on the choice of Selector.



Sample a world from database.

Execute learner (evaluate edges) to a get a state.

Query clairvoyant oracle for edge (action) to evaluate.

Continue aggregating data till the end, update learner.

Figure 2.2: Overview of StrOLL- a training procedure for a Selector to select edges to evaluate in the LazySP framework.

1. The returned path ξ^* is verified to be feasible, i.e. $\forall e \in \xi^*,\ e \in E_{\text{eval}},\ \phi(e) = 1$

2. All paths shorter than ξ^* are verified to be infeasible, i.e. $\forall \xi_i,\ \ell(\xi_i) < \ell(\xi^*),\ \exists e \in \xi_i,\ e \in E_{\text{eval}},\ \phi(e) = 0$

2.2.2 *The Lazy Shortest Path (LazySP) Framework*

We are interested in shortest path algorithms that minimize the number of evaluated edges $|E_{\text{eval}}|$.[1] These are *lazy* algorithms, i.e. they seek to defer the evaluation of an edge as much as possible. When this laziness is taken to the limit, one arrives at the *Lazy Shortest Path* (LazySP) class of algorithms. Under a set of assumptions, this framework can be shown to contain the optimally lazy algorithm [178].

Algorithm 1 describes the LazySP framework. The algorithm maintains a set of evaluated edges that are valid E_{val} and invalid E_{inv}. At every iteration, the algorithm lazily finds the shortest path ξ on the potentially valid graph $G = (V, E \setminus E_{\text{inv}})$ *without evaluating any new edges* (Line 4). It then calls a function, Selector, to select an edge e from this path ξ (Line 5). Based on the outcome, this edge is added to either E_{val} or E_{inv}. This continues until conditions in Section 2.2.1 are satisfied, i.e. the shortest feasible path is found.

The algorithm has one free parameter - the Selector function. The only requirement for a valid Selector is to select an edge on the path. As shown in [121], one can design a range of selectors such as:

1. Forward: select the first unevaluated edge $e \in \xi$. Effective if invalid edges are near the start.

2. Backward: select the last unevaluated edge $e \in \xi$. Effective if invalid edges are near the goal.

3. Alternate: alternates between first and last edge. This approach hedges its bets between start and goal.

4. FailFast: selects the least likely edge $e \in \xi$ to be valid based on prior data.

5. PostFailFast: selects the least likely edge $e \in \xi$ to be valid using a Bayesian posterior based on edges checked so far.

[1] The framework can be extended to handle non-uniform evaluation cost as well

Algorithm 1: LazySP

Input : Graph G, start v_s, goal v_g, world ϕ

Parameter: Selector

Output : Path ξ^*, evaluated edges E_{eval}

$E_{\text{val}} \leftarrow \emptyset$ ▷ Valid evaluated edges

$E_{\text{inv}} \leftarrow \emptyset$ ▷ Invaid evaluated edges

repeat

  $\xi \leftarrow \textsc{ShortestPath}(E \setminus E_{\text{inv}})$

  $e \leftarrow \textsc{Selector}(\xi, E_{\text{val}}, E_{\text{inv}})$ ▷ Select edge on ξ

  if $\phi(e) \neq 0$ **then**

    $E_{\text{val}} \leftarrow E_{\text{val}} \cup \{e\}$

  else

    $E_{\text{inv}} \leftarrow E_{\text{inv}} \cup \{e\}$

  end

until *feasible path found s.t.* $\forall e \in \xi, e \in E_{\text{val}}$;

return $\{\xi^* \leftarrow \xi,\ E_{\text{eval}} \leftarrow E_{\text{val}} \cup E_{\text{inv}}\}$;

While these baseline selectors are very effective in practice, their performance, i.e. the number of edges evaluated $|E_{\text{eval}}|$ depends on the underlying world ϕ which dictates which edges are invalid. Hence the goal is to compute a good Selector that is effective given a *distribution of worlds*, $P(\phi)$. We formalize this as follows

Problem 1 (Optimal Selector Problem)**.** *Let the edges evaluated by* Selector *on world* ϕ *be denoted by* $E_{\text{eval}}(\phi, \textsc{Selector})$*. Given a distribution of worlds,* $P(\phi)$*, find a* Selector *that minimizes the expected number of evaluated edges, i.e.*

$$\min \mathbb{E}_{\phi \sim P(\phi)} \left[|E_{\text{eval}}(\phi, \textsc{Selector})| \right]$$

Problem 1 is a sequential decision making problem, i.e. decisions made by the selector in one iteration (edge selected) affects the input to the selector in the next iteration (shortest path). We show how to formally handle this in the next section. It is interesting to note

that Problem 1 can be solved optimally under certain strong assumptions as detailed in Section 2.5.

2.2.3 Mapping the Optimal Selector Problem to an MDP

Problem 1 can be mapped to a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, \mathrm{T}, \mathrm{R}, \gamma \rangle$ as follows:

State Space The state $s = (E_{\text{val}}, E_{\text{inv}})$ is the set of evaluated valid edges E_{val} and evaluated invalid edges E_{inv}. This can be represented by a vector of size $|E|$, each element being one of $\{-1, 0, 1\}$ - unevaluated, evaluated invalid, and evaluated valid respectively. For simplicity, we assume that the explicit graph $G = (V, E)$ is fixed.[2] Since each $e \in E$ can be in one of 3 sets, the cardinality of the state space is $|S| = 3^{|E|}$. The MDP has an absorbing goal state set $\mathcal{G} \subset \mathcal{S}$ which is a set of states where all the edges on the current shortest path are evaluated to be valid, i.e.

$$\mathcal{G} = \{(E_{\text{val}}, E_{\text{inv}}) \mid \forall e \in \textsc{ShortestPath}(E \setminus E_{\text{inv}}), e \in E_{\text{val}}\} \tag{2.2}$$

Action Set The action set $\mathcal{A}(s)$ is the set of unevaluated edges on the current shortest path, i.e.

$$\mathcal{A}(s) = \{e \in \textsc{ShortestPath}(E \setminus E_{\text{inv}}), e \notin \{E_{\text{val}} \cup E_{\text{inv}}\}\} \tag{2.3}$$

Transition Function Given a world ϕ, the transition function is deterministic $s' = \Gamma(s, a, \phi)$:

$$\Gamma(s, a, \phi) = \begin{cases} (E_{\text{val}} \cup \{e\}, E_{\text{inv}}) & \text{if } \phi(e) = 1 \\ (E_{\text{val}}, E_{\text{inv}} \cup \{e\}) & \text{if } \phi(e) = 0 \end{cases} \tag{2.4}$$

Since ϕ is latent and distributed according to $P(\phi)$, we have a stochastic transition function

$$\mathrm{T}(s, a, s') = \sum_{\phi} P(\phi) \mathbb{I}(s = \Gamma(s, a, \phi))$$

[2] We can handle a varying graph by adding it to the state space.

Reward Function The reward function penalizes every state other than the absorbing goal state $\mathcal{G}$, i.e.

$$\mathrm{R}(s,a) = \begin{cases} 0 & \text{if } s \in \mathcal{G} \\ -1 & \text{otherwise} \end{cases} \tag{2.5}$$

γ is a discount factor that is used to favor immediate rewards over later ones. We can define the value of a given policy π as $V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) \mid s_0 = s\right]$ and the action-value function as $Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) \mid s_0 = s, a_0 = a\right]$ where the expectation is over the policy and transition function. Further, we can also define the advantage function $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$ which measures how good an action is compared to the action taken by the policy in expectation.

2.3 Challenges

In this section, we examine tiny graphs and show that even for such problems, a choice of world distributions where edges are correlated can affect SELECTOR choices. However, by solving the MDP using tabular Q-learning we can automatically recover the optimal SELECTOR.

2.3.1 Experimental setup

We train selectors on two different graphs and corresponding distribution of worlds $P(\phi)$.

Environment 1 Fig. 2.3(a) illustrates the distribution of Environment 1. The graph has 6 edges. With 70% probability, `top_left` edge is invalid. If `top_left` is invalid, then `middle_right` is always invalid. If `top_left` is valid, then with 50% probability, `top_right` is invalid plus any one of remaining four are invalid.

The optimal policy is to check `top_left` edge first.

- *If invalid*, check `middle_right` (which is necessarily invalid) and check bottom two edges which are feasible. This amounts to 4 evaluated edges.

- *If valid*, check other edges in order as they all have 50% probability of being valid.

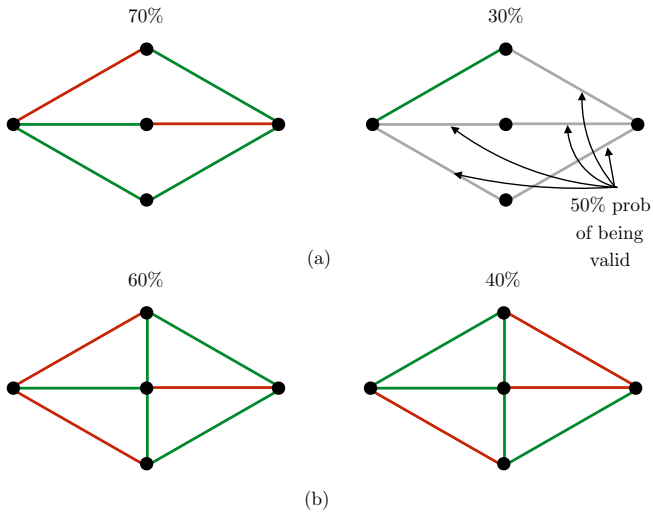


Figure 2.3: Distribution over worlds for (a) Environment 1 and (b) Environment 2. The goal is to find a path from left to right. Edges are valid (green) or invalid (red)

Environment 2 Fig. 2.3(b) illustrates the distribution of Environment 2. The graph has 8 edges. 60% of the time `top_left`, `middle_right` and `bottom_left` are invalid. Else, the `top_right` and the `middle_right` are invalid. Intuitively, 60% of the time, SelectAlternate is optimal and 40% of the time, SelectBackward is the best.

2.3.2 Solving the MDP via Q-learning

We apply tabular Q-learning [502] to compute the optimal value $Q^*(s, a)$. Broadly speaking, the algorithm utilizes an $\epsilon-$greedy policy to visit states, gather rewards, and perform Bellman backups to update the value function. Environment 1 has 729 states, Environment 2 has 6561 states. The learning parameters are shown in Table 2.1.

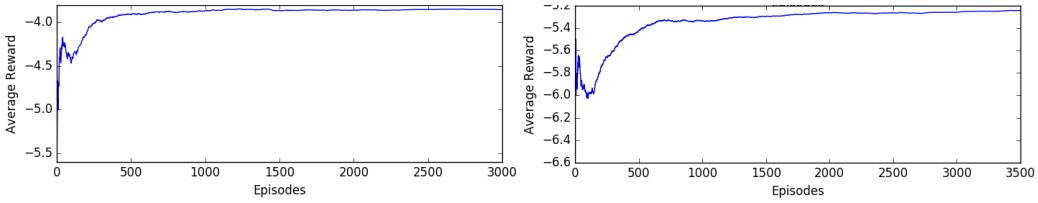
Fig. 2.4 shows the average reward during training for Q-learning. Environment 1 converges after $\approx$ 1000 episodes, environment 2 after $\approx$ 3000 episodes. Table 2.2 shows a comparison of Q-learning with other heuristic baselines in terms of average reward on a validation dataset of 1000 problems. In Environment 1, the learner discovers the optimal policy. Interestingly, Alternate also achieves this result since the correlated edges are alternating. In Environment 2, the learner has a clear margin as compared to heuristic baselines, all of

Table 2.1: Q-learning parameters.

Parameter	**Env. 1**	**Env. 2**
Number of episodes	3000	3500
Exploration episodes	100	150
ϵ_0	1	1
Discount factor	1	1
Learning rate	0.5	0.5

Table 2.2: Avg. reward after 1000 test episodes.

Method	**Env. 1**	**Env. 2**
Tabular Q-learning	-3.85	-5.24
FORWARD	-4.54	-6.00
BACKWARD	-4.42	-5.79
ALTERNATE	-3.86	-6.00
RANDOM	-4.48	-5.90



(a) Environment 1 (3000 train episodes)

(b) Environment 2 (3500 train episodes)

Figure 2.4: Average reward per epsiode of Tabular Q-learning.

which are vulnerable to one of the modes. This shows that, even on such small graphs, it is possible to create an environment where heuristic baselines fail. However, the fact that the learner can recover optimal policies is promising.

2.3.3 Challenges on scaling to larger graphs

While we can solve the MDP for tiny graphs, we run into a number of problems as we try to scale to larger graphs:

Exponentially large state space The size of the state space is $|S| = 3^{|E|}$. This leads to exponentially slower convergence rates as the size of the graph increases. Even if we could manage to visit only the relevant portion of this space, this approach would not generalize

across graphs.

Convergence issues with approximate value iteration We can scale to large graphs if we use a function approximator. In this case, we have to featurize (s, a) as a vector f, i.e. we are trying to approximate $Q(s, a) \approx Q(f)$. Fortunately, we have a set of baseline heuristics(Section 2.2.2) that can be used as a feature vector. This choice allows us to potentially improve upon baselines and easily switch between problem domains.

We run into another problem - approximate value iteration is not guaranteed to converge [170]. This is exaggerated in our case where f is a set of baseline heuristics that may not retain the same information content as the state s. Hence multiple states map to the same feature f, which leads to oscillations and local minima.

Sparse rewards Every state gets a penalization except the absorbing state, i.e. rewards are sparse. Because we are using a function approximator, updates to $Q(f)$ for reaching the goal state are overridden by updates due to -1 penalization.

2.4 Approach

Our approach, StrOLL (**S**earch **th**r**o**ugh **O**racle **L**earning and **L**aziness), is to imitate clairvoyant oracles that can show how to evaluate edges optimally given full knowledge of the MDP at training time. To deal with distribution mismatch between oracle and learner, we use established techniques for iterative supervised learning.

2.4.1 Optimistic Value Estimate using a Clairvoyant Oracle

Consider the situation where the world ϕ is fully known to the selector, i.e. the 0/1 status of all edges are known. The selector can then judiciously select edges that are not only invalid, but eliminate paths quickly. We call such a selector a *clairvoyant oracle*. We show that the optimal clairvoyant oracle, that evaluates the minimal number of edges, is the solution to a set cover problem.

Theorem 1 (Clairvoyant Oracle as Set Cover)**.** *Let $s = (E_{\text{val}}, E_{\text{inv}})$ be a state. Let $V^*(s, \phi)$ be the optimal state action value when the world ϕ is known. Then $V^*(s, \phi)$ is the solution*

to the following set cover problem

$$
\begin{aligned}
-\min_{E_{\text{eval}} \subset \{e \in E \mid \phi(e)=0\}} & |E_{\text{eval}}| \\
\text{s.t.} \quad & \forall \xi,\ \ell(\xi) < \ell(\xi^*), \xi \cap E_{\text{inv}} = \emptyset, \\
& \xi \cap E_{\text{eval}} \neq \emptyset
\end{aligned}
\tag{2.6}
$$

where ξ^* is the shortest feasible path for world ϕ.

Proof. (Sketch) Let $\Xi = \{\xi_1, \ldots, \xi_n\}$ be the set of paths that satisfy the constraints of (2.6)

1. Shorter than ξ^*, i.e. $\ell(\xi_i) < \ell(\xi^*)$

2. Paths are not yet invalidated i.e. $\xi \cap E_{\text{inv}} = \emptyset$

Let $\{e \in E \mid \phi(e) = 0\}$ be the set of invalid edges. Each edge e covers a path $\xi_i \in \Xi$ if $e \in \xi_i$. We define a cover as a set of edges E_{eval} that covers all paths in Ξ, i.e. $\xi_i \cap E_{\text{eval}} \neq \emptyset$.

If we select a min cover, i.e. $\min |E_{\text{eval}}|$ then all shorter paths will be eliminated. Hence this is equal to the optimal value $-V^*(s, \phi)$. □

Theorem 1 says that given a world and a state of the search, the clairvoyant oracle selects the minimum set of invalid edges to eliminate paths shorter than the shortest feasible path.

Let $\pi_{\text{OR}}(s, \phi)$ be the corresponding oracle policy. We note that the optimal clairvoyant oracle can be used to derive an upper bound for the optimal value

$$
Q^*(s, a) \leq Q^{\pi_{\text{OR}}}(s, a) = \sum_{\phi} P(\phi|s) Q^{\pi_{\text{OR}}}(s, a, \phi) \tag{2.7}
$$

where $P(\phi|s)$ is the posterior distribution over worlds given state and $Q^{\pi_{\text{OR}}}(s, a, \phi)$ is the value of executing action a in state s and subsequently rolling-out the oracle. Hence this upper bound can be used for learning.

2.4.2 Approximating the Clairvoyant Oracle

Since set cover is NP-Hard, we have to approximately solve (2.6). Fortunately, a greedy approximation exists which is near-optimal. The greedy algorithm iterates over the following

Algorithm 2: APPROXIMATE CLAIRVOYANT ORACLE

Input : State $s = (E_{\text{val}}, E_{\text{inv}})$, world ϕ

Output: Action a

Compute shortest path $\hat{\xi} = \textsc{ShortestPath}(E \setminus E_{\text{inv}})$

$\Delta \leftarrow 0_{|E|\times 1}$

for $e \in \hat{\xi}, \phi(e) = 0$ **do**

  $\Delta(e) \leftarrow \ell(\textsc{ShortestPath}(E \setminus \{E_{\text{inv}} \cup \{e\}\})) - \ell(\hat{\xi})$

return Action $a = \arg\max_{e \in \hat{\xi}} \Delta(e)$;

rule:

$$e_i = \arg\max_{e \in E, \phi(e)=0} |\{\xi \mid \ell(\xi) < \ell(\xi^*),\ \xi \cap E_{\text{inv}} = \emptyset,\ e \in \xi\}| \tag{2.8}$$
$$E_{\text{eval}} \leftarrow E_{\text{eval}} \cup \{e_i\}$$

The approach greedily selects an invalid edge that covers the maximum number of shorter paths, which have not yet been eliminated. This greedy process is repeated until all paths are eliminated.

There are two practical problems with computing such an oracle. First, exhaustively enumerating all shorter paths $\{\xi \mid \ell(\xi) < \ell(\xi^*)\}$ is expensive, even at train time. Second, if we simply wish to query the oracle for which edge to select on the current shortest path $\hat{\xi} = \textsc{ShortestPath}(E \setminus E_{\text{inv}})$, it has to execute (2.8) potentially multiple times before such an edge is discovered - which also can be expensive. Hence we perform a double approximation.

The first approximation to (2.8) is to constrain the oracle to only select an edge on the current shortest path $\hat{\xi} = \textsc{ShortestPath}(E \setminus E_{\text{inv}})$

$$\approx \arg\max_{e \in \hat{\xi},\ \phi(e)=0} |\{\xi \mid \ell(\xi) \leq \ell(\xi^*),\ \xi \cap E_{\text{inv}} = \emptyset,\ e \in \xi\}| \tag{2.9}$$

The second approximation to (2.9) is to replace the number of paths covered with the marginal gain in path length on invalidating an edge.

$$\approx \arg\max_{e \in \hat{\xi},\ \phi(e)=0} \ell(\textsc{ShortestPath}(E \setminus \{E_{\text{inv}} \cup \{e\}\})) - \ell(\hat{\xi}) \tag{2.10}$$

Alg. 2 summarizes this approximate clairvoyant oracle.

2.4.3 Bootstrapping with Imitation Learning

Imitation learning is a principled way to use the clairvoyant oracle $\pi_{\text{OR}}(s, \phi)$ to assist in training the learner $\pi(s)$. In our case, we can use the oracle action value $Q^{\pi_{\text{OR}}}(s, a)$ as a target for our learner as follows:

$$\arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[Q^{\pi_{\text{OR}}}(s, \pi(s)) \right] \tag{2.11}$$

where $d_\pi(s)$ is the distribution of states. Note that this is now a classification problem since the labels are provided by the oracle. However the distribution d_π depends on the learner's π. [401] shows that this type of imitation learning problem can be reduced to interactive supervised learning. We simplify further. Computing the oracle value requires rolling out the oracle until termination. We empirically found this to significantly slow down training time. Instead, we train the policy to directly predict the action that is selected by the oracle. This is the same as (2.11) but with a 0/1 loss [402] -

$$\arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[\mathbb{I}(\pi(s) = \pi_{\text{OR}}(s, \phi)) \right] \tag{2.12}$$

We justify this simplification by first showing that maximizing action value is same as maximizing the advantage $Q^{\pi_{\text{OR}}}(s, a) - V^{\pi_{\text{OR}}}(s)$. Since all the rewards are -1, the advantage can be lower bounded by the 0/1 loss. We summarize this as follows:

$$\begin{aligned} & \max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[Q^{\pi_{\text{OR}}}(s, \pi(s)) \right] \\ = \; & \max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[Q^{\pi_{\text{OR}}}(s, \pi(s)) - V^{\pi_{\text{OR}}}(s) \right] \\ \geq \; & \max_{\pi \in \Pi} \mathbb{E}_{s \sim d_\pi(s)} \left[\mathbb{I}(\pi(s) = \pi_{\text{OR}}(s, \phi)) - 1 \right] \end{aligned} \tag{2.13}$$

Finally, we do not use the exact clairvoyant oracle but rather an approximation (Section 2.4.2). In other words, there can exist policies $\pi \in \Pi$ that outperform the oracle. In such a case, one can potentially apply policy improvement after imitation learning. However, we leave the exploration of this direction to future work.

Algorithm 3: StrOLL

Input : World distribution $P(\phi)$, oracle π_{OR}

Parameter: Iter N, roll-in policy π_{roll}, mixing $\{\beta_i\}_{i=1}^{N}$

Output : Policy $\hat{\pi}$

Initialize $\mathcal{D} \leftarrow \emptyset$, $\hat{\pi}_1$ to any policy in Π

for $i = 1, \ldots, N$ **do**

  Initialize sub-dataset $\mathcal{D}_i \leftarrow \emptyset$

  Let mixture policy be $\pi_{\text{mix}} = \beta_i \pi_{\text{roll}} + (1 - \beta_i)\hat{\pi}_i$

  for $j = 1, \ldots, m$ **do**

    Sample $\phi \sim P(\phi)$;

    Rollin π_{mix} to get state trajectory $\{s_t\}_{t=1}^{T}$

    Invoke oracle to get $a_t = \pi_{\text{OR}}(s_t, \phi)$

    $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{(s_t, a_t)\}_{t=1}^{T}$;

  Aggregate data $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$;

  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$;

return Best $\hat{\pi}$ on validation;

2.4.4 Algorithm

The problem in (2.12) is a non-*i.i.d* classification problem - the goal is to select the same action the oracle would select on the *on policy distribution of learner*. [402] proposed an algorithm, DAgger, to exactly solve such problems.

Alg. 3, describes the StrOLL framework which iteratively trains a sequence of policies $(\hat{\pi}_1, \hat{\pi}_2, \ldots, \hat{\pi}_N)$. At every iteration i, we collect a dataset $\mathcal{D}_i$ by executing m different episodes. In every episode, we sample a world ϕ which already has every edge evaluated. We then roll-in a policy (execute a selector) which is a mixture π_{mix} that blends the learner's current policy, $\hat{\pi}_i$ and a base roll-in policy π_{roll} using blending parameter β_i. At every time step t, we query the clairvoyant oracle with state s_t to receive an action a_t. We use the approximate oracle in Alg. 2. We then extract a feature vector f from all (s_t, a) tuples and

create a classification datapoint. We add this datapoint to the dataset $\mathcal{D}_i$. At the end of m episodes, this data is then *aggregated with the existing dataset* $\mathcal{D}$. A new classifier $\hat{\pi}_{i+1}$ is trained on the aggregated data. At the end of N iterations, the algorithm returns the best performing policy on a set of held-out validation environments.

We have two algorithms based on the choice of π_{roll}:

1. StROLL: We set $\pi_{\text{roll}} = \pi_{\text{OR}}$. This is the default mode of DAgger. This uses the oracle state distribution to stabilize learning initially.

2. StROLL-R: We set π_{roll} to be the best performing heuristic on training as defined in Section 2.2.2. This uses a heuristic state distribution to stabilize learning. Since the heuristic is realizable, it can have a stabilizing effect on datasets where the oracle is far from realizable.

In the next section we discuss the different sources of error in our proposed framework and provide regret guarantees on the performance of StROLL.

2.5 Theoretical Bounds on Performance

We wish to bound the performance of the policy output by StROLL $\hat{\pi}$ versus the optimal MDP policy π^*.

$$\mathbb{E}_{s\sim d_\pi(s)}\left[V^{\pi^*}(s) - V^{\hat{\pi}}(s)\right] \leq \epsilon \tag{2.14}$$

where $V^{\pi^*}(s)$ is the value of the optimal policy, $V^{\hat{\pi}}(s)$ is the value of the STROLL policy and $d_\pi(s)$ is the distribution of states visited by the learner. To understand the sub-optimality bound ϵ, we need to examine the various components of this error.

2.5.1 Component 1: Unrealizability of the Clairvoyant Oracle

The first source of approximation error comes from our use of a clairvoyant oracle. The clairvoyant oracle has access to the true status of all edges in the graph whereas the learner is only privy to the status of edges checked so far. The realizability gap between the two is vast, resulting in a trivially large regret bound [401]. Instead, [95] shows that imitating

the clairvoyant oracle is in fact equivalent to imitating a corresponding *hallucinating oracle*, that computes an instantaneous posterior over worlds given the edge evaluations so far and computes the expected clairvoyant oracle action value over this posterior i.e,

$$Q^{OR}(s,a) = \mathbb{E}_{\phi \sim P(\phi|s)}\left[Q^{\pi_{\mathrm{OR}}}(s,a,\phi)\right] \tag{2.15}$$

where $Q^{OR}(s,a)$ is the value of action a in state s, $P(\phi|s)$ is the posterior over worlds and $Q^{\pi_{\mathrm{OR}}}(s,a,\phi)$ is the action value computed by the clairvoyant oracle.

The hallucinating oracle policy is defined as one that greedily maximizes the action value $Q^{OR}(s,a)$

$$\tilde{\pi}_{\mathrm{OR}} \cong \arg\max_{a \in \mathcal{A}} Q^{OR}(s,a) \tag{2.16}$$

The hallucinating oracle uses the same information as the learner and is equivalent to the well-known QMDP policy [305]. The QMDP approximation, also known as *hindsight optimization* operates under the assumption that the agent's uncertainty over the true world will be entirely eliminated after the next action. Thus, it simply estimates the expected Q-function weighted by the posterior probability over worlds. This results in an agent that chooses actions that maximize long-term rewards for all worlds weighted by their probability, while ignoring explicit information gathering. Policies based on this assumption have been shown to be effective in several POMDP domains in prior work [219, 256, 525]. However, the QMDP algorithm requires sampling from the *true* posterior over worlds which is intractable in general. Nevertheless, Lemma 1 of [95] state we are effectively imitating this hallucinating oracle by imitating the clairvoyant oracle.

Hence, we assume that the error between the value of the optimal MDP policy and the hallucinating oracle is bounded by

$$\left|\left|V^{\pi^*}(s) - V^{\tilde{\pi}_{\mathrm{OR}}}(s)\right|\right|_{\infty} \leq \epsilon_{\mathrm{hal}} \tag{2.17}$$

We note that ϵ_{hal} can be large for problems requiring a great deal of active information gathering and is hence difficult to quantify in general.

2.5.2 Component 2: Approximations in the Oracle Selector

As discussed in Sec. 2.4.2, we make two simplifying approximations to efficiently compute the oracle at train time. Hence, instead of accurately computing $Q^{\pi_{\text{OR}}}(s, a, \phi)$, an approximation $Q^{\pi_{\text{AOR}}}(s, a, \phi)$ is computed instead.

Consequently, this results in an approximate hallucinating oracle $\tilde{\pi}_{\text{AOR}}$ that computes an approximate action value $Q^{AOR}(s, a) = \mathbb{E}_{\phi \sim P(\phi|s)} \left[Q^{\pi_{\text{AOR}}}(s, a, \phi) \right]$ and greedily maximizes it

$$\tilde{\pi}_{\text{AOR}} \cong \arg\max_{a \in \mathcal{A}} Q^{AOR}(s, a) \tag{2.18}$$

We can then bound the error between value of the hallucinating oracle and the approximate hallucinating oracle by

$$\left|\left| V^{\tilde{\pi}_{\text{OR}}}(s) - V^{\tilde{\pi}_{\text{AOR}}}(s) \right|\right|_{\infty} \leq \epsilon_{\text{app}} \tag{2.19}$$

2.5.3 Component 3: Errors from Imitation

StrOLL imitates the actions demonstrated by the approximate oracle policy. Since our imitation learning back-end is DAgger, we inherit the performance bounds from [402].

The error between the learnt policy $\hat{\pi}$ and the demonstrator policy $\tilde{\pi}_{\text{AOR}}$ can be bounded using Theorem 4.1 in [402].

$$\begin{aligned} \mathbb{E}_{s \sim d_\pi(s)} \left[V^{\tilde{\pi}_{\text{AOR}}}(s) - V^{\hat{\pi}}(s) \right] &\leq \epsilon_{\text{im}} \\ &= A_{\infty} \epsilon_{\text{class}} + \epsilon_{\text{reg}} + O(\frac{1}{N}) \end{aligned} \tag{2.20}$$

where ϵ_{class} is the classification error of the best policy in the policy class on the aggregated dataset, A_{∞} is the maximum advantage w.r.t $\tilde{\pi}_{\text{AOR}}$ and ϵ_{reg} is the average regret.

We can now combine all components

$$\begin{aligned} &\mathbb{E}_{s \sim d_\pi(s)} \left[V^{\pi^*}(s) - V^{\hat{\pi}}(s) \right] \\ &\leq \mathbb{E}_{s \sim d_\pi(s)} \left[V^{\pi^*}(s) - V^{\tilde{\pi}_{\text{AOR}}}(s) \right] + \epsilon_{\text{im}} \\ &\leq \left|\left| V^{\pi^*}(s) - V^{\tilde{\pi}_{\text{AOR}}}(s) \right|\right|_{\infty} + \epsilon_{\text{im}} \\ &\leq \left|\left| V^{\pi^*}(s) - V^{\tilde{\pi}_{\text{OR}}}(s) \right|\right|_{\infty} + \epsilon_{\text{app}} + \epsilon_{\text{im}} \\ &\leq \epsilon_{\text{hal}} + \epsilon_{\text{app}} + \epsilon_{\text{im}} \end{aligned} \tag{2.21}$$

2.5.4 Special Case: Optimal Selector for Independent Bernoulli Edges

In this section we show that under certain conditions, StrOLL indeed contains the optimal policy in it's policy class. We will show here that if $P(\phi)$ is an independent Bernoulli distribution over edges $\mathbf{p}$ and no two paths ξ_i, ξ_j share an edge, the optimal Selector is the one that picks the edge on the shortest path with the lowest probability. As we show later in Section. 2.7.1, this selector is part of the StrOLL policy class. Intuitively, the selector tries to eliminate each path as quickly as possible - the lack of overlap implies the selector does not have to reason over the consequences of eliminating a path.

We first define a selector FailFast:

$$\textsc{FailFast} \equiv \arg\min_{e \in \xi} \mathbf{p}(e) \tag{2.22}$$

We then show that FailFast eliminates a path optimally:

Lemma 2. *Given a path ξ, FailFast minimizes the expected number of edges from ξ that are required to be evaluated to invalidate ξ.*

Proof. Given a path ξ, a sequence of edges
$S = \{e_1, e_2, \ldots, e_n\}$ belonging to the path, and the corresponding priors of the edges being valid $(p_1, p_2, \ldots, p_n)$, let the expected number of edge evaluations to invalidate ξ be $E_{\text{eval}}(S)$ which is given by

$$\begin{aligned} \mathbb{E}_{\mathbf{p}}\left[E_{\text{eval}}(S)\right] &= (1-p_1) + 2p_1(1-p_2) + \ldots \\ &= \sum_{l=1}^{n} \left(\prod_{m=1}^{l-1} p_m \right) (1-p_l)\, l \end{aligned} \tag{2.23}$$

Without loss of generality, let $p_i > p_{i+1}$ for a given i. Consider the alternate sequence of evaluations
$S' = \{e_1, e_2, \ldots, e_{i+1}, e_i \ldots, e_n\}$ where the positions of the edges e_i, e_{i+1} are swapped.

Consider the difference:

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{p}}\left[E_{\text{eval}}(S)\right] - \mathbb{E}_{\mathbf{p}}\left[E_{\text{eval}}(S')\right] \\
&= \ldots + \prod_{m=1}^{i-1} p_m \left[(1-p_i)i + p_i(1-p_{i+1})(i+1)\right] + \ldots \\
&- \ldots + \prod_{m=1}^{i-1} p_m \left[(1-p_{i+1})i + p_{i+1}(1-p_i)(i+1)\right] + \ldots \\
&= \prod_{m=1}^{i-1} p_m \left[-i(p_i - p_{i+1}) + (i+1)(p_i - p_{i+1})\right] \\
&= \prod_{m=1}^{i-1} p_m (p_i - p_{i+1}) \\
&> 0
\end{aligned}
\tag{2.24}
$$

Since each such swap results in monotonic decrease in the objective, there exists an unique fixed point, i.e., the optimal sequence S^* has $p_1 \leq p_2 \leq \ldots \leq p_n$. □

2.6 The Bayesian Lazy Shortest Path Problem

In this section, we ask the question – "What are the minimal number of edge evaluations required to identify the shortest path under uncertainty?". Formally, we define the Bayesian Lazy Shortest Path Problem —given a prior over world $P(\phi)$, what is the minimal number of edges needed to be evaluated until we can certify with certainty that a given path is the shortest feasible path. There is an important distinction from the LazySP paradigm —*we do not need to evaluate every candidate shortest path in sequential order*. The prior $P(\phi)$ maybe such that potential worlds ϕ maybe ruled out by evaluating edges that do not necessarily lie on the shortest path. This can, in theory, give rise to algorithms that with very little evaluation collapse posterior over worlds such that a particular path ξ can be claimed as the shortest feasible.

We adopt a treatment similar to [89, 97] where we show that the problem is equivalent to a problem in Bayesian Active Learning. While the problem is NP-Hard, we show that it is adaptive submodular which we leverage to derive greedy, near-optimal policies. Fig. 2.5 presents an overview of this connection. While the algorithm is simple to implement, it suffers from scalability as it requires explicit enumeration of all possible paths. Nonetheless,

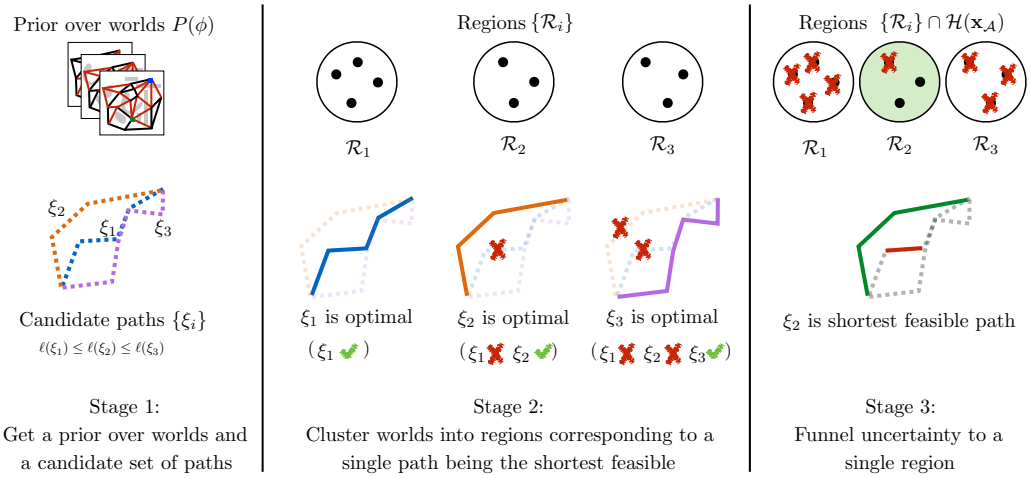


Figure 2.5: Overview of mapping the Bayesian Lazy Shortest Path to Decision Region Determination. Stage 1 illustrates inputs to Problem 2. Stage 2 illustrates the mapping in Section 2.6.2. Stage 3 illustrates the algorithm Algorithm 4

it serves as an important theoretical result and intuition pump for comparing efficacy of various SELECTOR s in LAZYSP.

2.6.1 Preliminaries: Decision Region Determination (DRD)

We first describe the problem of Bayesian Decision Region Determination (DRD). Given a set of hypotheses, a set of tests and a set of *regions* (cluster of hypotheses), the goal is to perform a minimal set of tests to identify a region of hypotheses where the true hypothesis belongs[3]. We formalize this below.

Let $\mathcal{H} = \{h_1, \cdots, h_n\}$ be a set of candidate hypotheses, only one of which is true. We have a prior distribution $P(h)$. Let $\mathcal{T} = \{t_1, \cdots, t_l\}$ be a set of tests. Running a test $t \in \mathcal{T}$ returns a binary outcome $x_t \in \{0, 1\}$ depending on the underlying hypothesis. Thus each hypothesis can be considered as a mapping from tests to outcomes $h : \mathcal{T} \to \{0, 1\}$. The cost of performing a test is $c(t)$[4].

Let a region $\mathcal{R} \subseteq \mathcal{H}$ be a set of a hypotheses. We will use $\{\mathcal{R}_i\}_{i=1}^m$ to denote the a set of

[3]As opposed to the problem of optimal decision tree (ODT), where the true hypothesis must be identified.

[4]If we are only interested in minimizing the number of tests, then $c(t) = 1$ for all t

regions.

For a set of tests $\mathcal{A} \subseteq \mathcal{T}$ that are performed, let the observed outcome vector be denoted by $\mathbf{x}_\mathcal{A}$. Let the version space $\mathcal{H}(\mathbf{x}_\mathcal{A})$ be the set of hypotheses consistent with outcome $\mathbf{x}_\mathcal{A}$, i.e. $\mathcal{H}(\mathbf{x}_\mathcal{A}) = \{h \in \mathcal{H} \mid \forall t \in \mathcal{A}, h(t) = \mathbf{x}_\mathcal{A}(t)\}$. We assume that at least one hypothesis is true, i.e. $|\mathcal{H}(\mathbf{x}_\mathcal{T})| \geq 1$.

We define a policy π as a mapping from the current outcome vector $\mathbf{x}_\mathcal{A}$ to the next test to select. A policy terminates when at least one region is valid, or all regions are invalid. Let h be the underlying hypothesis on which it is evaluated. Denote the outcome vector of a policy π as $\mathbf{x}_\mathcal{A}(\pi, h)$. The expected cost of a policy π is $c(\pi) = \mathbb{E}_h[c(\mathbf{x}_\mathcal{A}(\pi, h)]$ where $c(\mathbf{x}_\mathcal{A})$ is the cost of all tests $t \in \mathcal{A}$. The objective is to compute a policy π^* with minimum cost such that all the uncertainty is funneled into one region,

$$\pi^* \in \arg\min_\pi c(\pi) \text{ s.t } \forall h, \exists \mathcal{R}_d \ : \ \mathcal{H}(\mathbf{x}_\mathcal{A}) \subseteq \mathcal{R}_d \tag{2.25}$$

Obtaining an approximate policy π for which $c(\pi) \leq c(\pi^*).O(\log n)$ is NP-hard [69]. Fortunately, variants of DRD [168, 216] has been shown to have the property of adaptive submodularity [167]. This property implies that greedy policies are near-optimal. We harness this property in Section 2.6.3.

2.6.2 Bayesian Lazy Shortest Path as a DRD problem

We formally define the Bayesian Lazy Shortest Path problem, noting a distinction with the Optimal Selector problem Problem 1. We are given a prior distribution of worlds, $P(\phi)$. We are given a set of all candidate shortest paths $\{\xi_i\}$. Let $E_{\text{eval}}(\phi, \pi)$ denote the edges evaluated and outcomes by a policy π on world ϕ. Let $c(E_{\text{eval}}(\phi, \pi))$ denote the cost of edge evaluation. Let $P(\xi_i \text{ is feasible} | E_{\text{eval}}(\phi, \pi))$ be the posterior probability of a path being feasible. We then have the following problem:

Problem 2 (Bayesian Lazy Shortest Path (SP))**.** *Minimize the cost of edge evaluation until a path is declared to be shortest feasible with probability* 1*, i.e. it is deemed feasible and all*

shorter paths are deemed infeasible.

$$\begin{aligned} \min \quad & \mathbb{E}_{\phi \sim P(\phi)} \left[cE_{\text{eval}}(\phi, \pi) \right] \\ s.t. \quad & \exists \xi^* : P(\xi^* \textit{ is feasible} | E_{\text{eval}}(\phi, \pi)) = 1.0 \\ & \forall \xi_i : \ \ell(\xi_i) \leq \ell(\xi^*), \ P(\xi_i \textit{ is feasible} | E_{\text{eval}}(\phi, \pi)) = 0.0 \end{aligned} \tag{2.26}$$

We now map this problem to Bayesian Decision Region Determination in Table 2.3. Notably, each region consists of the number of worlds for which ξ_i is the shortest path. This mapping is also illustrated in Fig. 2.5

Table 2.3: Mapping Bayesian Lazy Shortest Path to DRD

Bayesian Lazy Shortest Path	**Decision Region Determination**
World (ϕ)	Hypothesis (h)
Evaluate edge (e)	Execute test (t)
Set of worlds for which path ξ_i is shortest feasible	Region $\mathcal{R}_i \subset \mathcal{H}$
Cost of evaluating edge ($c(e)$)	Cost of executing test ($c(t)$)

2.6.3 Efficienty solving DRD using the EC^2 algorithm

The DRD problem in (2.25) has a special property – *the regions are disjoint.* [168] addresses this disjoint setting and propose a greedy algorithm, EC^2, that is near-optimal. The idea behind EC^2 is elegant —define a graph with edges between hypotheses that we care to distinguish between. Tests 'cut' edges inconsistent with outcomes, distinguishing between the two hypothesis. The aim is to cut all inconsistent edges with minimum expected incurred cost. We adopt this algorithm to solve the Bayesian Shortest Path Problem.

The EC^2 algorithm defines a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, illustrated in Fig. 2.5, where the nodes are hypotheses and edges are between hypotheses in *different decision regions* $\mathcal{E} = \cup_{i \neq j} \{(h, h') \mid h \in \mathcal{R}_i, h' \in \mathcal{R}_j\}$.

The weight of an edge is $w((h,h')) = P(h)P(h')$. An edge is said to be 'cut' by an observation if either hypothesis is inconsistent with the observation, i.e. weight is 0. We can define the weight for all existing edges as a sum over individual weights $w(\mathcal{E}) = \sum_{e\in\mathcal{E}} w(e)$. Notice that if we drive $w(\mathcal{E}) \to 0$, we effectively disambiguate between hypothesis in different regions, i.e. we solve the DRD problem.

We define $w(\{\mathcal{R}_i\})$ as the total weight of edges between regions. Since regions are disjoint, the total weight can be computed efficiently as

$$\begin{aligned} w(\{\mathcal{R}_i\}) &= \frac{1}{2}\left(\sum_{i\neq j} P(\mathcal{R}_i)P(\mathcal{R}_j)\right) \\ &= \frac{1}{2}\left((\sum_i P(\mathcal{R}_i))^2 - \sum_i P(\mathcal{R}_i)^2\right) \end{aligned} \tag{2.27}$$

We can then define an objective function $f_{\mathrm{EC}}(\mathbf{x}_{\mathcal{A}})$ that measures progress, i.e., how many edges have been cut by looking at the weight of the pruned regions to the original regions, i.e.

$$f_{\mathrm{EC}}(\mathbf{x}_{\mathcal{A}}) = 1 - \frac{w(\{\mathcal{R}_i\} \cap \mathcal{H}(\mathbf{x}_{\mathcal{A}}))}{w(\{\mathcal{R}_i\})} \tag{2.28}$$

The objective function $f_{\mathrm{EC}}(\mathbf{x}_{\mathcal{A}})$ is initially 0 when no edges have been cut. When all the edges have been cut, i.e., $w(\{\mathcal{R}_i\} \cap \mathcal{H}(\mathbf{x}_{\mathcal{A}})) = 0$, the objective is $f_{\mathrm{EC}}(\mathbf{x}_{\mathcal{A}}) = 1$. Our goal is to maximize this objective while incurring minimum cost of evaluating tests. We do so efficiently by observing that $f_{\mathrm{EC}}()$ is *adaptive submodular*:

Lemma 3. *The objective function $f_{\mathrm{EC}}()$ is strongly adaptive monotone and adaptive submodular*

Proof. See proof of Proposition 2 in [168] □

The property above suggest a powerful technique — *greedily maximizing $f_{\mathrm{EC}}()$ is near-optimal* [167]. EC^2 does just this. First note that given a test t, we can compute the expected marginal gain of a test as

$$\Delta(t \mid \mathbf{x}_{\mathcal{A}}) = \mathbb{E}_{x_t}\left[f_{\mathrm{EC}}\left(\mathbf{x}_{\mathcal{A}\cup\{t\}}\right) - f_{\mathrm{EC}}(\mathbf{x}_{\mathcal{A}})\right] \tag{2.29}$$

A greedy policy π_{EC} selects a test $t^* \in \arg\max_t \frac{\Delta(t \mid \mathbf{x}_\mathcal{A})}{c(t)}$. Expanding this we get:

$$\begin{aligned}
&\arg\max_t \frac{1}{c(t)} \mathbb{E}_{x_t}\left[f_{\text{EC}}\left(\mathbf{x}_{\mathcal{A}\cup\{t\}}\right) - f_{\text{EC}}\left(\mathbf{x}_\mathcal{A}\right)\right] \\
&= \frac{1}{c(t)} \mathbb{E}_{x_t}\left[\frac{w(\{\mathcal{R}_i\} \cap \mathcal{H}(\mathbf{x}_\mathcal{A})) - w(\{\mathcal{R}_i\} \cap \mathcal{H}(\mathbf{x}_{\mathcal{A}\cup\{t\}}))}{w(\{\mathcal{R}_i\})}\right] \\
&= \frac{1}{c(t)} \mathbb{E}_{x_t}\left[w(\{\mathcal{R}_i\} \cap \mathcal{H}(\mathbf{x}_\mathcal{A})) - w(\{\mathcal{R}_i\} \cap \mathcal{H}(\mathbf{x}_{\mathcal{A}\cup\{t\}}))\right]
\end{aligned} \tag{2.30}$$

Theorem 4. *The greedy policy π_{EC} is near-optimal, i.e.*

$$c(\pi_{\text{EC}}) \leq (2\log(1/p_{\min}))c(\pi^*) \tag{2.31}$$

where $p_{\min} = \min_{h\in\mathcal{H}} P(h)$ is the minimum prior probability for any hypothesis and π^* is the optimal policy for the DRD problem.

Proof. See proof of Theorem 3 in [168]. □

2.6.4 Algorithm

We are now ready to apply the mapping in Section 2.6.2 to derive a greedy, near-optimal policy for evaluating edges until the shortest path is found. Algorithm. 4 describes the algorithm. It takes as input a set of candidate paths $\{\xi_i\}$and a prior over worlds $P(\phi)$. It then clusters the worlds according to regions $\mathcal{R}_i$ such that all such worlds correspond to ξ_i to be the shortest feasible path. We define two lambda functions. First, computes a posterior over region $P(\mathcal{R}_i|\mathbf{x}_\mathcal{A})$ given a set of edge evaluation outcomes $\mathbf{x}_\mathcal{A}$. Secondly, compute a weight function $w(\{\mathcal{R}_i\} \cap \mathcal{H}(\mathbf{x}_\mathcal{A}))$ using Eq. 2.27. This is the weight of all remaining edges that need to be cut. It then iteratively evaluates edges. In each iteration, it greedily selects an edge using Eq. 2.30. The edge is evaluated, outcome is received and $\mathbf{x}_\mathcal{A}$ updated. If the uncertainty over worlds has been funneled into a single region $\mathcal{R}_i$, then we know for certain the shortest feasible path is ξ_i. The iteration terminates and this path is returned.

2.6.5 Practical Challenges

Practically, implementing Algorithm. 4 is difficult due to a number of challenges:

Algorithm 4: Bayesian Lazy Shortest Path

Input : Set of candidate paths $\{\xi_i\}$, prior over worlds $P(\phi)$

Output : Shortest feasible path ξ^*

Create regions $\mathcal{R}_i = \{\phi \mid \xi_i \text{ shortest feasible path in } \phi\}$

Create function to compute posterior over regions

$P(\mathcal{R}_i|\mathbf{x}_\mathcal{A}) \leftarrow \sum_{\phi \in \mathcal{R}_i} P(\phi|\phi \text{ consistent with } \mathbf{x}_\mathcal{A})$

Create function to compute weights equation 2.27

$w(\{\mathcal{R}_i\} \cap \mathcal{H}(\mathbf{x}_\mathcal{A})) \leftarrow \frac{1}{2}\left((\sum_i P(\mathcal{R}_i|\mathbf{x}_\mathcal{A}))^2 - \sum_i P(\mathcal{R}_i|\mathbf{x}_\mathcal{A})^2\right)$

Initialize list of (edges evaluated, outcome) $\mathbf{x}_\mathcal{A} \leftarrow \emptyset$

repeat

  Select edge e^* that maximizes expected marginal gain equation 2.30

  $\arg\max_e \mathbb{E}_{x_e}\left[\frac{w(\{\mathcal{R}_i\}\cap\mathcal{H}(\mathbf{x}_\mathcal{A}))-w(\{\mathcal{R}_i\}\cap\mathcal{H}(\mathbf{x}_{\mathcal{A}\cup\{e\}}))}{c(e)}\right]$

  Evaluate e^* and observe outcome x_{e^*}

  Add $\mathbf{x}_\mathcal{A} \leftarrow \mathbf{x}_\mathcal{A} \cup (e^*, x_{e^*})$

until *shortest feasible path found, i.e.* $\exists \mathcal{R}_i, \mathcal{H}(\mathbf{x}_\mathcal{A}) \subseteq \mathcal{R}_i$;

return $\{\xi^* \leftarrow \xi_i\}$;

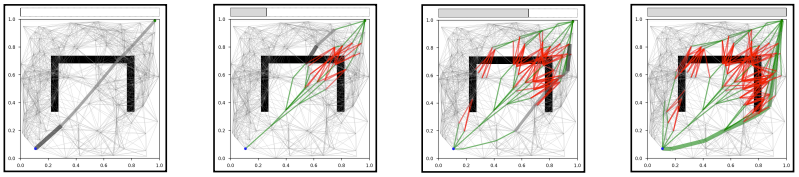
1. It requires enumerating the full set of candidate paths $\{\xi_i\}$ which can be $O(E!)$.

2. At each iteration, the complexity is $O(|\mathcal{R}|\,|\phi|)$, i.e. the number of paths times the number of worlds, which makes computation quite expensive.

3. It assumes realizability of the world at test time.

Depsite these shortcomings, it offers a very clean, analytic policy that could potentially be used if one were dealing with a small number of candidate paths. We leave exploring such algorithms for future work.

(a) ALTERNATE



(b) STROLL

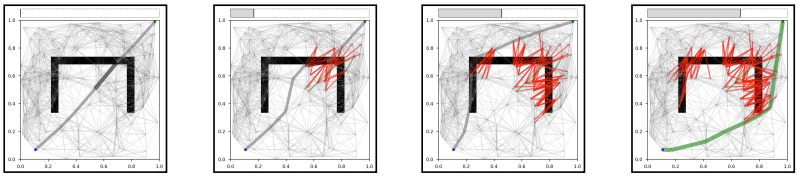


Figure 2.6: Snapshots of search progress using uniformed versus learned selectors on a BUGTRAP environment. The thicker gray edges depict the candidate shortest path with current selected edge in dark gray. Edges evaluated to be valid are shown in green and invalid edges are in red. The uninformed ALTERNATE selector evaluates several valid edges thus wasting search effort whereas STROLL focuses on edges more likely in collision thus leading to fewer overall edge evaluations as depicted in the top progress bar.

2.7 Experiments

2.7.1 Experimental Setup

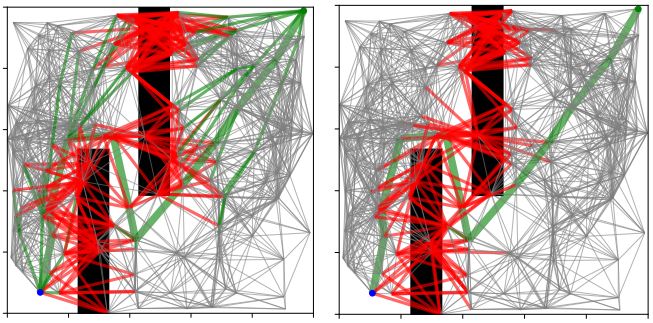
We use datasets from [97] in our experiments. The 2D datasets contain graphs with approximately 1600-5000 edges and varied obstacle distributions. The two 7D datasets involve a robot arm planning for a reaching task in clutter with large graphs containing 33286 edges.

Learning Details

We only consider policies that are a linear combination of a minimal set of features, where each feature is a different motion planning heuristic. The features we consider are:

1. PRIOR- the prior probability of an edge being invalid calculated over the training dataset.

2. POSTERIOR- the posterior probability of an edge being invalid given collision checks done thus far (described below)



(a) ALTERNATE $|E_{\text{eval}}| = 292$ (b) STROLL $|E_{\text{eval}}| = 180$

Figure 2.7: Edges evaluated (green valid, red invalid) on a world from BAFFLE. (a) ALTERNATE evaluates several valid edges (b) STROLL evaluates many fewer edges, all of which are invalid and eliminate a large number of paths.

3. LOCATION- score ranging from 1 (first unchecked edge) to 0 (last unchecked edge).

4. Δ-LENGTH- hallucinate that an edge is invalid, then calculate the difference in length of new shortest path compared with the current shortest path.

5. Δ-EVAL- hallucinate that an edge is invalid, the calculate the fraction of unevaluated edges on the new shortest path.

6. PΔ-LENGTH- calculated as POSTERIOR $\times$ Δ-LENGTH, it weighs the Δ-LENGTH of an edge with the probability of it being invalid and is effective in practice (Table 6.1).

POSTERIOR Selector: We define the posterior selector used in a manner similar to FAILFAST from Section 2.5.4

$$\text{POSTFAILFAST} \equiv \arg\min_{e \in \xi} \mathbf{p}(e|s_t) \tag{2.32}$$

where s_t is the state of search at time t. We aproximate the posterior using the training dataset of N worlds similar to [89, 97] as follows - for each training world ϕ_i a score z_i is calculated based on the discrepancy between s_t and the s_{ti} where the latter is what the state of the search would be if agent were operating in ϕ_i, i.e

$$z_i = -|s_t - s_{ti}|$$

where, the difference follows directly from defintion in Section 2.2.3. The probability for ϕ_i is then given by a softmax over training worlds,

$$P(\phi_i|s_t) = \frac{e^{z_i}}{\sum_{k=0}^{N} e^{z_k}}$$

Then, for every $e \in \xi$,

$$\mathbf{p}(e|s_t) = \sum_{k=0}^{N} P(\phi_k|s_t)\phi_i(e) \tag{2.33}$$

2.7.2 Baselines

We compare our approach to common heuristics used in LazySP as described in Section 2.2.2. We also analyze the improvement in performance as compared to vanilla behavior cloning of the oracle and reinforcement learning from scratch.

2.7.3 Analysis of Overall Performance

O 1. *StrOLL has consistently strong performance across different datasets.*

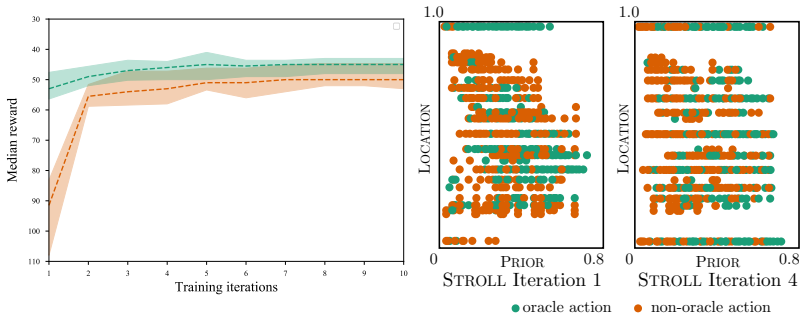
Fig. 2.10 shows that StrOLL is able to learn policies competitive with other motion planning heuristics. No other heuristic has as consistent a performance across datasets.

O 2. *The learner focuses collision checking on edges that are highly likely to be invalid and have a high measure of centrality.*

Fig. 2.11 shows the activation of different features across datasets. The learner places high importance on Posterior, Δ-Length and PΔ-Length. Posterior is an approximate likelihood of an edge being invalid and Δ-Length is an approximate measure of centrality i.e. edges with large Δ-Length have large number of paths passing through them (Note that the converse may not always apply).

O 3. *On datasets with strong correlations among edges, heuristics that take obstacle distribution into account outperform uninformed heuristics, and StrOLL is able to learn significantly better policies than uninformed heuristics.*

Examples of such datasets are gate, baffle, bugtrap and blob. Here, StrOLL and StrOLL-R eliminate a large number of paths by only evaluating edges which are highly



(a) (b)

Figure 2.8: (a) StROLL-R (green) vs StROLL (orange) (b) Densification of data.

likely to be in collision and have several paths passing through them (Figs. 2.6, 2.7). In the 7D datasets, obstacles are highly concentrated near the goal region, which explains the strong performance of the uninformed BACKWARD selector. However, due to a very large number of edges and limited training sets, POSTERIOR and Δ-LENGTH are inaccurate causing the learner to fail to outperform BACKWARD.

O 4. *On datasets with uniformly spread obstacles, uninformed heuristics can perform better than StROLL.*

Examples of such datasets are TWOWALL and FOREST where the lack of structures makes features such as posterior uninformative. This combined with the non-realizability of the oracle makes it difficult for StROLL to learn a strong policy.

2.7.4 Case Studies

Q 1. *How does performance vary with training data?*

Fig. 2.8(a) shows the improvement in median validation reward with an increasing number of training iterations. Also, Fig. 2.8(b) shows that with more iterations, the learner visits diverse parts of the state-space on x-axis not visited by the oracle.

Q 2. *How significant is the impact of heuristic roll-in on stabilizing learning in high-dimensional problems?*

Fig. 2.8 shows a comparison of the median validation return per iteration using StROLL

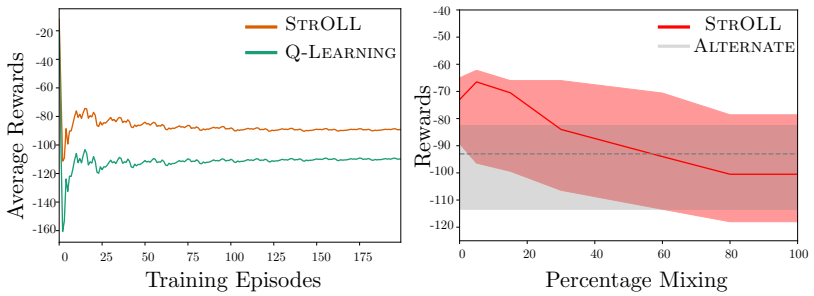


Figure 2.9: (a) Running average reward for 200 episodes of training. Q learning suffers due to large state space and sparse rewards. (b) Performance on validation set of 200 worlds with contamination from different distribution.

	Oracle	Back-ward	Alter-nate	Fail-Fast	PostFail-Fast	PΔ-Length	Super-vised	StrOLL	StrOLL-R
2D Geometric Planning									
OneWall	$80.0^{+6.0}_{-48.0}$	$87.0^{+8.4}_{-41}$	$112.0^{+12.8}_{-60.0}$	$82.0^{+3.0}_{-47.0}$	$81.0^{+3.0}_{-49.0}$	$85.0^{+6.8}_{-52.6.0}$	$79.0^{+3.0}_{-45.0}$	$79.0^{+5.0}_{-44.8}$	$79.0^{+5.0}_{-44.8}$
TwoWall	$107.0^{+23.0}_{-0.0}$	$199.0^{+8.0}_{-19.0}$	$138.0^{+7.0}_{-2.0}$	$178.0^{+0.0}_{-6.0}$	$177.0^{+0.0}_{-7.0}$	$120.0^{+19.0}_{-0.0}$	$177.0^{+0.0}_{-6.0}$	$177.0^{+0.0}_{-6.0}$	$170.0^{+12.2}_{-0.0}$
Forest	$90^{+14.4}_{-10.0}$	$128.0^{+15.0}_{-16.4}$	$115.0^{+12.0}_{-13.2}$	$135.0^{+13.0}_{-16.0}$	$116.0^{+13.2}_{-16.4}$	$102.0^{+15.0}_{-12.0}$	$117.0^{+19.2}_{-17.0}$	$115.0^{+20.0}_{-15.0}$	$115.0^{+21.2}_{-13.4}$
Gate	$50.0^{+6.0}_{-9.0}$	$74.0^{+8.0}_{-9.0}$	$75.0^{+14.2}_{-6.2}$	$60.0^{+8.0}_{-6.2}$	$50.0^{+7.0}_{-8.2}$	$53.0^{+7.0}_{-9.2}$	$50.0^{+7.0}_{-7.2}$	$48.0^{+10.0}_{-7.2}$	$48.0^{+9.2}_{-9.2}$
Maze	$537.0^{+37.0}_{-24.6}$	$668.5^{+40.3}_{-56.1}$	$613.0^{+39.6}_{-33}$	$512^{+52.2}_{-34.0}$	$516.5^{+33.70}_{-36.50}$	$529.0^{+40.0}_{-37.2}$	$502.5^{+58.7}_{-28.2}$	$512.0^{+42.0}_{-31.0}$	$554.0^{+52.2}_{-59.0}$
Baffle	$219.0^{+18.0}_{-12.0}$	$244.0^{+14.6}_{-6.0}$	$311.0^{+8.0}_{-15.6}$	$232.0^{+6.0}_{-12.0}$	$211.0^{+7.8}_{-6.0}$	$230.0^{+18.0}_{-17.0}$	$206.0^{+6.8}_{-3.0}$	$205.0^{+6.0}_{-3.0}$	$207.0^{+7.0}_{-2.0}$
Bug-trap	$77.0^{+12.0}_{-9.4}$	$104.0^{+6.0}_{-14.0}$	$112.5^{+16.9}_{-11.5}$	$90.5^{+10.9}_{-13.5}$	$75.5^{+16.5}_{-6.5}$	$84.5^{+12.9}_{-10.5}$	$75.0^{+15.4}_{-6.4}$	$75.0^{+15.4}_{-6.4}$	$75.0^{+15.4}_{-6.4}$
Blob	$72.0^{+12.0}_{-4.0}$	$92.0^{+5.4}_{-3.4}$	$109.0^{+5.0}_{-7.0}$	$70.0^{+6.0}_{-6.0}$	$70.0^{+6.0}_{-6.0}$	$80.0^{+9.0}_{-3.0}$	$72.0^{+5.8}_{8.0}$	$70.0^{+6.0}_{-6.0}$	$70.0^{+6.0}_{-6.0}$
7D Manipulation Planning									
Clut-ter1	$35.5^{+1.5}_{-1.5}$	$38.0^{+12.0}_{-10.0}$	$44.0^{+14.2}_{-4.0}$	$92.0^{+5.0}_{-0.0}$	$88.0^{+9.6}_{-0.0}$	$37.5^{+2.1}_{-1.5}$	$95.5^{+17.7}_{-11.1}$	$50.0^{+3.0}_{-6.0}$	$45.0^{+2.0}_{-1.6}$
Clut-ter2	$34.0^{+1.0}_{-2.0}$	$32.0^{+3.0}_{-4.0}$	$41.0^{+2.0}_{-2.2}$	$85.0^{+0.0}_{-3.0}$	$84.0^{+0.0}_{-2.0}$	$37.0^{+0.0}_{-5.0}$	$104.0^{+6.2}_{-6.8}$	$47.0^{+2.0}_{-11.2}$	$44.0^{+5.0}_{-7.2}$

Figure 2.10: Edges evaluated by different algorithms across different datasets (median, upper and lower C.I on 200 held-out environments). Highlighted is the best performing selector in terms of median score not counting the oracle.

versus StrOLL-R on clutter1 dataset. Heuristic roll-in helps converge to a better policy in lesser number of iterations. Interestingly, the policy learned in the first iteration of StrOLL-R is significantly better than StrOLL, demonstrating the stabilizing effects of

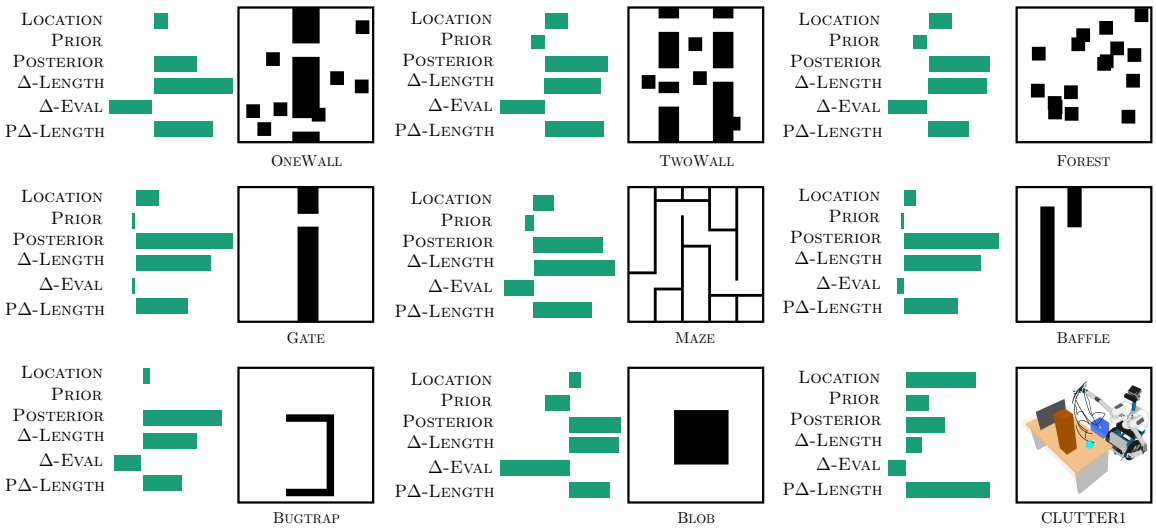


Figure 2.11: Weight bins depicting relative importance of each feature learned by the learner. StROLL focuses on edges that are highly likely to be invalid and have high measure of centrality.

heuristic roll-in.

Q 3. *How does performance compare to reinforcement learning with function approximation?*

Fig. 2.9(a) shows training curves for StROLL and Q-Learning with linear function approximation and experience replay. StROLL is more sample efficient and converges to a competitive policy faster.

Q 4. *How does performance vary with train-test mismatch?*

Fig. 2.9(b) shows a stress-test of a policy learned on one wall by running it on a validation set which is increasingly contaminated by environments from forest. The learned policy performs better than the best uninformed heuristic on forest for up to 60% contamination.

2.8 Related Work

In domains where edge evaluations are expensive and dominate planning time, a *lazy approach* is often employed [50] wherein the graph is constructed *without* testing if edges are collision-free. LazySP [121] extends the graph all the way to the goal, before evaluating

edges. LWA* [105] extends the graph only a single step before evaluation. (LRA*) [319] is able to trade-off between them by allowing the search to go to an arbitrary lookahead. The principle of laziness is reflected in similar techniques for randomized search [163, 183].

Several previous works investigated leveraging priors in search. FuzzyPRM [353] evaluates paths that minimize the probability of collision. The Anytime Edge Evaluation (AEE*) framework [347] uses an anytime strategy for edge evaluation informed by priors. BISECT [97] and DIRECT [89] casts search as Bayesian active learning to derive edge evaluation policies. However, none of these approaches formalized the problem of minimizing edge evaluation till the shortest feasible path is found. Our paper formalizes this problem, examines it in both the Bayesian setting as well as offers a practical learning based approach by leveraging imitation learning.

Efficient collision checking has its own history in the context of motion planning. Other approaches model belief over the configuration space to speed-up collision checking [101, 206], sample vertices in promising regions [47] or grow the search tree to explore the configuration space [61, 204, 270]. However, these approaches make geometric assumptions and rely on domain knowledge. We work directly with graphs and are agnostic to the domain.

Several recent works use imitation learning [401, 402, 451] to bootstrap reinforcement learning. THOR [450] performs a multi-step search to gain advantage over the reference policy. LOKI [87] switches from IL to RL. Imitation of clairvoyant oracles has been used in multiple domains like information gathering [95], heuristic search [43], and MPC [227, 463].

2.9 Discussion

In this chapter, we examined the problem of minimizing edge evaluations in lazy search on a distribution of worlds. We first formulated the problem of deciding which edge to evaluate as an MDP and presented an algorithm to learn policies by imitating clairvoyant oracles, which, if the world is known, can optimally evaluate edges. Further, we provide a theoretical analysis of our proposed framework that details different sources of approximation error. We also analyzed the problem in the Bayesian setting and draw a novel connection to Bayesian Active Learning. However, the current approach has certain limitations. First, we only consider learning edge selector policies that are a linear combination of heuristic edge

selectors. Although we found using baseline heuristics as features to yield strong results, the limited representational power of hand designed features can affect the performance of the policy. This can be improved in the future by considering more general function approximators such as Graph Neural Networks [510] to represent the selector policy. Second, while imitation learning of clairvoyant oracles is effective, the approach may be further improved through reinforcement learning [87, 450] since in in practice we do not use the exact oracle but a sub-optimal approximation which means that errors in the oracle will transfer to the learner, limiting performance.

Chapter 3

DIFFERENTIABLE GAUSSIAN PROCESS MOTION PLANNING

3.1 Introduction

In the previous chapter, we presented an imitation learning framework that leverages past experience to accelerate lazy search-based motion planning. This chapter focuses on how past experience can also be used to improve the performance of another class of planning algorithms based on gradient-based trajectory optimization. Trajectory optimization is a powerful approach for effectively solving the planning problem and state-of-the-art algorithms can find smooth, collision free trajectories in almost real-time for complex systems such as robot manipulators [336, 393, 415]. Although these approaches are easy to implement and generally applicable to a wide range of tasks, they have certain parameters which can strongly affect their performance in practice. This leads to two major problems: (i) there is no formal way of setting parameters for a given task and thus requires manual tuning that can be time-consuming and arduous; and (ii) the planner needs to be re-tuned if the distribution of obstacles in the environment changes significantly, making it brittle in practice, i.e. a planner that works well on one type of environment might completely fail on another. The above issues need to be addressed in order to create flexible robotic systems that can work seamlessly across a variety of environments and tasks. In order to do so, we ask the following question: can we leverage past experience to learn parameters of the planner in a way that directly improves its expected performance over the distribution of problems it encounters?

This work focuses on GPMP2 [131], a state-of-the-art motion planning algorithm that formulates trajectory optimization as inference on a factor graph and finds solutions by solving a nonlinear least squares optimization problem, where the inverse covariances of the factors manifest as weights in the objective function. While GPMP2 has been shown to be a leading optimization-based approach to motion planning [336], in Section 3.2.2 we illustrate

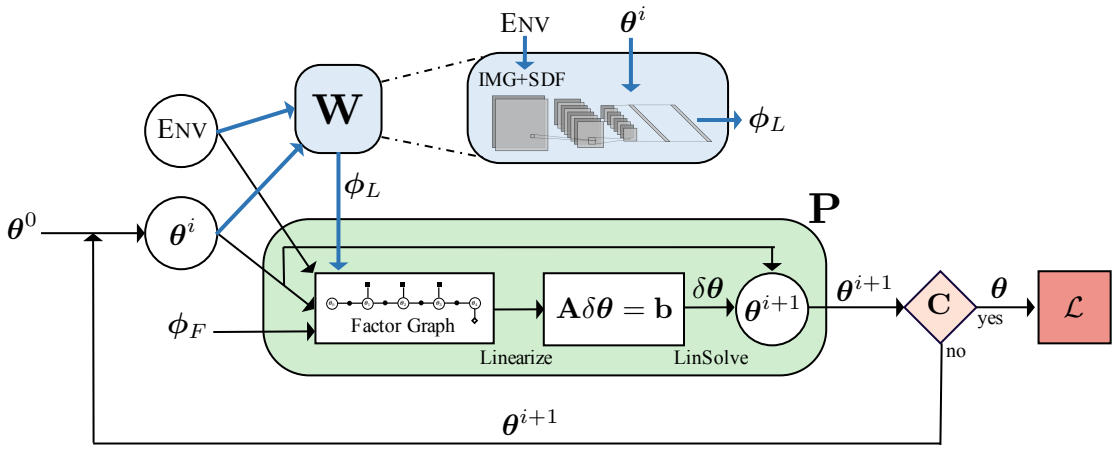


Figure 3.1: The computational graph of dGPMP2 where $\boldsymbol{\phi}_F$ are user defined planning parameters that are fixed and $\boldsymbol{\phi}_L$ are learned planning parameters. See Section 3.3 for details.

its sensitivity to its objective function parameters (specifically factor covariances). To contend with this problem, we leverage the key insight that GPMP2 can be rebuilt as a *fully differentiable computational graph* and learn the parameters for its objective function from data in an end-to-end fashion. This allows us to develop a learning strategy that can improve GPMP2's performance on a given distribution of problems. Our differentiable version can be trained in a self-supervised manner or from expert demonstrations to predict covariances that are time and space varying, in contrast to fixed, hand-tuned covariances, as used in the vanilla approach. Building on top of a structured planner offers interpretability and allows us to explicitly incorporate planning constraints such as collision avoidance and velocity limits. We perform several experiments in simulated 2D environments to demonstrate the benefits of our approach which we call Differentiable GPMP2 (dGPMP2), illustrated in Fig. 3.1.

3.2 Problem Formulation

We begin by reviewing the GPMP2 [336] planner that we will later reconstruct as a differentiable computational graph. Then, we discuss limitations of GPMP2 with respect to its sensitivity to objective function parameters, thus motivating our learning algorithm.

3.2.1 Planning as inference on factor graphs

We take a probabilistic inference perspective on motion planning as described in the GPMP2 framework [131]. The planning problem is posed as computing the maximum a posteriori (MAP) trajectory given a prior over trajectories and a likelihood of events of interest that the trajectory must satisfy. By selecting appropriate distributions, sparsity can be induced in the MAP problem, which allows for efficient inference. Following [131], we describe the essential components of GPMP2 here.

The Prior: In GPMP2, a continuous-time Gaussian process (GP) is used to define a prior distribution over trajectories, $\boldsymbol{\theta}(t) \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\mathcal{K}}(t,t'))$, where $\boldsymbol{\mu}(t)$ is the mean function and $\boldsymbol{\mathcal{K}}(t,t')$ is the kernel. For the purposes of our approach, we represent the trajectory using N support states, $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N]^T$ at different points in time and define the mean vector and covariance matrix as

$$\boldsymbol{\mu} = [\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_N]^T, \quad \boldsymbol{\mathcal{K}} = [\boldsymbol{\mathcal{K}}(t_i, t_j)]\Big|_{ij, 1 \leq i,j \leq N} \tag{3.1}$$

and this GP defines a prior on the space of trajectories

$$P(\boldsymbol{\theta}) \propto \exp\Big\{ -\frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\mu}\|^2_{\boldsymbol{\mathcal{K}}} \Big\}, \tag{3.2}$$

where $\|\boldsymbol{\theta} - \boldsymbol{\mu}\|^2_{\boldsymbol{\mathcal{K}}} \doteq (\boldsymbol{\theta} - \boldsymbol{\mu})^T \boldsymbol{\mathcal{K}}^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu})$ is the Mahalanobis distance. The GP prior distribution is generated by an LTV-SDE [36]

$$\dot{\boldsymbol{\theta}} = \boldsymbol{A}(t)\boldsymbol{\theta}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) + \boldsymbol{w}(t), \tag{3.3}$$

where $\boldsymbol{A}(t)$ and $\boldsymbol{B}(t)$ are system matrices, $\boldsymbol{u}(t)$ is a bias term, and $\boldsymbol{w}(t) \propto \mathcal{GP}(\mathbf{0}, \mathbf{Q}_c \delta(t - t'))$ is a white noise process with $\mathbf{Q}_c$ being the power spectral density matrix of the system and δ being the Dirac delta function. The first and second order moments of the solution to Eq. equation 3.3 gives us the mean and covariance of the desired GP prior. The resulting inverse kernel matrix of the GP has an exactly sparse block-tridiagonal structure making it ideal for fast inference. Here, we use the constant velocity prior model, where the covariance for a single time step is specified by

$$\mathbf{Q}_{t_i, t_{i+1}} = \begin{bmatrix} \frac{1}{3}\Delta t_i^3 \mathbf{Q}_c & \frac{1}{2}\Delta t_i^2 \mathbf{Q}_c \\ \frac{1}{2}\Delta t_i^2 \mathbf{Q}_c & \Delta t_i \mathbf{Q}_c \end{bmatrix}, \tag{3.4}$$

where $\Delta t_i = t_{i+1} - t_i$. The full GP covariance is obtained by composing $\mathbf{Q}_{t_i,t_{i+1}}$ at every time step along with the start and goal covariances, $\boldsymbol{K}_s$ and $\boldsymbol{K}_v$. Please refer to [131] and [36] for details. One important thing to note here is that the GP prior covariance is completely parameterized by the power spectral density matrix $\mathbf{Q}_c$.

The Likelihood function: The likelihood function is used to capture planning requirements in the form of events $\mathbf{e}$ that the trajectory must satisfy. These include constraints such as collision avoidance, joint or velocity limits, or other task relevant objectives. We define the likelihood function as a distribution in the exponential family given by

$$L(\boldsymbol{\theta}; \mathbf{e}) \propto \exp\Big\{ -\frac{1}{2} \|\boldsymbol{h}(\boldsymbol{\theta})\|^2_{\boldsymbol{\Sigma}} \Big\}, \tag{3.5}$$

where $\boldsymbol{h}(\boldsymbol{\theta})$ is a vector-valued cost function and $\mathbf{e}$ are the events of interest.

Inference: Given the prior and likelihood, the MAP problem can be solved as

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \{P(\boldsymbol{\theta}|\mathbf{e})\} = \arg\min_{\boldsymbol{\theta}} \big\{ -\log\big(P(\boldsymbol{\theta}) L(\boldsymbol{\theta};\mathbf{e})\big) \big\} \\ \boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} \Big\{ \frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\mu}\|^2_{\boldsymbol{\mathcal{K}}} + \frac{1}{2} \|\boldsymbol{h}(\boldsymbol{\theta})\|^2_{\boldsymbol{\Sigma}} \Big\}. \end{aligned} \tag{3.6}$$

In general, $\boldsymbol{h}(\boldsymbol{\theta})$ can be non-linear and thus the above equation is a Nonlinear Least Squares (NLLS) problem which can be solved using iterative approaches like Gauss-Newton or Levenberg-Marquardt (LM) algorithms. At any iteration i, these algorithms proceed by first linearizing the cost function around the current estimate of the trajectory, $\boldsymbol{\theta}^i$, using a Taylor expansion $\boldsymbol{h}(\boldsymbol{\theta}) = \boldsymbol{h}(\boldsymbol{\theta}^i) + \mathbf{H}\delta\boldsymbol{\theta}$, where $\mathbf{H} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^i}$ and then solving the following linear system to find the update, $\delta\boldsymbol{\theta}$:

$$\big(\boldsymbol{\mathcal{K}}^{-1} + \mathbf{H}^T \boldsymbol{\Sigma}^{-1} \mathbf{H}\big)\, \delta\boldsymbol{\theta} = -\boldsymbol{\mathcal{K}}^{-1}(\boldsymbol{\theta}^i - \boldsymbol{\mu}) - \mathbf{H}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{h}(\boldsymbol{\theta}^i). \tag{3.7}$$

Gauss-Newton optimization in particular updates the current estimate with the following rule

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i + \delta\boldsymbol{\theta}. \tag{3.8}$$

GPMP2 exploits the sparsity of the linear system in Eq. 3.7 to formulate MAP inference on a factor graph and solve it efficiently. While GPMP2 is a state-of-the-art method that

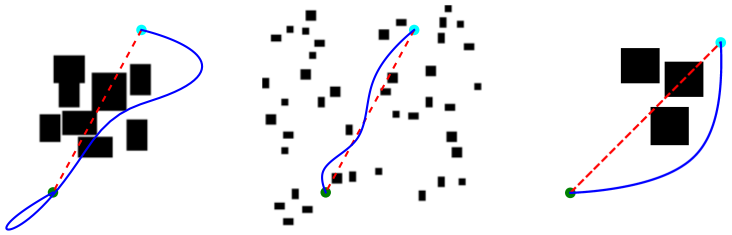
outperforms several leading sampling and optimization based approaches to motion planning [336], it still has some practical limitations with respect to setting the parameters in its objective in Eq. 3.6. Next, we will discuss these limitations in-depth with a few examples.

3.2.2 Sensitivity to objective function parameters

The performance of GPMP2 is dependent on the values of $\mathbf{Q}_C$ (the parameter that governs the covariance of the GP prior) and $\mathbf{\Sigma}$ (the covariance of the likelihood) as per its objective function from Eq. 3.6. For example, for collision avoidance, the distribution of obstacles in the environment affects what relative settings of $\mathbf{Q}_C$ and obstacle covariance σ_{obs} (such that $\mathbf{\Sigma} = \sigma_{obs}^2 \times \mathbf{I}$) will be effective in solving the planning problem.

Different datasets require different relative settings of parameters. Due to the nonlinear interactions between these parameters it might not be possible to find a fixed setting that will always work, and in practice it can be a tedious task to find a setting that works for many different environments. For example, in environments like the one in Fig. 3.2(a)-3.2(b), where the planner needs to find a trajectory that goes around the cluster of obstacles, a small obstacle covariance is required to make the planner navigate around the "tarpit." But, at the same time, if a large dynamics covariance is used, it might try to squeeze in between obstacles where the cost can have a local minima. So a smaller dynamics covariance is needed as well. Another example is shown in Fig. 3.2(c)-3.2(d) with dispersed obstacles near the start and goal. Here an entirely different setting of covariances is effective. Since obstacles are small and diffused, solutions can generally be found close to the straight line initialization. A smaller dynamics covariance helps with that. Also, the start and goal can be very near obstacles which means that a small obstacle covariance might lead to solutions that violate the start and goal constraints. Having a smaller obstacle covariance can also lead to trajectories that are very long and convoluted as they try to stay far away from obstacles.

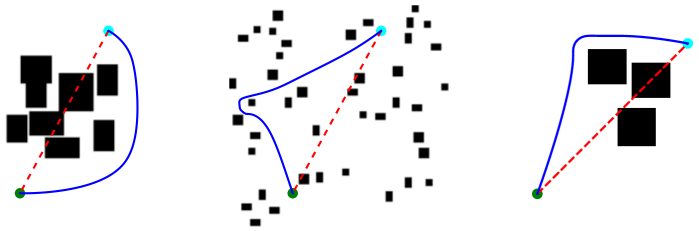
Small changes in parameters can lead to trajectories lying in different homotopy classes. For example, Fig. 3.2(e)-3.2(f) illustrates how even minor changes in the obstacle covariance can lead to significant changes in the resulting trajectories. This makes tuning covariances



(a) $\sigma_{obs} = 0.1$, $\boldsymbol{Q}_C = 0.5 \times I$

(c) $\sigma_{obs} = 0.1$, $\boldsymbol{Q}_C = 0.5 \times I$

(e) $\sigma_{obs} = 0.03$, $\boldsymbol{Q}_C = I$



(b) $\sigma_{obs} = 0.01$, $\boldsymbol{Q}_C = 0.5 \times I$

(d) $\sigma_{obs} = 0.01$, $\boldsymbol{Q}_C = 0.5 \times I$

(f) $\sigma_{obs} = 0.01$, $\boldsymbol{Q}_C = I$

Figure 3.2: (a)-(b) `tarpit` dataset For the same $\boldsymbol{Q}_C$, a smaller σ_{obs} is required to encourage the planner to navigate around obstacles. (c)-(d) `forest` dataset For the same $\boldsymbol{Q}_C$, a larger σ_{obs} is required to focus on finding solutions near the straight line trajectory. (e)-(f) `multi_obs` dataset A small change in obstacle covariance can lead to significant changes in the trajectory. In all figures, the red dashed trajectories are the initializations and the blue trajectories are the optimized solutions.

harder, as the effects are further aggravated over large datasets with diverse environments leading to inconsistent results.

With sufficient domain expertise, the parameters can be hand-tuned. However, this process can be very inefficient and becomes increasingly hard for problems in higher dimensions or when complex constraints are involved. An ideal setup would be to have an algorithm that can predict appropriate parameters automatically for each problem. Therefore, in this work, we rebuild the GPMP2 algorithm as a fully differentiable computational graph, such that these parameters can be specified by deep neural networks which can be trained end-to-end from data. When deployed, our differentiable GPMP2 approach (dGPMP2) can then

automatically select its own parameters given a particular motion planning problem.

3.3 A Structured Computational Graph for Motion Planning

In this section, we first explain how GPMP2 can be interpreted as a differentiable computation graph. Then, we explain how learning can be incorporated in the framework and finally, we show how the entire system can be trained end-to-end from data.

3.3.1 Differentiable GPMP2

Our architecture consists of two main components: a planning module **P** that is differentiable but has no learnable parameters and a trainable module **W** that can be implemented using a differentiable function approximator such as a neural network as shown in Fig. 3.1. As discussed in Section 3.2, GPMP2 performs trajectory optimization via MAP inference on a factor graph by solving an iterative nonlinear optimization, where at any iteration the factor graph is linearized at the current estimate of the trajectory to produce the linear system in Eq. equation 3.7 and an update step is computed by solving that linear system. At a high level, our planning module **P** implements this update step as a computational graph. The trainable module **W** is then set up to parameterize some desired planning parameters and outputs these as $\boldsymbol{\phi}_L$ at every iteration. These parameters correspond to factor covariances used by **P** to construct the linearized factor graph. Additionally, **P** takes as input a set of fixed planning parameters $\boldsymbol{\phi}_F$ to allow parameters that can be user-specified and are not being learned, for example, obstacle safety distance and covariances of constraint factors like start, goal, and velocity. The key insight is that since all operations are differentiable for solving Eq. 3.7, we can easily differentiate through it using standard autograd tools [366] and thus train **W** in an end-to-end fashion from data.

Similar to GPMP2, during the forward pass, dGPMP2 iteratively optimizes the trajectory where at the i^{th} iteration, the planning module **P** takes the current estimate of the trajectory $\boldsymbol{\theta}^i$ and planning parameters $\boldsymbol{\phi}_L$ and $\boldsymbol{\phi}_F$ as inputs (where $\boldsymbol{\phi}_L$ is the output of the trainable module **W** and ϕ_F are user-defined and fixed) and produces the next estimate $\boldsymbol{\theta}^{i+1}$ as shown in Fig. 3.1. The new estimate then becomes the input for the next iteration. This process continues until $\boldsymbol{\theta}^{i+1}$ passes a specified convergence check or a maximum of T

iterations and the optimization terminates (**C**). At the end of the optimization, we roll out a complete differentiable computation graph for the motion planner.

Notation: $\boldsymbol{\theta}^i$ refers to the trajectory estimate at the i^{th} iteration of the optimization that goes from $1, \ldots, T$ and $\boldsymbol{\theta}_i$ is the i^{th} state along the trajectory that goes from $1, \ldots, N$.

The planning module: $\boldsymbol{\theta}^i$ is fed into the planning module along with a signed distance field of the environment and additional planning parameters ($\boldsymbol{\phi}_F$ and $\boldsymbol{\phi}_L$) such as factor covariances, safety distance, robot kinematics, start-goal constraints, and other task related constraints. These inputs are used to construct the linear system in Eq. equation 3.7 corresponding to the linearized factor graph of the planning problem. Similar to standard GPMP2, constraints are implemented as factors with fixed small covariances and the likelihood function for obstacle avoidance is the hinge loss function (see Section 6.5) with covariance $\boldsymbol{\Sigma}$. The trajectory update $\delta\boldsymbol{\theta}^i$ is then computed by solving this linear system, using Cholesky decomposition of the normal equations [119, 336], and the new trajectory $\boldsymbol{\theta}^{i+1}$ is computed using a Gauss-Newton step. The above procedure is fully differentiable and allows computing gradients in the backwards pass with respect to $\boldsymbol{\theta}^i$, GP covariance $\boldsymbol{\mathcal{K}}$, and likelihood function covariance $\boldsymbol{\Sigma}$.

The trainable module: The trainable module $\mathbf{W}$ outputs planning parameters $\boldsymbol{\phi}_L$. These correspond to covariances of factors in Eq. 3.7 that we wish to learn from data. In practice, we can choose to learn the GP covariance $\boldsymbol{\mathcal{K}}$, the likelihood covariance $\boldsymbol{\Sigma}$, or both. Additionally, this approach allows us to learn individual covariances for different states along the trajectory $[\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N]$ and different iterations of the optimization thus offering much more expressiveness than a single hand-tuned covariance. We implement $\mathbf{W}$ as a feed-forward convolutional neural network that takes as input the bitmap image of the environment, the signed distance field and the current trajectory $\boldsymbol{\theta}^i$, and outputs a parameter vector $\boldsymbol{\phi}_L^i$ at every iteration i. Note that, given our architecture, $\mathbf{W}$ can be customized as per individual needs based on problem requirements or parameters chosen to be learned.

After a forward pass, we roll out a fully differentiable computation graph that outputs a sequence of trajectories $\{\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^T\}$. Then we evaluate a loss function on this sequence and backpropagate that loss to update the parameters of $\mathbf{W}$ such that it produces parameters $\boldsymbol{\phi}_L$ that allow us to optimize for better quality trajectories on the dataset as measured by

the loss. We explain our loss function and the training procedure in detail below.

3.3.2 Learning factor graph covariances

Imitation loss: Consider the availability of expert demonstrations for a planning problem. These may be provided by an asymptotically optimal (but slow) motion planner [237] or by human demonstration [389]. dGPMP2 can be trained to produce similar trajectories by minimizing an error metric between the demonstrations and learner's output with

$$\mathcal{L}_{imitation} = ||\boldsymbol{\theta}^e - \boldsymbol{\theta}||_2^2 \tag{3.9}$$

where $\boldsymbol{\theta}^e$ is the expert's demonstrated trajectory and the metric is the L2 norm.

Task loss: Naively trying to match the expert can be problematic for a motion planner. For example, when equally good paths lie in different homotopy classes, the learner may land in a different one than the expert. In this case, penalizing for not matching the expert may be excessively conservative. If using human demonstrations as an expert, a realizability gap can arise when the planner has different constraints as compared with the human. Thus, we use an external task loss as a regularizer that encourages smoothness and obstacle avoidance, while respecting start and goal constraints, as is often used in motion planning [543]:

$$\mathcal{L}_{plan} = \mathcal{F}_{smooth} + \lambda \times \mathcal{F}_{obs}, \tag{3.10}$$

where $\mathcal{F}_{smooth}$ corresponds to the GP prior error and $\mathcal{F}_{obs}$ is the obstacle cost that are described in Eq. equation 3.6 and λ is a user specified parameter. In practice, the performance is not sensitive to the setting of λ. Then, the overall loss for a single trajectory is, $\mathcal{L} = \mathcal{L}_{imitation} + \mathcal{L}_{plan}$. Note that our framework allows for any choice of loss function depending on the application.

Training: During training we roll out our learner for a fixed number of iterations T and use Backpropagation Through Time (BPTT) [406] on the sum of losses of the intermediate trajectories in order to update the parameters of the trainable module $\mathbf{W}$. Then, the total loss minimized for our learner over a batch of size K is

$$\mathcal{L}_{total} = \frac{1}{K}\frac{1}{T}\sum_{k=1}^{K}\sum_{i=1}^{T}\mathcal{L}^{k,i}. \tag{3.11}$$

3.4 Experiments

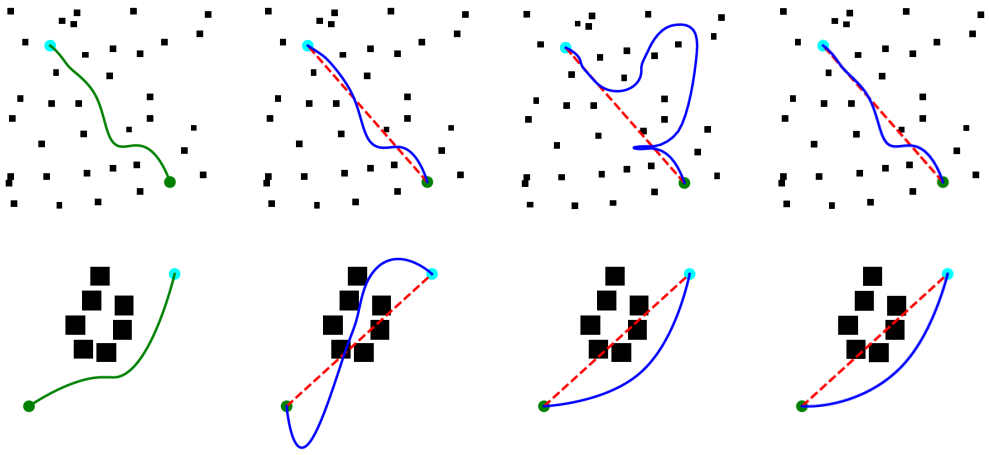
We test our approach on 2D navigation problems with complex environment distributions and problems with user-specified velocity constraints. Many real world motion planning problems such as warehouse automation (KIVA systems), extra-terrestrial rovers, in-home robots (Roomba), navigation from satellite data, and last mile delivery, among others, are inherently 2D. These problems are challenging partly because of local minima generated by complex distributions of obstacles and other constraints such as velocity limits. The sensitivity of planners to parameter settings further adds to the difficulty. This is captured by the datasets we consider, where the `forest` distribution consists of small obstacles scattered around the workspace and requires the robot to squeeze through several narrow corridors and the `tarpit` distribution contains a small number of larger obstacles clumped together near the center of the workspace and requires the robot to avoid the cluster of obstacles entirely. It is challenging for a single planner with fixed parameters to solve problems from both distributions.

3.4.1 Implementation details

All our experiments and training are performed on a desktop with 8 Intel Core i7-7700K @ 4.20GHz CPUs, 32GB RAM and a 12GB NVIDIA Titan Xp. We consider a 2D point robot in a cluttered environment and planning is done in a state space $\boldsymbol{\theta}_i = [x, y, \dot{x}, \dot{y}]^T$. The robot is represented as a circle with radius r centered on its center of mass and the environment is a binary occupancy grid. A Euclidean signed distance field is computed from the occupancy grid to evaluate distance to obstacles and check collisions. We utilize the same collision likelihood factor as GPMP2 [131], $\boldsymbol{h}(\boldsymbol{\theta}_i) = c(\boldsymbol{x}(\boldsymbol{\theta}_i))$, where $\boldsymbol{x}(\boldsymbol{\theta}_i) = [x, y]^T$ is the position coordinates of the center of mass and the hinge loss cost function c is

$$c(\boldsymbol{x}) = \begin{cases} -d(\boldsymbol{x}) + \epsilon & d(\boldsymbol{x}) \leq \epsilon \\ 0 & otherwise \end{cases} \tag{3.12}$$

where $\epsilon = r + \epsilon_{safe}$ with ϵ_{safe} as a user defined safety distance, and d is the signed distance. In our current experiments, we consider σ_{obs} as the learned parameter $\boldsymbol{\phi}_L$ and $\mathbf{Q}_C, \epsilon_{safe}, \boldsymbol{\mathcal{K}}_s, \boldsymbol{\mathcal{K}}_g$



(a) Expert (b) GPMP2, $\sigma_{obs} = 0.15$ (c) GPMP2, $\sigma_{obs} = 0.01$ (d) dGPMP2

Figure 3.3: Example comparison of (d) dGPMP2 against (b)-(c) GPMP2 (fixed hand tuned covariances) and (a) Expert on `forest` (top row) and `tarpit` (bottom row) datasets. Hand tuned covariances that work well on one distribution of obstacles fail on the other and vice versa. By imitating the expert, dGPMP2 is able to perform consistently across different environment distributions. Green circle is start, cyan is goal, dashed red line is initialization, and $\boldsymbol{Q}_c = 0.5 \times I$, $r = 0.4m$ for all. Trajectory is in collision if at any state the signed distance between robot center of mass and nearest obstacle is less than or equal to r.

to be the fixed parameters $\boldsymbol{\phi}_F$. Although, performance of the planner depends on both $\mathbf{Q}_C$ and $\boldsymbol{\Sigma}$, for our task they trade off against each other and thus we can achieve a similar behavior by varying one relative to the other. Since in our setup the environment changes, learning the likelihood covariance $\boldsymbol{\Sigma}$ is more relevant. In other problem domains learning $\mathbf{Q}_C$ instead might be more relevant such as [389]. It is important to note the difference in expressiveness of $\boldsymbol{\Sigma}$ between GPMP2 where $\boldsymbol{\Sigma} = \sigma_{obs}^2 \times \mathbf{I}$, and dGPMP2 where $\boldsymbol{\Sigma} = \text{diag}(\sigma_{obs_1}^2, \ldots, \sigma_{obs_N}^2)$ with any σ_{obs_i} being a function of the current trajectory and the environment.

Loss function: It can be expensive to gather a large number of human demonstrations to train the planner. Hence, we use a self-supervised approach. Sampling based asymptotically optimal planning methods such as RRT* [237] are effective in finding good homotopy classes to serve as an initialization for local trajectory optimizers, but can be slow to converge

and produce non-smooth solution paths. We use a combination of RRT* and GPMP2 as our expert. Expert trajectories are generated by first running RRT* and are then optimized with GPMP2 to yield smooth solutions. This allows dGPMP2 to learn by utilizing the best combination of local and global planning. We use the loss function defined in Section 3.3.2 with this expert.

Network architecture: For $\mathbf{W}$ we use a standard feed-forward neural network model consisting of convolutional and fully connected layers. The network consists of 5 convolutional layers with [16, 16, 16, 32, 32] filters respectively, all 3x3 in size. This is followed by two fully connected layers with [1000, 640] hidden units. We use ReLU activation with batch normalization in all layers and a dropout probability of 0.5 in the fully connected layers. The input to the neural network is a 128x128 bitmap of the environment stacked on top of the euclidean signed distance field of the same dimensions. Backpropagation is performed for fixed number of iterations, $T = 10$. At every iteration, the network outputs a different likelihood covariance for each state along the trajectory.

Comparing planners: The convergence for the optimization is based on the following criterion: a tolerance on the relative change in error across iterations $\texttt{tol}(\delta_{error})$, magnitude of update $\texttt{tol}(\delta\boldsymbol{\theta})$, and max iterations T_{max}. On convergence the final trajectory is returned. We report the following metrics on a test set of environments: (i) `success`, percent of problems solved i.e. when a collision free trajectory is found, (ii) average `gp_mse`, mean-squared GP error measuring smoothness and (iii) `collision_intensity`, the average portion of trajectory spent in collision when a collision occurs.

We test our framework on two different planning tasks to demonstrate (i) how learning covariances improves performance and (ii) how the planner's structure allows us to incorporate constraints. We compare against a baseline GPMP2 with hand-tuned parameters. However, we do not compare against other sampling and optimization-based planners and refer the reader to [336] for benchmarks of GPMP2 against leading sampling and optimization-based planners.

Table 3.1: Comparison of dGPMP2 versus GPMP2 with fixed hand tuned covariances. dGPMP2 learns the obstacle covariance σ_{obs} using training set of 5000 environments. $\boldsymbol{Q}_C = 0.5 \times I$ for all. Total trajectory time is 10s with 100 states along the trajectory and $\lambda = 1.0$ for training.

		GPMP2		dGPMP2
		$\sigma_{obs} = 0.15$	$\sigma_{obs} = 0.01$	
`forest` only	`success`	71.02	52.18	66.67
`tarpit` only		55.56	74.08	68.00
`mixed`		62.67	64.00	67.33
`gp_mse`		0.002	0.0484	0.0015
`num_iters`		55.69	86.74	50.00
`coll_intensity`		0.0464	0.0414	0.0374

3.4.2 Learning on complex distributions

In this experiment, we show that if the planner's parameters are fixed, performance can be highly sensitive to distribution of obstacles in the environment. However, if a function can be learned to set the parameters based on the current planning problem, this can help the planner achieve uniformly good performance across different obstacle distributions. We construct a hybrid dataset which is a mixture of two distinct distributions of obstacles, `forest` and `tarpit`, as shown in Fig. 3.3. We use a test set of 150 randomly sampled environments from this mixed dataset and further subdivide it into two sets for each of the constituent distributions (roughly equal in proportion). We then hand-tuned parameters for GPMP2 to find the best covariances for the individual distributions and compared them against dGPMP2 on three different test sets: two for the individual distributions and one for a mixed (roughly equal of the two distributions). Since there is no formal mathematical procedure for tuning parameters or even well-known heuristics, we rely on a manual line-search. Although this can likely be automated to find best static covariances for one given environment distribution, it is not practical when the environment distribution changes or when the parameters need to be adapted based on the location of the robot in the

environment or the time-step on the trajectory.

The results in Table 3.1 show that for GPMP2 the best parameters on one distribution perform poorly on the other distribution in terms of success, although their performances on the mixed dataset are similar. Conversely, dGPMP2 has uniform and consistent performance across both distributions even though it is only trained on the mixed dataset. This demonstrates that dGPMP2 does not require manual tuning for every distribution of planning environments, but can automatically predict the covariances to use based on the current trajectory and environment as can be seen in Fig. 3.3. Additionally, dGPMP2 has the lowest `gp_mse` on the mixed dataset meaning the trajectories produced are still smooth. dGPMP2 also converges in fewer number of iterations than the GPMP2 due to the covariance being more expressive and varying over iterations.

Limitations: Since BPTT is known to have issues with exploding and vanishing gradients for long sequences, we use a small number of iterations ($T = 10$) during training which prevents the learner from sufficiently exploring during training. The network architecture is a simple feed-forward network and does not have any memory and hence the learner does not learn to escape local minima very well. We believe that these issues can be addressed in the future using learning techniques such as Truncated Backpropagation Through Time [507], policy gradient methods [416, 418], and recurrent networks like LSTMs [195].

3.4.3 Planning with velocity constraints

We show that our learning method can explicitly incorporate planning constraints by including velocity limit factors into the optimization. We use a hinge loss similar to obstacle cost to bound the robot velocity $\dot{x}$ and $\dot{y}$ and set the covariance to a low value, $\boldsymbol{\mathcal{K}}_v = 10^{-4}$, analogous to joint limit factors in [336]. We evaluate the average `constraint_violation` on a dataset with multiple randomly placed obstacles and study the effect of incorporating constraints during training. Table 3.2 shows a comparison between dGPMP2 trained with mild constraints and tested on problems with mild and tight constraints versus dGPMP2 trained using tight constraints and tested on problems with tight constraints (details in the Table 3.2 caption). We see that, by incorporating tight constraints during training, dGPMP2

Table 3.2: Performance of dGPMP2 with velocity constraints on different combinations of training and testing. Mild constraints are $v_{xmax} = 1.5m/s$, $v_{ymax} = 1.5m/s$, and $maxtime = 15s$, tight constraints are $v_{xmax} = 1.0m/s$, $v_{ymax} = 1.0m/s$, and $maxtime = 10s$ for the same start and goal. $maxtime$ is maximum time allowed for the trajectory.

Training condition	Mild	Mild	Tight
Testing condition	Mild	Tight	Tight
`success`	96	96	98.12
`constraint_violation`	0.0022	0.104	0.097

can learn to handle tight constraints while avoiding obstacles. This illustrates that dGPMP2 can successfully incorporate constraints within its structure, and that the method can learn to plan while respecting user-defined planning constraints.

3.5 Related Work

Machine learning has been used to accelerate motion planning by combining reinforcement learning (RL) with sampling based planning [147], learning cost functions from demonstration [395], learning efficient heuristics [43], and learning collision checking policies [42]. As machine learning becomes more accessible, there has been a growing interest in using deep learning for planning such as end-to-end networks to perform value iteration [465] or learning a latent space embedding and a dynamics model that is suitable for planning by gradient descent within a goal directed policy [444]. Such approaches have demonstrated that *learning to plan* is a promising research direction as it allows the agent to explicitly reason about future actions. However, learning-based approaches still fall short on several fronts. Combining learning and planning in a way where domain knowledge, constraints, and uncertainty are properly handled is challenging, and learned representations are often difficult to interpret.

Recent work in structured learning techniques offer avenues towards contending with these challenges. Several methods have focused on incorporating optimization within neural network architectures. For example, [104] implicitly learns to perform nonlinear least squares

optimization by learning an RNN that predicts its update steps, [24] learns to perform gradient descent, and [77] utilizes a ODE solver within its network. Other methods like [23] learn a sequential quadratic program as a layer in its network, which was later extended to solve model predictive control [22]. [62] learns structured dynamics models for reactive visuomotor control. Taking inspiration from this body of work, in this paper we present a differentiable inference-based motion planning technique that through its structure allows us to combine the strengths of both traditional model-based methods and modern learning methods, while mitigating their respective weaknesses.

Another related field of work is on automatic parameter tuning of motion planning algorithms. Approaches such as [60, 66] treat the planner as a black-box and use machine learning tools such as Bayesian optimization, bandits, and random forests to optimize a single configuration of parameters that improves planner performance. However, such a single configuration does not adapt to changes in the environment distribution which hinders generalization. Additionally, the number of parameters optimized in these approaches is far fewer than ours and they do not incorporate high dimensional inputs such as images.

3.6 Discussion

This chapter presented dGPMP2, a novel differentiable motion planning algorithm, by reformulating GPMP2 as a differentiable computational graph. We developed a method that learns to predict objective function parameters as part of the differentiable planner and demonstrates competitive performance against planning with fixed, hand-tuned parameters. The experimental results show that this strategy is indeed an effective way to leverage experience to further improve upon traditional state-of-the-art motion planning algorithms. In order to investigate the properties of the algorithm in a controlled setting, the experiments in this work were limited to point robots in 2D environments . However, since the formulation was built on the GPMP2 planner, we believe that it can be extended to handle more complicated motion planning problems including articulated robots in 3D workspaces.

Chapter 4

FAST JOINT-SPACE MODEL-PREDICTIVE CONTROL FOR REACTIVE MANIPULATION

4.1 Introduction

In the previous chapters, we considered robot motion planning under the assumption of full model knowledge. However, this assumption can be highly restrictive in several real-world settings where the complexity of dynamic environments makes it challenging to model them exactly. Often the best we can achieve is the development of approximate predictive models. Furthermore, most implementations of motion planning algorithms run at a slow rates on high dimensional systems [21, 238, 267, 480], making them unsuitable for dynamic scenarios where robots must constantly update their motion plans in response to sensory input. In this chapter, we show how the paradigm of sampling-based MPC can enable robust decision-making by optimizing complex behaviors with approximate models at fast control rates. We detail the design of an efficient sampling-based MPC framework that implements a highly parallelized control architecture and enables real-world manipulators to perform a variety of dynamic tasks. In the following chapter, we take this a step further and demonstrate how MPC can be improved from experience through model-free RL to systematically overcome different sources of bias resulting from short planning horizons and approximate models.

Real-world robot manipulation can greatly benefit from real-time perception-driven feedback control [49, 235]. Consider an industrial robot tasked with stacking boxes from one pallet to another or a robot bartender moving drinks placed on a tray [313]. In both cases, the robot must ensure object stability via perception while respecting joint limits, avoiding singular configurations and collisions, and handling task constraints such as maintaining orientation during transfer. This leads to a complex control problem with competing objectives that is difficult to solve in real-time.

Classic approaches to solving these tasks rely on operational-space control (OSC) [84, 128, 345], where the different task costs are formulated in their respective spaces and then projected into the joint space (i.e., the control space of the robot) via a Jacobian map. OSC methods are inherently local as they only optimize for the next time step while ignoring future actions or states.

MPC based approaches attempt to find a locally-optimal policy over a finite horizon starting from the current state using a potentially imperfect dynamics model. An action from the policy is executed on the system and the optimization is performed again from the resulting next state which can overcome the effects of model-bias. MPC has been successfully applied on real robotic systems allowing them to rapidly react to changes in the environment [35, 84, 141, 423, 505]. Existing MPC methods that operate in the joint space of a manipulator are limited to gradient-based approaches [468, 504] which require the cost and dynamics to be differentiable. However, manipulation tasks often involve discontinuous phenomena such as contact and complex cost terms that are hard to differentiate analytically. Sampling-based methods such as Model-Predictive Path Integral (MPPI) [506] and Cross Entropy-Method (CEM) offer a promising alternative. Here, control sequences are sampled using a simple policy followed by rolling out the dynamics model to compute a sample-based gradient of the objective function. These algorithms make no restrictive assumptions about the cost, dynamics or policy class, are straightforward to parallelize and can be effectively applied on highly dynamic systems [493, 505]. These properties have also been a major factor in the increased adoption of sampling-based MPC by the Model-based RL community in recent years [103]

However, a key question still remains to be answered - *how well do these control algorithms transfer to high-dimensional robots like manipulators?* In this chapter, we describe an integrated system for sampling-based MPC that aims to answer this question. Our proposed framework, Stochastic Tensor Optimization for Robot Motion (STORM) implements a highly-parallelized control architecture that can optimize complex task objectives while simultaneously ensuring desirable properties such as smoothness, constraint satisfaction, and low control latency.

A major criticism of sampling-based MPC algorithms for full joint space control has been

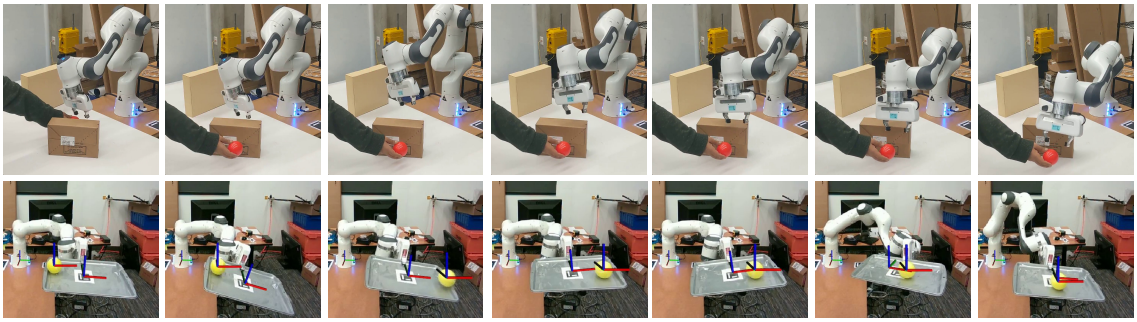


Figure 4.1: Our sampling-based model-predictive control framework operates in the joint space to enable a robot to achieve manipulation objectives such as tracking a moving target or balancing a ball on a plate while respecting constraints such as joint limits, singularity avoidance and collision avoidance via learned collision checking from raw sensor data.

their inability to produce smooth (low-jerk) trajectories [344]. To address this challenge, we study different sub-components of sampling-based MPC and make several novel contributions such as low discrepancy action sampling, smooth interpolation and cost-function design, and demonstrate their effectiveness in scaling sampling-based methods to real robot manipulators. We also demonstrate that STORM can incorporate learned components in the control loop, by using learned self and environment collision costs. We integrate our system on a real Franka Panda robot arm where we demonstrate that feedback driven sampling-based MPC is able to solve complex and dynamic manipulation tasks with simple models.

4.1.1 Contributions

1. We present a novel sampling-based MPC with the introduction of low discrepancy sampling, smooth trajectory generation and behavior-based cost functions that are key for producing smooth, reactive, and precise robot motions.

2. We also present a feedback control framework that directly integrates learned perception components into the control loop in the form of a learned self collision cost and a discrete collision checker between the robot links and raw environment pointcloud

from [112].

3. We provide an open-source and highly optimized implementation of sampling-based joint-space MPC, which achieves a control rate of 125Hz on a single GPU, a speedup of 100x compared to existing MPPI based manipulation implementations [112].

4. We empirically evaluate the key components of the approach in simulation and a real-world Franka Panda robot on dynamic control tasks.

4.2 Problem Formulation

We consider the problem of generating a feedback control law (or policy) for a robot manipulator with d joints performing user-specified tasks in an unstructured environment where it is subject to non-linear constraints and must react in real-time to overcome errors due to its internal dynamics model, state estimation, and perception. The problem can be modelled as optimal control of a discrete-time stochastic dynamical system described by the equation, $x_{t+1} \sim P(x_{t+1}|x_t, u_t)$, where $P(x_{t+1}|x_t, u_t)$ defines the probability of transitioning into state x_{t+1} conditioned on x_t and control input $u_t \in \mathbb{R}^d$. The robot chooses controls using a deterministic or stochastic closed-loop policy $u_t \sim \pi(\cdot|x_t)$, incurs an instantaneous cost $c(x_t, u_t)$ and transitions into the next state, and the process continues for a finite horizon T. The goal is to design a policy that optimizes a task-specific objective function, $\pi^* = \arg\min_{\pi \in \Pi} \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{T-1} c(x_t, u_t)\right]$ where Π is the space of all policies.

The above setup is akin to optimizing a finite horizon Markov Decision Process (MDP) with continuous state and action spaces as done in reinforcement learning approaches [540]. Solving for a complex globally optimal policy is a hard problem especially since the task objective c could be sparse or difficult to optimize. MPC can be viewed as a practically motivated strategy that simplifies the overall problem by focusing only on the states that the robot encounters *online* during execution and rapidly re-calculating a "simple" locally optimal policy. At state x_t, MPC performs a look-ahead for a shorter horizon $H < T$ using an approximate model $\hat{P}(\hat{x}_{t+1}|\hat{x}_t, a_t)$, approximate cost function $\hat{c}(x_t, a_t)$ and a parameterized policy π_ϕ to find the optimal parameters ϕ^* that minimize an objective function L

Variable	Description
$\theta_t, \dot{\theta}_t, \ddot{\theta}_t$	joint position, velocity, acceleration
$x_t = [\theta_t, \dot{\theta}_t, \ddot{\theta}_t]$	robot state at time t
u_t	joint space command at time t
$h \in [0, H)$	number of steps in horizon H
$n \in [0, N)$	batch of control sequences
K	iterations of optimization
dt	change in time between steps in horizon
$\mathbf{dt}$	vector of dt $[0, H)$
$\phi_t = \{\phi_{t,h}\}$	parameters of policy at time t.
π_{ϕ_t}	distribution over control given state
$\mu_t = \mu_{t,h}$	sequence of H Gaussian means
$\mathbf{\Sigma}_t = \Sigma_{t,h}$	sequence of H Gaussian Covariances
$\mathbf{u} = \mathbf{u}_{n,h}$	batch of N control sequences of length H
$\hat{\mathbf{x}} = \hat{\mathbf{x}}_{n,h}$	batch of N state sequences of length H
$\hat{\mathbf{c}} = \hat{\mathbf{c}}_{n,h}$	batch of N cost sequences of length H
L	MPC loss function

Algorithm 5: Sampling-Based MPC

Input : θ_0,

Parameter: H, N, K

for $t = 1 \dots T$ **do**
- $x_t \leftarrow$ GET_STATE()
- $\pi_\theta \leftarrow$ SHIFT()
- **for** $i = 1 \dots K$ **do**
  - $\mathbf{u} \leftarrow$ SAMPLE_CONTROLS(π_{ϕ_t}, H, N)
  - $\hat{\mathbf{x}}, \hat{\mathbf{c}}, \hat{\mathbf{q}} \leftarrow$ GENERATE_ROLLOUTS(x_t, H)
  - $\phi_t \leftarrow$ UPDATE_DISTRIBUTION$(\hat{\mathbf{c}}, \mathbf{u})$
- $u_t =$ NEXT_COMMAND(π_ϕ)
- EXECUTE_COMMAND(u_t)

Figure 4.2: We summarize the notations used on the left and the sampling based MPC algorithm on the right.

$$\phi^* = \arg\min_{\phi} L(\hat{c}, \hat{q}, \pi_\phi, \hat{P}, x_t) \tag{4.1}$$

where $\hat{q}(\cdot)$ is a terminal cost function that serves as a coarse approximation of the cost-to-go beyond the horizon. An action is sampled from π_{ϕ^*} and the optimization is performed again from the resulting state after applying the action to the robot. The optimization is hot-started from the solution at the previous timestep by using a *shift* operator, which allows MPC to produce good solutions with few iterations of optimization (usually 1 in practice).

In the next section, we first introduce a sampling-based MPC technique to solving the optimization in Eq. 4.1 and discuss the objective function, policy class, and update equations. We then present our approach for applying it to manipulation problems. An overview of the notation used in the paper is presented in Fig. 4.2.

4.3 Sampling-Based Model Predictive Control

Sampling-based MPC iteratively optimizes simple policy representations like time independant Gaussians over open-loop controls with parameters ϕ_t such that $\pi_{\phi_t} = \prod_{h=0}^{H-1} \pi_{\phi_{t,h}}$. Here, ϕ_t represent the sequence of means $\mu_t = [\mu_{t,0} \dots \mu_{t,H-1}]$ and covariances $\mathbf{\Sigma}_t = [\Sigma_{t,0} \dots \Sigma_{t,H-1}]$ at every step along the horizon H. A standard algorithm is shown in Fig. 4.2. At every iteration, the optimization proceeds by sampling a batch of N control sequences of length H, $\mathbf{u}_{n\in[0,N),h\in[0,H)}$, from the current distribution (Line 5), followed by rolling out the approximate dynamics function using the sampled controls to get a batch of corresponding states $\hat{\mathbf{x}}_{n\in[0,N),h\in[0,H)}$ and costs $\hat{\mathbf{c}}_{n\in[0,N),h\in[0,H)}$ (Line 6). The policy parameters are then updated using a sample-based gradient of the objective function (Line 7). After $K \geq 1$ optimization iterations we can either sample an action from the resulting distribution or execute the first action from the mean (Line 8). We next describe how the distribution is updated. Consider the function $\hat{C}(\cdot)$,

$$\hat{C}(x_t, u_t) = \sum_{h=0}^{H-2} \gamma^h \hat{c}(\hat{x}_{t,h}, u_{t,h}) + \gamma^{H-1} \hat{q}(\hat{x}_{t,H-1}, a_{t,H-1}). \tag{4.2}$$

where $\gamma \in [0, 1]$ is a discount factor that is used to favor immediate rewards. A widely used objective function is the exponentiated utility or the risk-seeking objective,

$$L = \mathbb{E}_{\pi_\theta, \hat{P}} \left[\exp\left(\frac{-1}{\beta} \hat{C}(x_t, u_t) \right) \middle| \hat{x}_0 = x_t \right] \tag{4.3}$$

where β is a temperature parameter. For this choice of objective, the mean and covariance are updated using a sample-based gradient as,

$$\mu_{t,h} = (1 - \alpha_\mu)\mu_{t-1,h} + \alpha_\mu \frac{\sum_{i=1}^N w_i u_{t,h}}{\sum_{i=1}^N w_i} \tag{4.4}$$

$$\Sigma_{t,h} = (1 - \alpha_\sigma)\Sigma_{t-1,h} + \alpha_\sigma \frac{\sum_{i=1}^N w_i (u_{t,h} - \mu_{t,h})(u_{t,h} - \mu_{t,h})^\top}{\sum_{i=1}^N w_i} \tag{4.5}$$

where α_μ and α_σ are step-sizes that regularize the current solution to be close to the previous one and,

$$w_i = \exp \frac{-1}{\beta} \left(\sum_{h=0}^{H-2} \gamma^h \hat{c}(\hat{x}_{h,i}, a_{h,i}) + \gamma^{H-1} \hat{q}(\hat{x}_{H-1,i}, a_{H-1,i}) \right). \tag{4.6}$$

The update equation for the mean is the same as the well-known Model-Predictive Path Integral Control (MPPI) algorithm [506]. We refer the reader to [494] for the connection and derivation of update equations. While covariance adaptation has previously been explored in the context of Path Integral Policy Improvement [446] to automatically adjust exploration, standard implementations of MPPI on real systems generally do not update the covariance [494, 506]. However, we observed that updating the covariance leads to better performance with a fewer number of particles, such as stable behavior upon convergence to the goal. Once an action from the updated distribution is executed on the system, the mean and covariance are shifted forward with default values appended at the end to warmstart the optimization at the next timestep (Line 3).

The above formulation of MPC offers the flexibility to extract different behaviors from our algorithm by tuning different knobs such as the choice of approximate dynamics, running cost, terminal cost, the policy class and parameters such as the horizon length, number of particles, step sizes and discount factor γ. Next, we switch our focus to the domain of robot manipulation and build our approach for sampling-based MPC by systematically evaluating the effects of a subset of key design choices on the overall performance of the controller.

4.3.1 Approximate Model

The MPC paradigm allows us to effectively leverage simple models that are both computationally cheap to query and easy to optimize, as re-optimizing the control policy at every timestep can help to overcome the effects of errors in the approximate model. We leverage this error correcting property of MPC and utilize the kinematic model of the manipulator as our approximate transition model. Let the robot state be defined in the joint space as $x = [\theta, \dot{\theta}, \ddot{\theta}] \in \mathbb{R}^{3d}$ and the commanded action be the joint acceleration $u \in \mathbb{R}^d$. At every optimization iteration, we compute the state of the robot across the horizon ($\hat{\mathbf{x}} = [\Theta_{n,h}, \dot{\Theta}_{n,h}, \ddot{\Theta}_{n,h}]$, $h \in [0, H)$, $n \in [0, N)$) by integrating the sampled control sequences $\mathbf{u}_{n\in[0,N),h\in[0,H)}$ from the robot's initial state $x_{\text{init}} = [\theta_{\text{init}}, \dot{\theta}_{\text{init}}, \ddot{\theta}_{\text{init}}]$ (i.e., current state of the real robot). This can be efficiently implemented as a tensor product followed by a sum:

$$\ddot{\Theta} = \mathbf{u} \qquad \dot{\Theta} = \dot{\theta}_{\text{init}} + S_l(1)\,\text{diag}(\mathbf{dt})\ddot{\Theta} \qquad \Theta = \theta_{\text{init}} + S_l(1)\,\text{diag}(\mathbf{dt})\dot{\Theta} \tag{4.7}$$

where $S_l(1)$ is a lower triangular matrix filled with 1, and $\mathbf{dt}$ is a vector of delta timesteps across the horizon. We use variable timesteps, with a smaller dt for the earlier steps along the horizon for higher resolution cost near the robot's current state and larger ones for later steps to get a better approximation of cost-to-go, similar to [141]. We intentionally write $\Theta_{n,h}$ as Θ to highlight the fact that we compute the states across the batch and horizon with a tensor operation without the need to iteratively compute the states across the horizon. This significantly speeds up the computation and is key to achieving the 8ms control latency.

Given a batch of states $\hat{\mathbf{x}}$, the Cartesian poses $\mathbf{X} \in \mathbf{SE}(3)$, velocities $\dot{\mathbf{X}}$, and accelerations $\ddot{\mathbf{X}}$ of the end-effector are obtained by using forward kinematics $FK(\Theta)$, the kinematic Jacobian $J(\Theta)$, and its derivative $\dot{J}(\Theta)$ as

$$X = FK(\Theta) \qquad \dot{X} = J(\Theta)\dot{\Theta} \qquad \ddot{X} = \dot{J}(\Theta)\dot{\Theta} + J(\Theta)\ddot{\Theta} \tag{4.8}$$

We can compute the forward kinematics in batch as they depend only on the current state. This provides an easily parallelizable formulation of the robot model that is amenable to GPU acceleration.

4.3.2 Cost Function

The cost function $\hat{c}(s, a)$ encodes high-level robot behavior directly into the MPC optimization. This can be viewed as a form of cost-shaping that allows MPC to achieve sparse task objectives while also satisfying auxillary requirements such as avoiding joint limits, ensuring smooth motions and safety. We consider cost functions that are a weighted sum of simple cost terms, where each individual term encodes a desired robot behavior that can be easily implemented in a batched manner.

Reaching Goal Poses

Given the desired and current Cartesian poses for the end-effector ${}^{w}X_g, {}^{w}X_{ee} \in \mathbb{SE}(3)$ respectively in the world frame w, we compute a task-space cost term that penalizes their distance

$$\hat{c}_{\text{pose}}({}^{w}X_{ee}, {}^{w}X_g) = ||\alpha_1(I - {}^{w}R_g^{\top}\,{}^{w}R_{ee})||_2 + ||\alpha_2({}^{w}R_g^{\top}\,{}^{w}d_{ee} - {}^{w}R_g^{\top}\,{}^{w}d_g)||_2 \tag{4.9}$$

where ${}^{w}R_{ee}, {}^{w}R_{g}$ are the rotation components and ${}^{w}d_{ee}, {}^{w}d_{g}$ are translation components of the current and goal end-effector pose respectively. The weight vectors $\alpha_1, \alpha_2 \in \mathbb{R}^3$ allow us to weigh errors in different directions and orientations with respect to each other and can be set to different values to obtain qualitatively different behavior such as partial pose reaching or enforcing partial pose constraints. For example, we can set a high weight in α_1 along a desired axes to maintain the goal orientation throughout the motion of the robot as we demonstrate in our experiments.

Stopping for Contingencies

The finite horizon makes MPC myopic to events that can occur further in the future. Thus, it is desirable to ensure that the robot can safely stop within the horizon in reaction to events that might be observed at timestep $H-1$, especially in dynamic environments. We encode this behavior by computing a time varying velocity limit $\dot{\theta}_{max} \in \mathbb{R}^H$ for every timestep in the horizon based on a user-specifed maximum acceleration $\ddot{\theta}_{max}$ and the time until $H-1$. This means the joint velocity of the robot must allow it to come to a stop at the end of the horizon by applying the max acceleration. Any state that exceeds this velocity is penalized by a cost which is expressed as

$$\dot{\theta}_{max} = S_u(1)\ddot{\theta}_{max}dt \qquad \hat{c}_{\text{stop}}(\dot{\theta}_t) = \begin{cases} ||\dot{\theta}_{max,t} - |\dot{\theta}|||_2 & \text{if } \dot{\theta}_{max,t} - |\dot{\theta}| > 0.0 \\ 0, & \text{otherwise} \end{cases} \tag{4.10}$$

where $S_u(1)$ is an upper triangular matrix filled with 1.

Joint Limit Avoidance

Given minimum and maximum limits on joint state $\theta_{min}, \theta_{max}$ respectively, we penalize the joint state θ_t only if it exceeds a safety threshold defined by a k_{jl} ratio of its full range.

$$\hat{\theta}_{min} = \theta_{min} + k_{jl}(\theta_{max} - \theta_{min}) \qquad \hat{\theta}_{max} = \theta_{max} - k_{jl}(\theta_{max} - \theta_{min})$$

$$\hat{c}_{\text{joint}}(\theta_t) = \begin{cases} ||\theta_t - \hat{\theta}_{min}||_2 & \text{if } \theta_t < \hat{\theta}_{min} \\ ||\hat{\theta}_{max} - \theta_t||_2 & \text{else if } \theta_t > \hat{\theta}_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

where $\hat{\theta}_{min}$, $\hat{\theta}_{max}$ adds a safety threshold from the actual bounds of the robot. In our experiments, we chose $k_{jl} = 0.1$.

Avoiding Cartesian Local Minima

The manipulability score describes the ability of the end-effector to achieve any arbitrary velocity from a given joint configuration. It measures the volume of the ellipsoid formed by the kinematic Jacobian which collapses to zero at singular configurations. Thus, to encourage the robot to optimize control policies that avoid future kinematic singularities, we employ a cost term that penalizes small manipulability scores [247, 484]

$$\hat{c}_{\text{manip}}(\theta_t) = \begin{cases} 1.0 - \sqrt{J(\theta_t)J(\theta_t)^\top}, & \text{if} \sqrt{J(\theta_t)J(\theta_t)^\top} < k_m \\ 0.0, & \text{otherwise} \end{cases} \tag{4.12}$$

Self Collision Avoidance

Computing self-collision between the links of the robot for a large number of configurations can be computationally expensive [112, 386]. Hence, similar to previous approaches [112, 386], we train a neural network that predicts the closest distance [1] between the links of the robot given a joint configuration (θ). One difference in our approach is the use of positional encoding (i.e.,$[\sin(\theta), \cos(\theta)]$) as we found this to improve the accuracy of the distance prediction [325]. Our network (which we call jointNERF) has three layers with $[256, 128, 64]$ neurons and ReLU activations. We compute a cost term as shown below,

$$\hat{c}_{\text{self-coll}}(\theta_t) = \max(0, \text{jointNERF}(\theta_t)) \tag{4.13}$$

Environment Collision Avoidance

Safe operation in cluttered environments requires a tight coupling between perception and control for collision avoidance. Gradient-based approaches [543] generally rely on either known object shapes or pre-computed signed distance fields that provide gradient information for optimization. However, our sampling-based approach can handle discrete costs and

[1] Distance is positive when two links are penetrating and negative when not colliding.

as such we explore collision avoidance without using signed distances. Specifically, we use a learned collision checking function from Danielczuk *et al.* [112] that operates directly on raw pointcloud data and classifies if an robot link pointcloud pc_l is in collision with the environment pointcloud pc_{env} given the robot link's pose X_l.

$$\hat{c}_{\text{coll}}(pc_l, pc_{env}, X_l) = \begin{cases} 1, & \text{if } \textit{collision}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.14}$$

Finally our running cost function is given by

$$\hat{c}(x_t, u_t) = \alpha_p \hat{c}_{\text{pose}} + \alpha_s \hat{c}_{\text{stop}} + \alpha_j \hat{c}_{\text{joint}} + \alpha_m \hat{c}_{\text{manip}} + \alpha_c(\hat{c}_{\text{self-coll}} + \hat{c}_{\text{coll}}) \tag{4.15}$$

4.3.3 Sampling Strategy for Control Sequences

The method used for sampling controls from the Gaussian policy can have a great impact on the convergence of the optimization and can help embed different desirable behaviors such as ensuring smoothness. Pseudorandom sequences typically used in Monte Carlo integration exhibit an undesirable clustering of sampled points which results in empty regions. Whereas low-discrepancy sequences, where *low discrepancy* refers to the degree of deviation from perfect uniform sampling, alleviate this problem by defining deterministic samplers that correlate each point to avoid groupings [352]. Halton sequences [179] are a widely used form of *low discrepancy* number generators that attempt to improve the rate of convergence of Monte Carlo algorithms, and are reported to achieve superior performance in high-dimensional problems [127]. In particular, the Halton sequence uses the first $p_i, \ldots, p_d$ prime numbers to define a sequence, $\mathbf{w}_1, \mathbf{w}_2, \ldots$, for integers $i \geq 0$ and $b \geq 2$ where $\mathbf{w}_i = (\phi_{p_1}(i), \ldots, \phi_{p_d}(i))$ and $\phi_b(i) = \sum_{a=1}^{\infty} i_a b^{-a}$, with $i = \sum_{a=1}^{\infty} i_a b^{a-1}$ for $i_0, i_1, \cdots \in \{0, 1, \ldots, b-1\}$ [2]. We incorporate Halton sequences for sampling controls that can provide a better estimate of the objective function gradient. Controls from the Halton sequence are sampled once at the beginning and then transformed using the mean (μ_t) and Covariance $\mathbf{\Sigma}_t$ of the current Gaussian policy.

[2] See http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/ for a visualization of different low-discrepancy methods.

Furthermore, we explore two different strategies for enforcing smoothness in sampled control sequences. The first method is a *comb* filter that uses user-specified coefficients $[c_1, c_2, c_3]$ to filter out each sampled control trajectory along the horizon as $u_{t,h} = c_1 u_{t,h} + c_2 u_{t,h-1} + c_3 u_{t,h-1}$ This method has previously been used with sampling-based control techniques [449, 521], however, it requires extensive tuning and filtered trajectories are not guaranteed smooth as neighboring samples can have large difference in magnitude.

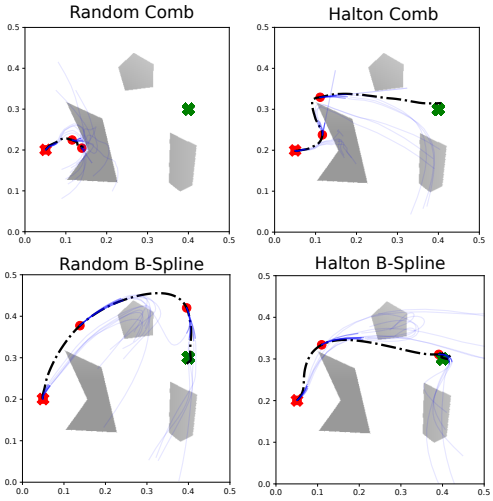


Figure 4.3: We compare our sampling scheme on a planar robot reacher task. The holonomic robot must move from its initial position (green cross) to the desired position (red cross) while avoiding obstacles(grey regions). The path taken by the robot is the black dot-dashed line and red circles are positions of the robot after every 30 timesteps. The blue lines are the top 5 rolled out trajectories. Our Halton B-Spline is able to find a smooth short path to the goal while Random B-Spline takes a longer path. Halton with a comb filter is not smooth as shown by the sudden path changes.

We propose an alternate strategy to enforce smoothness by fitting B-splines of degree 3 to controls sampled using a Halton Sequence. The resulting curve is sub-sampled at a finer resolution to obtain smooth joint acceleration samples which are then integrated to obtain corresponding joint velocity and position trajectories using Eq. 4.7. In Fig. 4.3 we show a qualitative comparison between different combinations of sampling and smoothing strategies for a planar robot trying to reach a goal while avoiding obstacles where we see that our Halton + B-Spline sampling strategy is able to better explore the action space while maintaining smoothness.

Covariance Parameterization: Conventionally, sampling-based MPC algorithms such as MPPI parameterize the covariance of the Gaussian policy to be of the form $\Sigma_{t,h} = \sigma_u * I_{d\times d}$ where σ_u is a scalar value and $I_{d\times d}$ is a $d\times d$ identity matrix, which forces the covariance to be the same across all control dimensions. However, in the case of manipulators it is desirable to allow the covariance of different joints to adapt independently so they can potentially optimize different cost terms such as pose reaching versus increasing manipulability. Thus we also consider covariance of the form $\Sigma^U_{t,h} = \sigma_u^T I_{d\times d}$ where $\sigma_u = [\sigma_1, \ldots, \sigma_d]$. Each term in

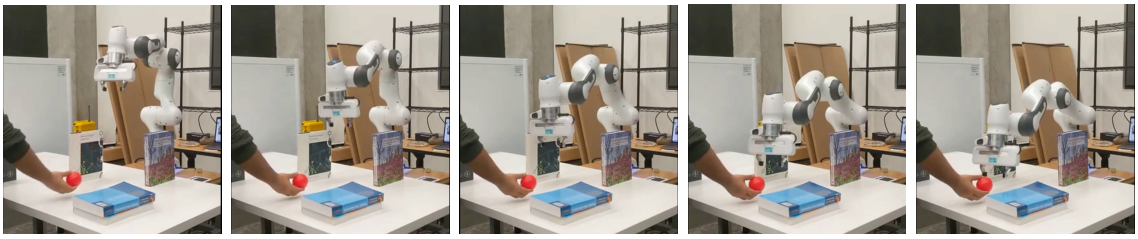


Figure 4.4: We show a sequence from our collision avoidance experiment where the robot has to move between thin walls to reach the orange ball held by the human. The robot tries to reach the ball but moves only as close as possible as any further motion would cause the elbow to hit the right book.

σ_u is then adapted based on the rollouts. Adapting the covariance along action dimensions has also been employed by [103] for CEM.

Our sampling strategy also offers us the flexibility of incorporating certain fixed set of action trajectories which could be task-dependent or even a library of pre-defined desired motions. We leverage this fact by incorporating a set of zero acceleration or "null" particles which allows the robot to coast at a constant velocity once the robot is accelerated sufficiently and also easily stop at the goal as demonstrated in our experiments.

4.4 Experiments

Through our experiments we aim to analyze the effectiveness of STORM as a framework for real-time, perception-driven feedback control in real-world manipulation scenarios. To this end, we first study the performance of STORM in reacting to changing end-effector targets from perception data while satisfying task constraints such as maintaining user-specified orientation and avoiding obstacles in cluttered scenes. Second, we consider the dynamic task of balancing a ball on a tray grasped by the end effector that uses an approximate model of the ball dynamics. We provide supplementary material, results of ablation studies in simulation for different components of our framework such as cost terms, sampling strategy and policy parameterization, a comparison to MoveIt! and OSC [187] for the standard pose reaching problem as well as a qualitative comparison to Reimannian Motion Policies (RMPs) [394] at https://sites.google.com/view/manipulation-mpc.

4.4.1 Tracking Moving Targets while Handling Task Constraints

A key strength of feedback-based MPC over "plan and execute" and OSC approaches is its ability to simultaneously optimize complex cost functions over a long horizon while demonstrating reactive behavior. We demonstrate this by having the robot react to changing goal poses obtained from noisy perception, while satisfying task constraints such as maintaining desired orientation and avoiding obstacles. In these experiments, a ball held by a human is tracked using a depth camera and the robot tries to reach the ball as the human moves it to different locations in the workspace.

Obstacle Avoidance: Demonstrating reactive motion and reasoning about obstacle avoidance in cluttered environments, while simultaneously coordinating large degrees of freedom leads to a hard online optimization problem. Standard planning and OSC approaches often fall short in optimizing for such behavior.

To test our method's capability to handle such scenarios, we setup two different table top environments, one consisting of two blocks representing common pick and place environments as shown in Fig. 1 and an environment with thin walls to represent a densely occupied space, as shown in Fig. 4.4. We use our perception based ball tracker to make the robot reach different positions in the environment. During the experiment, we also move the ball to some positions that are not reachable by the robot due to possibility of collision between the robot's links and the obstacles. As seen in Fig. 4.4, the robot handles these situations very well as it prioritizes collision avoidance over reaching the pose accurately. We present our full obstacle avoidance experiments as well as several experiments in simulation in the accompanying videos.

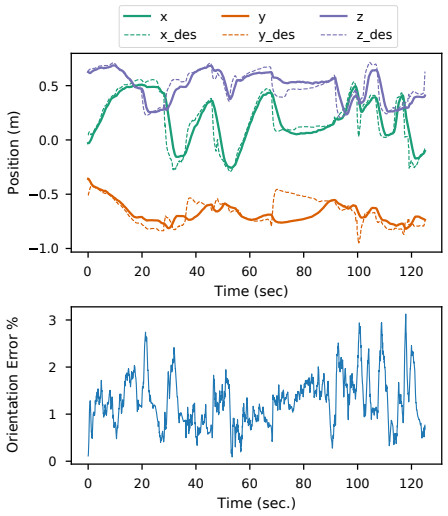


Figure 4.5: We move the ball across the workspace while having a high weight on maintaining a specific orientation of the end-effector. Our control scheme can maintain the orientation during motion as seen by the very low orientation error ($< 3\%$).

Orientation Constraints: Several common manipulation tasks such as moving a filled

cup require maintaining orientation during motion which reduces the feasibility region of a controller. We test this scenario by imposing orientation constraints on the end-effector while tracking the ball. Our controller achieves a median quaternion error of 1.2485% while tracking the ball with sufficient accuracy as shown in Fig. 4.5.

4.4.2 Dynamic Object Balancing

We consider a hard dynamic manipulation task where the robot tries to balance a ball placed on a tray grasped by a parallel jaw gripper as shown in Fig. 6.1. The location of the ball is tracked using the RGBD input from a RealSense camera at 30Hz. We use a simplified dynamics model of the object rolling on the tray under acceleration due to gravity and do not explicitly account for friction or inertial properties of the ball and do not perform any system identification. Fig. 4.6 shows a snapshot of the robot performing the task. The purpose of this task is to demonstrate the robustness of MPC under severe model-bias while optimizing complex objectives and dealing with noisy perception. We refer the reader to the supplementary material on the project website for more details of the experimental setup and ball dynamics.

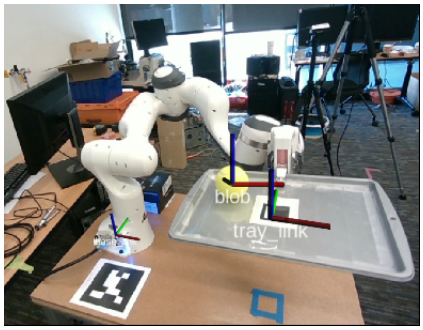


Figure 4.6: Snapshot of Ball Balancing

Analysis: Fig. 4.7 shows a superimposed plot of the (x,y) trajectories ball with respect to the tray frame for 10 different trials of the experiment. We observe that, even with our highly biased model and noise due to perception and state estimation, our MPC framework is able to achieve a median error of 3.9 cm in positioning the ball at the center of the tray. Moreover, the robot did not drop the ball in any trial. This experiment demonstrates the efficacy of MPC in correcting for model bias while maintaining reactivity and handling complex task constraints.

Our experimental results indicate that accurate and reactive manipulation behavior can be obtained by using relatively simple models and intuitive cost functions in an MPC framework. Sampling-based MPC also provides us the flexibility to encode different desired be-

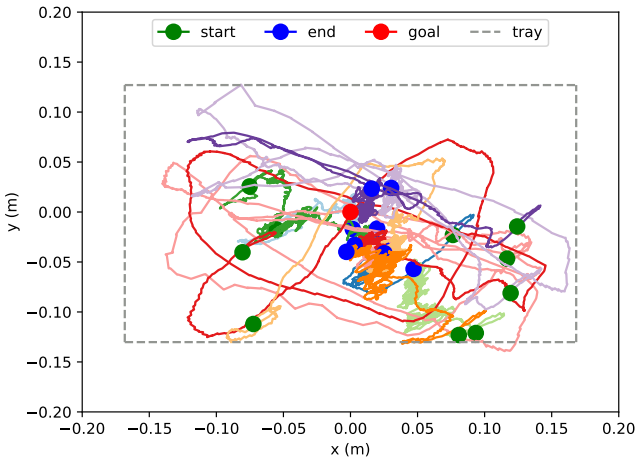


Figure 4.7: Dynamic Balancing Task: Trajectories of ball in tray frame for 10 different episodes. In every episode the robot starts from the home configuration and the ball is placed at an arbitrary location on the tray. Our framework is able to achieve a median error of 3.9 cm. Note that due to perception errors, sometimes the center of the ball is detected to be outside the tray when near the edge.

haviors such as smooth motions directly in the optimization and tightly couple perception with control which are key components for real-world robot control.

4.5 Related Work

Perception driven feedback control on high dimensional systems is a large field of research with several existing approaches [234]. Operation Space Controllers (OSC) are some of the fastest algorithms available for feedback control, with methods achieving control latency of 1-2 ms [84, 128, 187, 386]. However OSC methods rely heavily on a higher level planner for avoiding local minima (e.g.obstacles).

A more global approach has been explored by reformulating standard motion planning methods to do feedback control via online replanning [21, 238, 267, 341, 480]. However most of these methods run at a slow rates on high dimensional systems, with control latencies between 140ms and 1000ms [21, 238, 267, 480]. Murray *et al.* [341] researched leveraging a custom chip to do fast parallel collision checking and use this with a PRM style planner

to replan at 1ms for reaching Cartesian Poses in a semi-structured environment. Their chip based collision checker uses a complete pointcloud of the environment obtained by placing many cameras in the environment and combining their pointclouds. This is an highly unrealistic setting for real world manipulation in unstructured environments.

In the realm of Model predictive control approaches, only gradient based joint space MPC methods have shown to work on real manipulation systems as their control latency is in an acceptable range (20ms - 125ms) for feedback control [141, 150, 212]. Ishihara *et al.* [212] explore two stage hierarchical ILQR for fast MPC on a humanoid robot. Their two stage approach enables a very low control latency of 20ms. Erez *et al.* [141] explore ILQR on humanoid robots leveraging a simulator for the dynamics model. Fishman *et al.* [150] use gradient based MPC for finding trajectories for a manipulator while simultaneously predicting user intent in a human robot interaction setting. They solve the optimization problem leveraging Levenberg-Marquardt at a control latency of 140ms. Hogan and Rodriguez [198] explore gradient based MPC in the task space for planar non-prehensile manipulation leveraging a learned mode switcher to switch between different dynamic models. They are able to obtain a control latency of 5ms as their dynamic models are smooth.

Sampling-based methods have a rich history in MPC. Model-Predictive Path Integral Control (MPPI) [505] is one of the leading sampling-based MPC approaches that has shown great performance on real-world aggressive off-road autonomous driving by leveraging learned models [506] and GPU acceleration [504]. Wagener *et al.* [493] analyze MPC algorithms from the perspective of online learning and show connections between different methods such as Cross-Entropy method (CEM) and MPPI and have also demonstrated control rates of 40Hz with 1200 samples and a horizon of 2.5 seconds for off-road driving using GPU acceleration.

However, in the context of manipulation, sampling-based control in joint space has only been explored by optimizing in the joint position space without considering velocity and acceleration limits [112, 208, 209]. Danielczuk *et al.* [112] learn a collision classifier and do online replanning in joint space leveraging an inverse kinematic function to get a goal joint configuration and find straight line paths in joint space to reach the goal while avoiding collisions. Their approach doesn't handle different task spaces directly in the form of cost functions and also has a much larger control latency of 1000 ms. Hyatt *et al.* [208, 209]

compare sampling based MPC with gradient based MPC on large dimensional robots with piecewise linear functions. They show that sampling based MPC can run at 200Hz (5ms) even with large number of dimensions due to it's parallelizability on the GPU. However, they do not explore collision avoidance or task space cost terms in their approach and leave it for future work. Pinneri *et al.* [375] provide a method for adding correlated noise to action samples in Cross-Entropy Method to reduce the number of particles required for obtaining good performance on high-dimensional control tasks. However, their multi-threaded CPU implementation is unable to achieve real-time performance. Their experiments are also limited to simulation with access to true dynamics.

Sampling based optimization has also been used for motion planning in high dimensional systems [231, 249, 250]. Kalakrishnan *et al.* [231] formulate planning as a stochastic trajectory optimization problem and plan over the joint position space to reach Cartesian positions while orientation constraints on the end-effector. They structure their co-variance matrix based on the finite difference matrix to sample delta joint positions that start and stop at 0 with a smooth profile in between. This sampling along with projecting the weighted action through this matrix, pushes the optimization towards a smooth trajectory for execution on the real robot. Kobilarov [249, 250] explored using cross-entropy optimization for motion planning and showed global convergence in very tight environments on quadrotors and simple planar environments.

4.6 Discussion

In this chapter, we presented a sampling-based MPC framework for manipulation that leverages approximate dynamics models to optimize behaviors in the joint space and is able achieve smooth and reactive motions while respecting constraints. We demonstrated its performance on several real-world dynamic manipulation tasks. The first key component of our approach is a fully tensorized kinematic model that allows for GPU-based acceleration of rollouts. Second, we demonstrated how intuitive cost terms can encourage desirable behaviors. Third, this formulation allows us to leverage diverse sampling strategies to embed desirable properties directly into the optimization. This work opens up exciting directions for future research. First, performance can be made more robust by directly accounting for

state uncertainty in the control loop. Second, at higher speeds and in presence of multiple objects, the simple models used might induce significant bias. This might require reasoning about more complicated dynamical interactions, and it is interesting to explore how general purpose GPU-based simulators [296] can be integrated into the framework as well. This question also motivates our exploration in the next chapter of how more generally model-free RL can be used to mitigate the effects of model-bias while maintaining computational speed.

Chapter 5

BLENDING MPC & VALUE FUNCTION APPROXIMATION FOR EFFICIENT REINFORCEMENT LEARNING

5.1 Introduction

In recent years, model-free RL approaches have led to exciting breakthroughs in challenging sequential decision-making problems including simulated high-dimensional robotics control tasks [176, 420] as well as video and board games [430, 431]. While these approaches are extremely general, and can theoretically solve complex problems with little prior knowledge, they also typically require a large quantity of training data to succeed. In robotics and engineering domains, data may be collected from real-world interaction, a process that can be dangerous, time consuming, and expensive.

In Chapter 4, we demonstrated how MPC can offer a simpler, more practical alternative. In contrast to RL that typically uses data to learn a global model offline, which is then deployed at test time, MPC solves for a policy *online* by optimizing an approximate model for a finite horizon at a given state. This policy is then executed for a single timestep and the process repeats. As also discussed in Chapter 4, MPC is one of the most popular approaches for control of complex, safety-critical systems such as autonomous helicopters [3], aggressive off-road vehicles [505] and humanoid robots [142], owing to its ability to use approximate models to optimize complex cost functions with nonlinear constraints [320, 321].

However, approximations in the model used by MPC can significantly limit performance. Specifically, *model bias* may result in *persistent errors* that eventually compound and become catastrophic. For example, in non-prehensile manipulation, practitioners often use a simple quasi-static model that assumes an object does not roll or slide away when pushed. For more dynamic objects, this can lead to aggressive pushing policies that perpetually over-correct, eventually driving the object off the surface.

Recently, there have been several attempts to combine MPC with model free RL, showing

that the combination can improve over the individual approaches alone. Many of these approaches involve using RL to learn a terminal cost function, thereby increasing the effective horizon of MPC [44, 312, 541]. However, the learned value function is only applied at the end of the MPC horizon. Model errors would still persist in horizon, leading to sub-optimal policies. Similar approaches have also been applied to great effect in discrete games with known models [25, 430, 431], where value functions and policies learned via model-free RL are used to guide Monte-Carlo Tree Search. In this chapter, we focus on a broader question: can machine learning be used to both increase the effective horizon of MPC, while also correcting for model bias?

One straightforward approach is to try to learn (or correct) the MPC model from real data encountered during execution; however there are some practical barriers to this strategy. Hand-constructed models are often crude-approximations of reality and lack the expressivity to represent encountered dynamics. Moreover, increasing the complexity of such models leads to computationally expensive updates that can harm MPC's online performance. Model-based RL approaches such as [103, 343, 428] aim to learn general neural network models directly from data. However, learning globally consistent models is an exceptionally hard task due to issues such as covariate shift [400].

We propose a framework, MPQ(λ), for weaving together MPC with learned value estimates to trade-off errors in the MPC model and approximation error in a learned value function. Our key insight is to view MPC as tracing out a series of local Q-function approximations. We can then blend each of these Q-functions with value estimates from reinforcement learning. We show that by using a blending parameter λ, similar to the trace decay parameter in TD(λ), we can systematically trade-off errors between these two sources. Moreover, by smoothly decaying λ over learning episodes we can achieve the best of both worlds - a policy can depend on a prior model before its has encountered any data and then gradually become more reliant on learned value estimates as it gains experience.

5.1.1 Contributions

1. We present a framework that unifies MPC and model-free RL through the perspective of value function approximation.

2. We present a theoretical analysis of finite horizon planning with approximate models and value functions.

3. We empirically evaluate our proposed approach on challenging manipulation problems with varying degrees of model-bias.

5.2 Problem Formulation

5.2.1 Reinforcement Learning

We consider an agent acting in an infinite-horizon discounted Markov Decision Process (MDP). An MDP is defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, c, P, \gamma, \mu)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $c(s, a)$ is the per-step cost function, $s_{t+1} \sim P(\cdot|s_t, a_t)$ is the stochastic transition dynamics and γ is the discount factor and $\mu(s_0)$ is a distribution over initial states. A closed-loop policy $\pi(\cdot|s)$ outputs a distribution over actions given a state. Let $\mu^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}$ be the distribution over state-action trajectories obtained by running policy π on $\mathcal{M}$. The value function for a given policy π, is defined as $V^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) = \mathbb{E}_{\mu^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) \mid s_0 = s\right]$ and the action-value function as $Q^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s, a) = \mathbb{E}_{\mu^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) \mid s_0 = s, a_0 = a\right]$. The objective is to find an optimal policy $\pi^* = \arg\min_{\pi} \mathbb{E}_{s_0 \sim \mu}\left[V^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s_0)\right]$. We can also define the (dis)-advantage function $A^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s, a) = Q^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s, a) - V^{\pi}(s)$, which measures how good an action is compared to the action taken by the policy in expectation. It can be equivalently expressed in terms of the Bellman error as $A^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s, a) = c(s, a) + \gamma \mathbb{E}_{s' \sim P, a' \sim \pi}\left[Q^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s', a')\right] - \mathbb{E}_{a \sim \pi}\left[Q^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s, a)\right]$.

5.2.2 Model-Predictive Control as Q-Function Approximation

MPC is a widely used technique for synthesizing closed-loop policies for MDPs. Instead of trying to solve for a single, globally optimal policy, MPC follows a more pragmatic approach

of optimizing simple, local policies *online*. At every timestep on the system, MPC uses an approximate model of the environment to search for a parameterized policy that minimizes cost over a finite horizon. An action is sampled from the policy and executed on the system. The process is then repeated from the next state, often by warm-starting the optimization from the previous solution.

We formalize this process as solving a simpler *surrogate* MDP $\hat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \hat{c}, \hat{P}, \gamma, \hat{\mu}, H)$ online, which differs from $(\mathcal{S}, \mathcal{A}, c, P, \gamma, \mu)$ by using an approximate cost function $\hat{c}$, transition dynamics $\hat{P}$ and limiting horizon to H. Since it plans to a finite horizon, it's also common to use a terminal state-action value function $\hat{Q}$ that estimates the cost-to-go. The start state distribution $\hat{\mu}$ is a dirac-delta function centered on the current state $s_0 = s_t$. MPC can be viewed as iteratively constructing an estimate of the Q-function of the original MDP $(\mathcal{S}, \mathcal{A}, c, P, \gamma, \mu)$, given policy π_ϕ at state s:

$$Q_H^\phi(s, a) = \mathbb{E}_{\mu_{\hat{\mathcal{M}}}^{\pi_\phi}} \left[\sum_{i=0}^{H-1} \gamma^i \hat{c}(s_i, a_i) + \gamma^H \hat{Q}(s_H, a_H) \mid s_0 = s, a_0 = a \right] \tag{5.1}$$

MPC then iteratively optimizes this estimate (at current system state s_t) to update the policy parameters

$$\phi_t^* = \arg\min_\phi Q_H^\phi(s_t, \pi_\phi(s_t)) \tag{5.2}$$

Alternatively, we can also view the above procedure from the perspective of disadvantage minimization. Let us define an estimator for the 1-step disadvantage with respect to the potential function $\hat{Q}$ as $A(s_i, a_i) = c(s_i, a_i) + \gamma \hat{Q}(s_{i+1}, a_{i+1}) - \hat{Q}(s_i, a_i)$. We can then equivalently write the above optimization as minimizing the discounted sum of disadvantages over time via the telescoping sum trick

$$\arg\min_{\pi \in \Pi} \mathbb{E}_{\mu_{\hat{\mathcal{M}}}^{\pi_\phi}} \left[\hat{Q}(s_0, a_0) + \sum_{i=0}^{H-1} \gamma^i A(s_i, a_i) \mid s_0 = s_t \right] \tag{5.3}$$

Although the above formulation queries the $\hat{Q}$ at every timestep, it is still exactly equivalent to the original problem and hence, does not mitigate the effects of model-bias. In the next section, we build a concrete method to address this issue by formulating a novel way to blend Q-estimates from MPC and a learned value function that can balance their respective errors.

5.3 Approach

In this section, we develop our approach to systematically deal with model bias in MPC by blending-in learned value estimates. First, we take a closer look at the different sources of error in the estimate in Eq. 5.1 and then propose an easy-to-implement, yet effective strategy for trading them off.

5.3.1 Sources of Error in MPC

The performance of MPC algorithms critically depends on the quality of the Q-function estimator $Q_H^{\phi}(s,a)$ in Eq. 5.1. There are three major sources of approximation error. First, model bias can cause compounding errors in predicted state trajectories, which biases the estimates of the costs of different action sequences. The effect of model error becomes more severe as $H \to \infty$. Second, the error in the terminal value function gets propagated back to the estimate of the Q-function at the start state. With discounting, the effect of error due to inaccurate terminal value function diminishes as H increases. Third, using a small H with an inaccurate terminal value function can make the MPC algorithm greedy and myopic to rewards further out in the future.

We can formally bound the performance of the policy with approximate models and approximate learned value functions. In Theorem 5, we show the loss in performance of the resulting policy as a function of the model error, value function error and the planning horizon.

Theorem 5 (Proof Appendix 5.7.1)**.** *Let MDP $\hat{\mathcal{M}}$ be an α-approximation of $\mathcal{M}$ such that $\forall(s,a)$, we have $\left|\left|\hat{P}(s'|s,a) - P(s'|s,a)\right|\right|_1 \leq \alpha$ and $|\hat{c}(s,a) - c(s,a)| \leq \alpha$. Let the learned value function $\hat{Q}(s,a)$ be an ϵ-approximation of the true value function $\left|\left|\hat{Q}(s,a) - Q^{\pi^*}_{\mathcal{M}}(s,a)\right|\right|_\infty \leq \epsilon$. The performance of the MPC policy is bounded w.r.t the optimal policy as $\left|\left|V^{\pi^*}_{\mathcal{M}}(s) - V^{\hat{\pi}}_{\mathcal{M}}(s)\right|\right|_\infty$*

$$\leq 2\left(\frac{\gamma(1-\gamma^{H-1})}{(1-\gamma^H)(1-\gamma)}\alpha H\left(\frac{c_{\max}-c_{\min}}{2}\right) + \frac{\gamma^H \alpha H}{1-\gamma^H}\left(\frac{V_{\max}-V_{\min}}{2}\right) + \frac{\alpha}{1-\gamma} + \frac{\gamma^H \epsilon}{1-\gamma^H}\right) \tag{5.4}$$

This theorem generalizes over various established results. Setting $H = 1, \epsilon = 0$ gives us the 1-step simulation lemma in [242] (Appendix 5.7.1). Setting $\alpha = 0$, i.e. true model, recovers the cost-shaping result in [450].

Further inspecting Eq. 5.4, we see that the model error *increases* with horizon H (the first two terms) while the learned value error *decreases* with H, which matches our intuitions.

In practice, the errors in model and value function are usually unknown and hard to estimate making it impossible to optimally set the MPC horizon. Instead, we next propose a strategy to blend the Q-estimates from MPC and the learned value function at every timestep along the horizon, instead of just the terminal step such that we can properly balance the different sources of error.

5.3.2 Blending Model Predictive Control and Value Functions

A naive way to blend Q-estimates from MPC with Q-estimates from the value function would be to consider a convex combination of the two

$$(1-\lambda)\ \underbrace{\hat{Q}(s,a)}_{\text{model-free}} + \lambda\ \underbrace{Q_H^{\phi}(s,a)}_{\text{model-based}} \tag{5.5}$$

where $\lambda \in [0,1]$. Here, the value function is contributing to a residual that is added to the MPC output, an approach commonly used to combine model-based and model-free methods [286]. However, this is solution is rather *ad hoc*. If we have at our disposal a value function, why invoke it at only at the first and last timestep? As the value function gets better, it should be useful to invoke it *at all timesteps*.

Instead, consider the following recursive formulation for the Q-estimate. Given (s_i, a_i), the state-action encountered at horizon i, the blended estimate $Q^{\lambda}(s_i, a_i)$ is expressed as

$$\underbrace{Q^{\lambda}(s_i,a_i)}_{\text{current blended estimate}} = (1-\lambda)\underbrace{\hat{Q}(s_i,a_i)}_{\text{model-free}} + \lambda\Big(\underbrace{\hat{c}(s_i,a_i)}_{\text{model-based}} + \gamma\ \underbrace{Q^{\lambda}(s_{i+1},a_{i+1})}_{\text{future blended estimate}}\Big) \tag{5.6}$$

where $\lambda \in [0,1]$. The recursion ends at $Q^{\lambda}(s_H, a_H) = \hat{Q}(s_H, a_H)$. In other words, the current blended estimate is a convex combination of the model-free value function and the one-step model-based return. The return in turn uses the future blended estimate. Note unlike Eq. 5.5, the model-free estimate is invoked at *every timestep*.

We can unroll Eq. 5.6 in time to show $Q_H^{\lambda}(s,a)$, the blended $H-$horizon estimate, is simply an exponentially weighted average of *all horizon* estimates

$$Q_H^{\lambda}(s,a) = (1-\lambda)\sum_{i=0}^{H-1}\lambda^i Q_i^{\phi}(s,a) + \lambda^H Q_H^{\phi}(s,a) \tag{5.7}$$

where $Q_k^\phi(s,a) = \mathbb{E}_{\mu^{\pi_\phi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}}\left[\sum_{i=0}^{k-1}\gamma^i\hat{c}(s_i,a_i) + \gamma^k\hat{Q}(s_k,a_k) \mid s_0 = s, a_0 = a\right]$ is a k-horizon estimate. When $\lambda = 0$, the estimator reduces to the just using $\hat{Q}$ and when $\lambda = 1$ we recover the original MPC estimate Q_H^ϕ in Eq. 5.1. For intermediate values of λ, we interpolate smoothly between the two by interpolating all H estimates.

Implementing Eq. 5.7 naively would require running H versions of MPC and then combining their outputs. This is far too expensive. However, we can switch to the disadvantage formulation by applying a similar telescoping trick

$$Q_H^\lambda(s,a) = \mathbb{E}_{\mu^{\pi_\phi}_{\hat{\mathcal{M}}}}\left[\hat{Q}(s_0,a_0) + \sum_{i=0}^{H-1}(\gamma\lambda)^i A(s_i,a_i)\right] \tag{5.8}$$

This estimator has a similar form as the $TD(\lambda)$ estimator for the value function. However, while $TD(\lambda)$ uses the λ parameter for bias-variance trade-off, our blended estimator aims trade-off bias in dynamics model with bias in learned value function.

Why use blending λ when one can simply tune the horizon H? First, H limits the *resolution* we can tune since it's an integer – as H gets smaller the resolution becomes worse. Second, the blended estimator $Q_H^\lambda(s,a)$ uses far more samples. Say we have access to optimal horizon H^*. Even if both Q_H^λ and $Q_{H^*}^\phi$ had the same bias, the latter uses a strict subset of samples as the former. Hence the variance of the blended estimator will be lower, with high probability.

5.4 The MPQ(λ) Algorithm

We develop a simple variant of Q-Learning, called Model-Predictive Q-Learning with λ Weights (MPQ(λ)) that learns a parameterized Q-function estimate $\hat{Q}_\theta$. Our algorithm, presented in Alg. 6, modifies Q-learning to use blended Q-estimates as described in the Eq. 5.8 for both action selection and generating value targets. The parameter λ is used to trade-off the errors due to model-bias and learned Q-function, $\hat{Q}_\theta$. This can be viewed as an extension of the MPQ algorithm from [44] to explicitly deal with model bias by incorporating the learned Q-function at all timesteps. Unlike MPQ, we do not explicitly consider the entropy-regularized formulation, although our framework can be modified to incorporate soft-Q targets.

Algorithm 6: MPQ(λ)

Input: Initial Q-function weights θ, Approximate dynamics $\hat{P}$ and cost function $\hat{c}$

Parameter: MPC horizon H, λ schedule $[\lambda_1, \lambda_2, \ldots]$, discount factor γ, minibatch size K, num mini-batches N, update frequency t_{update}

$\mathcal{D} \leftarrow \emptyset$

for $t = 1 \ldots \infty$ **do**

`// Update` λ

$\lambda = \lambda_t$

`// Blended MPC action selection`

$\phi_t \leftarrow \arg\min_{\phi} \mathbb{E}_{\mu_{\hat{\mathcal{M}}}^{\pi_\phi}} \left[\hat{Q}_\theta(s_0, a_0) + \sum_{i=0}^{H-1} (\gamma\lambda)^i A(s_i, a_i) \mid s_0 = s_t \right]$

$a_t \sim \pi_{\phi_t}$

Execute a_t on the system and observe (c_t, s_{t+1})

$\mathcal{D} \leftarrow (s_t, a_t, c_t, s_{t+1})$

if $t \% t_{update} == 0$ **then**

Sample N minibatches $\left(\left\{ s_{k,n}, a_{k,n}, c_{k,n}, s'_{k,n} \right\}_{k=1}^{K} \right)_{n=1}^{N}$ from $\mathcal{D}$

`// Generate Blended MPC value targets`

$\hat{y}_{k,n} = c_{k,n} + \gamma \min \mathbb{E}_{\mu_{\hat{\mathcal{M}}}^{\pi_\phi}} \left[\hat{Q}_\theta(s_0, a_0) + \sum_{i=0}^{H-1} (\gamma\lambda)^i A(s_i, a_i) \mid s_0 = s'_{k,n} \right]$

Update θ with SGD on loss $\mathcal{L} = \frac{1}{N} \frac{1}{K} \sum_{n=1}^{N} \sum_{k=1}^{K} \left(\hat{y}_{k,n} - \hat{Q}_\theta(s_{k,n}, a_{k,n}) \right)^2$

At every timestep t, MPQ(λ) proceeds by using H-horizon MPC from the current state s_t to optimize a policy π_ϕ with parameters ϕ. We modify the MPC algorithm to optimize for the greedy policy with respect to the blended Q-estimator in (5.8), that is

$$\phi_t^* = \arg\min_{\phi} \mathbb{E}_{\mu_{\hat{\mathcal{M}}}^{\pi_\phi}} \left[\hat{Q}_\theta(s_0, a_0) + \sum_{i=0}^{H-1} (\gamma\lambda)^i A(s_i, a_i) \mid s_0 = s_t \right] \tag{5.9}$$

An action sampled from the resulting policy is then executed on the system. A commonly used heuristic is to warm start the above optimization by shifting forward the solution from the previous timestep, which serves as a good initialization if the noise in the dynamics

in small [494]. This can significantly cut computational cost by reducing the number of iterations required to optimize (5.9) at every timestep.

Periodically, the parameters θ are updated via stochastic gradient descent to minimize the following loss function with N mini-batches of experience tuples of size K sampled from the replay buffer

$$\mathcal{L}(\theta) = \frac{1}{N}\frac{1}{K}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(\hat{y}_{k,n} - \hat{Q}_\theta(s_{k,n}, a_{k,n})\right)^2 \tag{5.10}$$

The H-horizon MPC with blended Q-estimator is again invoked to calculate the targets

$$\hat{y}_j = c(s_j, a_j) + \gamma \min_{\phi} \mathbb{E}_{\mu_{\hat{\mathcal{M}}}^{\pi_\phi}}\left[\hat{Q}_\theta(s_0, a_0) + \sum_{i=0}^{H-1}(\gamma\lambda)^i A(s_i, a_i) \mid s_0 = s'_{k,n}\right] \tag{5.11}$$

Using MPC to reduce error in Q-targets has been previously explored in literature [44, 312], where the model is either assumed to be perfect or model-error is not explicitly accounted for. MPC with the blended Q-estimator and an appropriate λ allows us to generate more stable Q-targets than using Q_θ or model-based rollouts with a terminal Q-function alone. However, running H-horizon optimization for all samples in a mini-batch can be time-consuming, forcing the use of smaller batch sizes and sparse updates. In our experiments, we employ a practical modification where during the action selection step, MPC is also queried for value targets which are then stored in the replay buffer, thus allowing us to use larger batch sizes and updates at every timestep.

Finally, we also allow λ to vary over time. In practice, λ is decayed as more data is collected on the system. Intuitively, in the early stages of learning, the bias in $\hat{Q}_\theta$ dominates and hence we want to rely more on the model. A larger value of λ is appropriate as it up-weights longer horizon estimates in the blended-Q estimator. As $\hat{Q}_\theta$ estimates improve over time, a smaller λ is favorable to reduce the reliance on the approximate model.

5.5 Experiments

Task Details: We evaluate MPQ(λ) on simulated robot control tasks based on the MuJoCo simulator [478], including a complex manipulation task with a 7DOF arm and in-hand manipulation with a 24DOF anthropomorphic hand [384] as shown in Fig. 5.1. For each

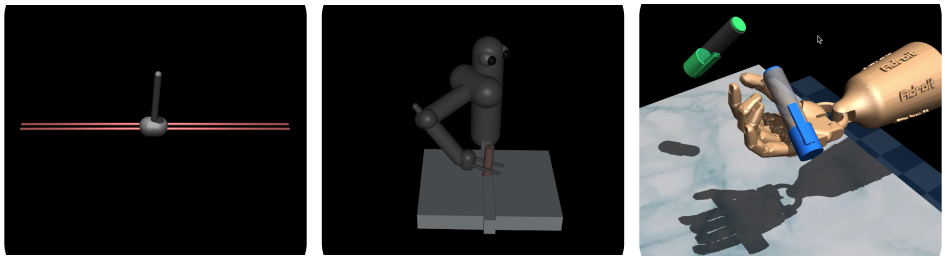


Figure 5.1: Tasks for evaluating MPQ(λ). Left to right - cartpole, peg insertion with 7DOF arm, and in-hand manipulation to orient align pen(blue) with target(green) with 24DOF dexterous hand.

task, we provide the agent with a biased version of simulation that is used as the dynamics model for MPC. We use Model Predictive Path Integral Control (MPPI) [506], a state-of-the-art sampling-based algorithm as our MPC algorithm throughout.

1. CartpoleSwingup: A classic control task where the agent slides a cart along a rail to swingup the pole attached via an unactuated hinge joint. Model bias is simulated by providing the agent incorrect masses of the cart and pole. The masses are set lower than the true values to make the problem harder for MPC as the algorithm will always input smaller controls than desired as also noted in [388]. Initial position of cart and pole are randomized at every episode.

2. SawyerPegInsertion: The agent controls a 7DOF Sawyer arm to insert a peg attached to the end-effector into a hole at different locations on a table in front of the robot. We test the effects of inaccurate perception by simulating a sensor at the target location that provides noisy position measurements at every timestep. MPC uses a deterministic model that does not account for sensor noise as common in controls. This biases the cost of simulated trajectories, causing MPC to fail to reach the target.

3. InHandManipulation: A challenging in-hand manipulation task with a 24DOF dexterous hand from [384]. The agent must align the pen with target orientation within certain tolerance for succcess. The initial orientation of the pen is randomized at every episode. Here, we simulate bias by providing larger estimates of the mass and

inertia of the pen as well as friction coefficients, which causes the MPC algorithm to optimize overly aggressive policies and drop the pen.

Appendix 5.7 provides more details of the tasks, success criterion and biased simulations.

Baselines: We compare MPQ(λ) against both model-based and model-free baselines - MPPI with true dynamics and no value function, MPPI with biased dynamics and no value function and Proximal Policy Optimization (PPO) [420].

Learning Details: We represent the Q-function with a feed-forward neural network. Simulation parameters like mass or friction coefficients are biased using the formula $m = (1 + b)m_{true}$, where b is a bias-factor. We also employ a practical modification to Alg. 6 to speed up training as discussed in Section 5.4. Instead of maintaining a large replay-buffer and re-calculating targets for every experience tuple in a mini-batch, as done by [44, 312], we simply query MPC for the value targets online and store them in a smaller buffer, which allows us to perform updates at every timestep. For PPO, we used the publically available implementation at https://bit.ly/330z5vi. Refer to the Appendix 5.7.2 for more details.

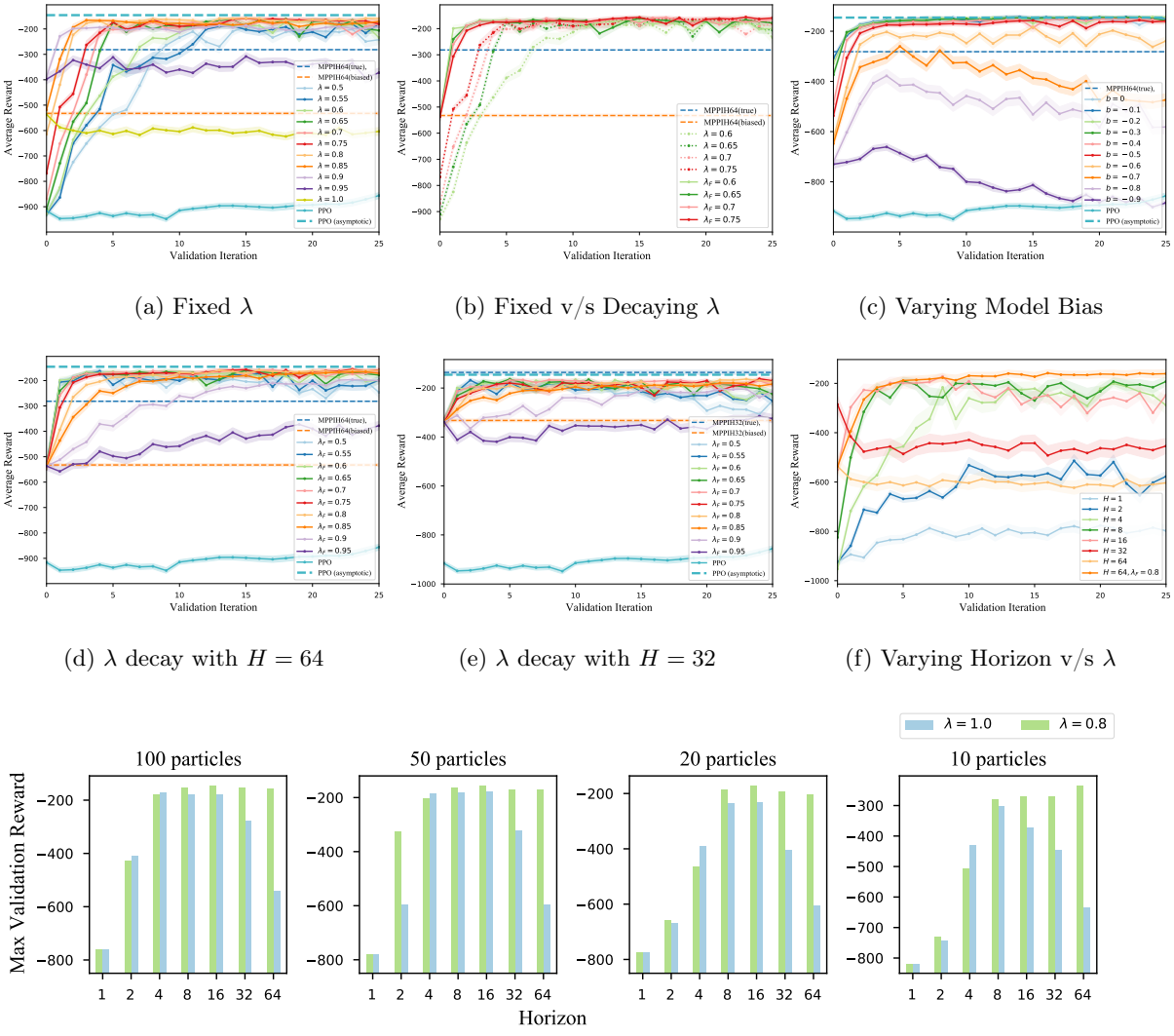
5.5.1 Analysis of Overall Performance

O 5. *MPQ(λ) is able to overcome model-bias in MPC for a wide range of λ values.*

Fig. 5.2(a) shows a comparison of MPQ(λ) with MPPI using true and biased dynamics with $b = -0.5$ and $H = 64$ for various settings of λ. There exists a wide range of λ values for which MPQ(λ) can efficiently trade-off model-bias with the bias in the learned Q-function and out-perform MPPI with biased dynamics. However, setting λ to a high value of 1.0 or 0.95, which weighs longer horizons heavily, leads to poor performance as compounding effects of model-bias are not compensated by Q_θ. Performance also drops as λ decreases below 0.6. MPQ(λ) outperforms MPPI with access to the true dynamics and reaches close to asymptotic performance of PPO. This is not surprising as the learned Q-function adds global information to the optimization and λ corrects for errors in optimizing longer horizons.

O 6. *Faster convergence can be achieved by decaying λ over time.*

As more data is collected on the system, we expect the bias in Q_θ to decrease, whereas model-bias remains constant. A larger value of λ that favors longer horizons is better



(a) Fixed λ (b) Fixed v/s Decaying λ (c) Varying Model Bias

(d) λ decay with $H = 64$ (e) λ decay with $H = 32$ (f) Varying Horizon v/s λ

(g) Bias-Variance Trade-off

Figure 5.2: CartpoleSwingup experiments. Solid lines show average rewards over 30 validation episodes (fixed start states) and 3 different seeds. The dashed lines are average reward of MPPI for the same validation episodes. Shaded region depicts the standard error of the mean Training is performed for 100k steps with validation after every 4k steps. In (b),(d),(e), (f) λ_F denotes the λ value at the end of training. PPO asymptotic performance is reported as average reward of last 10 validation iterations. (g) shows the best validation reward at the end of training for different horizon values and MPPI trajectory samples (particles) using $\lambda = 1.0$ and $\lambda = 0.8$

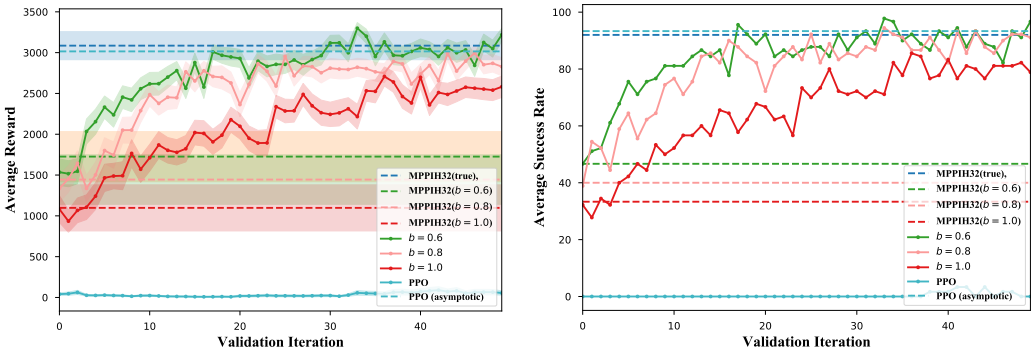
during initial steps of training as the effect of a randomly initialized Q_θ is diminished due to discounting and better exploration is achieved by forward lookahead. Conversely, as Q_θ gets more accurate, model-bias begins to hurt performance and a smaller λ is favorable. We test this by decaying λ in $[1.0, \lambda_F]$ using a fixed schedule and observe that indeed faster convergence is obtained by reducing the dependence on the model over training steps as shown in 5.2(b). Figures 5.2(d) and 5.2(e) present an ablations that show that MPQ(λ) is robust to a wide range of decay rates with $H = 64$ and 32 respectively. When provided with true dynamics, MPPI with $H = 32$ performs better than $H = 64$ due to optimization issues with long horizons. MPQ(λ) reaches performance comparable with MPPI $H = 32$ and asymptotic performance of PPO in both cases showing robustness to horizon values which is important since in practice we wish to set the horizon as large as our computation budget permits. However, decaying λ too fast or too slow can have adverse effects on performance. An interesting question for future work is whether λ can be adapted in a state-dependent manner. Refer to Appendix 5.7.2 for details on the decay schedule.

O 7. *MPQ(λ) is much more sample efficient compared to model-free RL on high-dimensional continuous control tasks, while maintaining asymptotic performance.*

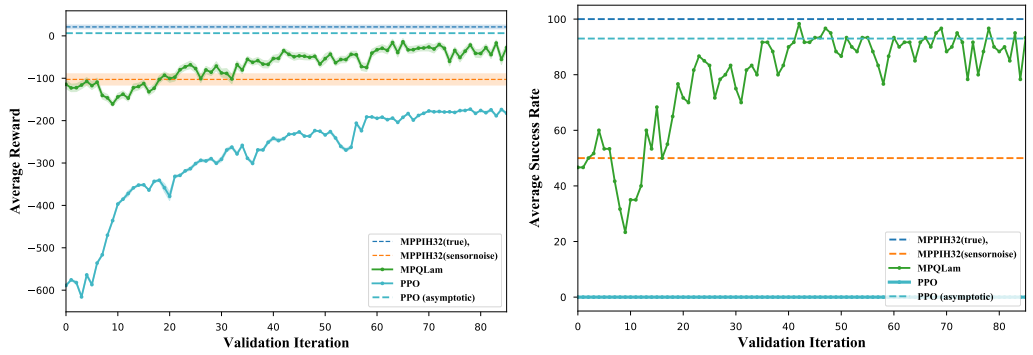
Figures 5.2 and 5.3 show a comparison of MPQ(λ) with the model-free PPO baseline. In all cases, we observe that MPQ(λ), through its use of approximate models, learned value functions, and a dynamically-varying λ parameter to trade-off different sources of error, rapidly improves its performance and achieves average reward and success rate comparable to MPPI with access to ground truth dynamics and model-free RL in the limit. In InHandManipulation, PPO performance does not improve at all over 150k training steps. In SawyerPegInsertion, the small magnitude of reward difference between MPPI with true and biased models is due to the fact that despite model bias, MPC is able to get the peg close to the table, but sensor noise inhibits precise control to consistently insert it in the hole. Here, the value function learned by MPQ(λ) can adapt to sensor noise and allow for fine-grained control near the table.

O 8. *MPQ(λ) is robust to large degree of model misspecification.*

Fig. 5.2(c) shows the effects of different values of the bias factor b used to vary the



(a) InHandManipulation Reward (b) InHandManipulation Success Rate



(c) SawyerPegInsertion Reward (d) SawyerPegInsertion Success Rate

Figure 5.3: Robustness and sample efficiency of MPQ(λ). (a),(b) Varying bias factor over mass, inertia and friction of pen (c),(d) Peg insertion with noisy perception. Same bias factor b is used for all altered properties per task. Curves depict average reward over 30 validation episodes with multiple seeds and shaded areas are the standard error of the mean. Asymptotic performance of PPO is average of last 10 validation iterations.

mass of the cart and pole for MPQ(λ) with a fixed λ decay rate of $[1.0, 0.75]$. MPQ(λ) achieves performance better than MPPI ($H = 64$) with true dynamics and comparable to model-free RL in the limit, for a wide range of bias factors b, and convergence rate is generally faster for smaller bias. For large values of b, MPQ(λ) either fails to improve or diverges as the compounding effects of model-bias hurt learning, making model-free RL the more favorable alternative. A similar trend is observed in Figures 5.3(a) and 5.3(b) where MPQ(λ) outperforms MPPI with corresponding bias in the mass, inertia and friction coefficients of the pen with atleast a margin of over 30% in terms of success rate. It also achieves performance comparable to MPPI with true dynamics and model-free RL, but is unable to do so for $b = 1.0$. We conclude that although MPQ(λ) is robust to large amount of model bias, if the model is extremely uninformative, relying on MPC can degrade performance.

O 9. *MPQ(λ) is robust to planning horizon and number of trajectory samples in sampling-based MPC.*

TD(λ) based approaches are traditionally used for bias-variance trade-off in model-free value function estimation. In our framework, λ plays a similar role, but it trades-off bias due to the dynamics model and learned value function against variance due to long-horizon rollouts. We empirically quantify this on the CartpoleSwingup task by training MPQ(λ) with different values of horizon and number of particles with $\lambda = 1.0$ and $\lambda = 0.8$ respectively. Results in Fig. 5.2(g) show that - (1) using λ can overcome effects of model-bias by irrespective of the planning horizon (except for very small values of $H = 1$ or 2) and (2) using λ can overcome variance due to limited number of particles with long horizon rollouts. The ablative study in Fig. 5.2(f) lends evidence to the fact that is preferable to simply decay λ over time than tuning the discrete horizon value to balance model bias. Not only does decaying λ achieve a better convergence rate and asymptotic performance, the performance is more robust to different decay rates (as evidenced from Fig. 5.2(d)), whereas the same does not hold for varying horizon.

5.6 Discussion

In this chapter, we presented a general framework to mitigate model-bias in MPC by blending model-free value estimates using a parameter λ, to systematically trade-off different sources of error. We developed a practical algorithm that is theoretically well-founded and achieves performance close to MPC with access to the *true* dynamics and asymptotic performance of model-free RL, while being sample efficient. For future work, an interesting avenue is to explore how λ can be automatically adapted in a state-dependent fashion. In particular, reasoning about the model and value function uncertainty may allow us to vary λ to rely more or less on our model in certain parts of the state space. Another promising direction is to extend the framework to explicitly incorporate constraints by exploring different constrained MPC approaches.

5.7 Appendix

5.7.1 Proofs

We present upper-bounds on performance of a greedy policy when using approximate value functions and models. We also analyze the case of finite horizon planning with an approximate dynamics model and terminal value function which can be seen as a generalization of [450]. For simplicity, we switch to using $\hat{V}(s)$ to the learnt model-free value function (instead of $\hat{Q}(s,a)$)

Let $\hat{V}(s)$ be an ϵ-approximation $\left|\left|\hat{V}(s) - V^{\pi^*}_{\mathcal{M}}(s)\right|\right|_\infty \leq \epsilon$. Let MDP $\hat{\mathcal{M}}$ be an α-approximation of $\mathcal{M}$ such that $\forall(s,a)$, we have $\left|\left|\hat{P}(s'|s,a) - P(s'|s,a)\right|\right|_1 \leq \alpha$ and $|\hat{c}(s,a) - c(s,a)| \leq \alpha$.

A Gentle Start: Bound on Performance of 1-Step Greedy Policy

Theorem 6. *Let the one-step greedy policy be*

$$\hat{\pi}(s) = \arg\min_{a \in \mathcal{A}} \hat{c}(s,a) + \gamma \Sigma_{s'} \hat{P}(s'|s,a)\hat{V}(s') \tag{5.12}$$

The performance loss of $\hat{\pi}(s)$ w.r.t optimal policy π^* on MDP $\mathcal{M}$ is bounded by

$$\left|\left|V^{\hat{\pi}}_{\mathcal{M}}(s) - V^{\pi^*}_{\mathcal{M}}(s)\right|\right|_\infty \leq \frac{2\left(\gamma\epsilon + \alpha + \gamma\alpha\left(\frac{V_{\max}-V_{\min}}{2}\right)\right)}{1-\gamma} \tag{5.13}$$

Proof. From equation 5.12 we have $\forall s \in \mathcal{S}$

$$\hat{c}(s,\hat{\pi}(s)) + \gamma\sum_{s'}\hat{P}(s'|s,\hat{\pi}(s))\hat{V}(s') \leq \hat{c}(s,\pi^*(s)) + \gamma\sum_{s'}\hat{P}(s'|s,\pi^*(s))\hat{V}(s')$$

$$\hat{c}(s,\hat{\pi}(s)) - \hat{c}(s,\pi^*(s)) \leq \gamma\left(\sum_{s'}\hat{P}(s'|s,\pi^*(s))\hat{V}(s') - \sum_{s'}\hat{P}(s'|s,\hat{\pi}(s))\hat{V}(s')\right)$$

$$\text{(using } \left|\left|\hat{V}(s) - V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s)\right|\right|_\infty \leq \epsilon\text{)}$$

$$\hat{c}(s,\hat{\pi}(s)) - \hat{c}(s,\pi^*(s)) \leq$$

$$\gamma\left(\sum_{s'}\hat{P}(s'|s,\pi^*(s))V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') - \sum_{s'}\hat{P}(s'|s,\hat{\pi}(s))V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s')\right) + 2\gamma\epsilon$$

$$\text{(using } |\hat{c}(s,a) - c(s,a)| \leq \alpha\text{)}$$

$$c(s,\hat{\pi}(s)) - c(s,\pi^*(s)) \leq$$

$$2\gamma\epsilon + 2\alpha + \gamma\left(\sum_{s'}\hat{P}(s'|s,\pi^*(s))V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') - \sum_{s'}\hat{P}(s'|s,\hat{\pi}(s))V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s')\right)$$

Now, let s be the state with the max loss $V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) - V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s)$,

$$V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) - V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) =$$
$$c(s,\hat{\pi}) - c(s,\pi^*) + \gamma\sum_{s'}\left(P(s'|s,\hat{\pi})V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') - P(s'|s,\pi^*)V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s')\right)$$

Substituting from equation 5.14

$$V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) - V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) \leq$$
$$2\gamma\epsilon + 2\alpha + \gamma\sum_{s'}\hat{P}(s'|s,\pi^*(s))V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') - \gamma\sum_{s'}\hat{P}(s'|s,\hat{\pi}(s))V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s')$$
$$- \gamma\sum_{s'}P(s'|s,\pi^*)V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') + \gamma\sum_{s'}P(s'|s,\hat{\pi})V^{\hat{\pi}}_{\mathcal{M}}(s')$$

Add and subtract $\gamma \sum_{s'} P(s'|s,\hat{\pi}) V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s')$ and re-arrange

$$
\begin{aligned}
&V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) - V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) \leq \\
&2\gamma\epsilon + 2\alpha + \gamma \sum_{s'} \left(\hat{P}(s'|s,\pi^*) - P(s'|s,\pi^*) \right) V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') \\
&\qquad - \gamma \sum_{s'} \left(\hat{P}(s'|s,\hat{\pi}) - P(s'|s,\hat{\pi}) \right) V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') \\
&\qquad + \gamma \sum_{s'} P(s'|s,\hat{\pi}) \left(V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') - V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') \right) \\
&\leq 2\gamma\epsilon + 2\alpha + 2\gamma\alpha \left(\frac{V_{\max} - V_{\min}}{2} \right) \\
&+ \gamma \sum_{s'} P(s'|s,\hat{\pi}) \left(V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') - V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s') \right)
\end{aligned}
$$

Since s is the state with largest loss

$$
\begin{aligned}
&\left|\left| V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) - V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) \right|\right|_\infty \leq \\
&2\gamma\epsilon + 2\alpha + 2\gamma\alpha \left(\frac{V_{\max} - V_{\min}}{2} \right) + \gamma \sum_{s'} P(s'|s,\hat{\pi}) \left|\left| V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) - V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) \right|\right|_\infty \\
&\leq 2\gamma\epsilon + 2\alpha + 2\gamma\alpha \left(\frac{V_{\max} - V_{\min}}{2} \right) + \gamma \left|\left| V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) - V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) \right|\right|_\infty
\end{aligned}
$$

Re-arranging terms we get

$$
\left|\left| V^{\pi^*}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) - V^{\hat{\pi}}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}(s) \right|\right|_\infty \leq \frac{2\left(\gamma\epsilon + \alpha + \gamma\alpha\left(\frac{V_{\max} - V_{\min}}{2}\right)\right)}{1-\gamma} \tag{5.14}
$$

which concludes the proof. □

Bound on Performance of H-Step Greedy Policy

Notation: For brevity let us define the following macro,

$$
\langle V, \pi, \mathcal{M} \rangle_H = \mathbb{E}_{\mu^{\pi}_{(\mathcal{S},\mathcal{A},c,P,\gamma,\mu)}} \left[\sum_{i=0}^{H-1} \gamma^i c(s_i, a_i) + \gamma^H V(s_H) \right] \tag{5.15}
$$

which represents the expected cost achieved when executing policy π on $(\mathcal{S}, \mathcal{A}, c, P, \gamma, \mu)$ using V as the terminal cost. We can substitute different policies, terminal costs and MDPs. For example, $\left\langle \hat{V}, \hat{\pi}, \hat{\mathcal{M}} \right\rangle_H$ is the expected cost obtained by running policy $\hat{\pi}$ on simulator $\hat{\mathcal{M}}$ for H steps with approximate learned terminal value function $\hat{V}$.

Lemma 7. *For a given policy π, the optimal value function $V^{\pi^*}_{\mathcal{M}}$ and MDPs $\mathcal{M}, \hat{\mathcal{M}}$ the following performance difference holds*

$$\left|\left|\left\langle V^{\pi^*}_{\mathcal{M}}, \pi, \mathcal{M}\right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi, \hat{\mathcal{M}}\right\rangle_H\right|\right|_\infty \leq$$
$$\gamma\left(\frac{1-\gamma^{H-1}}{1-\gamma}\right)\alpha H\left(\frac{c_{\max}-c_{\min}}{2}\right) + \gamma^H \alpha H\left(\frac{V_{\max}-V_{\min}}{2}\right) + \frac{1-\gamma^H}{1-\gamma}\alpha$$

Proof. We temporarily introduce a new MDP $\mathcal{M}'$ that has the same cost function as a $\mathcal{M}$, but transition function of $\hat{\mathcal{M}}$

$$\begin{aligned}\left\langle V^{\pi^*}_{\mathcal{M}}, \pi, \mathcal{M}\right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi, \hat{\mathcal{M}}\right\rangle_H &= \left\langle V^{\pi^*}_{\mathcal{M}}, \pi, \mathcal{M}\right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi, \mathcal{M}'\right\rangle_H \\ &+ \left\langle V^{\pi^*}_{\mathcal{M}}, \pi, \mathcal{M}'\right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi, \hat{\mathcal{M}}\right\rangle_H\end{aligned} \tag{5.16}$$

Let $\Delta P(s_0 \dots s_H) = P(s_0 \dots s_H) - \hat{P}(s_0 \dots s_H)$ represent the difference in distribution of states encountered by executing π on $(\mathcal{S}, \mathcal{A}, c, P, \gamma, \mu)$ and $\hat{\mathcal{M}}$ respectively starting from state s_0.

Expanding the RHS of (5.16)

$$= \sum_{s_0,\dots,s_H} \Delta P(s_0 \dots s_H)\left(\sum_{i=0}^{H-1}\gamma^i c(s_i,a_i) + \gamma^H V^{\pi^*}_{\mathcal{M}}(s_H)\right) + \mathbb{E}_{\mu^\pi_{\hat{\mathcal{M}}}}\left[\sum_{i=0}^{H-1}\gamma^i\left(c(s_i,a_i) - \hat{c}(s_i,a_i)\right)\right] \tag{5.17}$$

Since the first state s_1 is the same

$$\begin{aligned}&= \sum_{s_1,\dots,s_H} \Delta P(s_1 \dots s_H)\left(\sum_{i=1}^{H-1}\gamma^i c(s_i,a_i) + \gamma^H V^{\pi^*}_{\mathcal{M}}(s_H)\right) \\ &\qquad + \mathbb{E}_{\mu^\pi_{\hat{\mathcal{M}}}}\left[\sum_{i=0}^{H-1}\gamma^i\left(c(s_i,a_i) - \hat{c}(s_i,a_i)\right)\right] \\ &\leq \left|\left|\sum_{s_1,\dots,s_H} \Delta P(s_1 \dots s_H)\left(\sum_{i=1}^{H-1}\gamma^i c(s_i,a_i) + \gamma^H V^{\pi^*}_{\mathcal{M}}(s_H)\right)\right|\right|_\infty \\ &\qquad + \left|\left|\mathbb{E}_{\mu^\pi_{\hat{\mathcal{M}}}}\left[\sum_{i=0}^{H-1}\gamma^i\left(c(s_i,a_i) - \hat{c}(s_i,a_i)\right)\right]\right|\right|_\infty \\ &\leq \left|\left|\sum_{s_1,\dots,s_H} \Delta P(s_1 \dots s_H)\left(\sum_{i=1}^{H-1}\gamma^i c(s_i,a_i) + \gamma^H V^{\pi^*}_{\mathcal{M}}(s_H)\right)\right|\right|_\infty + \frac{1-\gamma^H}{1-\gamma}\alpha\end{aligned} \tag{5.18}$$

where the first inequality is obtained by applying the triangle inequality and the second

one is obtained by applying triangle inequality followed by the upper bound on the error in cost-function.

$$\leq \left|\left| \sum_{s_2,\ldots,s_{H+1}} \Delta P(s_2 \ldots s_{H+1}) \right|\right|_\infty \sup\left(\sum_{i=1}^{H-1} \gamma^i c(s_i, a_i) + \gamma^H V^{\pi^*}_{\mathcal{M}}(s_H) - K\right) + \frac{1-\gamma^H}{1-\gamma}\alpha \tag{5.19}$$

By choosing $K = \sum_{i=1}^{H-1} \gamma^i (c_{\max} + c_{\min})/2 + \gamma^H (V_{\max} + V_{\min})/2$ we can ensure that the term inside sup is upper-bounded by $\gamma(1-\gamma^{H-1})/(1-\gamma)((c_{\max} - c_{\min})/2) + \gamma^H (V_{\max} - V_{\min})/2$

$$\leq \gamma \left(\frac{1-\gamma^{H-1}}{1-\gamma}\right) \alpha H \left(\frac{c_{\max} - c_{\min}}{2}\right) + \gamma^H \alpha H \left(\frac{V_{\max} - V_{\min}}{2}\right) + \frac{1-\gamma^H}{1-\gamma}\alpha \tag{5.20}$$

□

The above lemma builds on similar results in [7, 228, 400].

We are now ready to prove our main theorem, i.e. the performance bound of an MPC policy that uses an approximate model and approximate value function.

Proof of Theorem 5

Proof. Since, $\hat{\pi}$ is the greedy policy when using $\hat{\mathcal{M}}$ and $\hat{V}$,

$$\begin{aligned} \left\langle \hat{V}, \hat{\pi}, \hat{\mathcal{M}} \right\rangle_H &\leq \left\langle \hat{V}, \pi^*, \hat{\mathcal{M}} \right\rangle_H \\ \left\langle V^{\pi^*}_{\mathcal{M}}, \hat{\pi}, \hat{\mathcal{M}} \right\rangle_H &\leq \left\langle V^{\pi^*}_{\mathcal{M}}, \pi^*, \hat{\mathcal{M}} \right\rangle_H + 2\gamma^H \epsilon \text{ (using } \left|\left| \hat{V} - V^{\pi^*}_{\mathcal{M}} \right|\right|_1 \leq \epsilon) \end{aligned} \tag{5.21}$$

Also, we have

$$\begin{aligned} \left\langle V^{\pi^*}_{\mathcal{M}}, \hat{\pi}, \mathcal{M} \right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi^*, \mathcal{M} \right\rangle_H &= \left(\left\langle V^{\pi^*}_{\mathcal{M}}, \hat{\pi}, \mathcal{M} \right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \hat{\pi}, \hat{\mathcal{M}} \right\rangle_H \right) \\ &\quad - \left(\left\langle V^{\pi^*}_{\mathcal{M}}, \pi^*, \mathcal{M} \right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi^*, \hat{\mathcal{M}} \right\rangle_H \right) \\ &\quad + \left(\left\langle V^{\pi^*}_{\mathcal{M}}, \hat{\pi}, \hat{\mathcal{M}} \right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi^*, \hat{\mathcal{M}} \right\rangle_H \right) \end{aligned} \tag{5.22}$$

The first two terms can be bounded using Lemma 7 and the third term using Eq. 5.21 to get

$$\begin{aligned}&\left\langle V^{\pi^*}_{\mathcal{M}}, \hat{\pi}, \mathcal{M} \right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi^*, \mathcal{M} \right\rangle_H \\ &\leq 2\left(\gamma \frac{1-\gamma^{H-1}}{1-\gamma}\alpha H\left(\frac{c_{\max}-c_{\min}}{2}\right) + \gamma^H \alpha H\left(\frac{V_{\max}-V_{\min}}{2}\right) + \frac{1-\gamma^H}{1-\gamma}\alpha + \gamma^H \epsilon\right)\end{aligned} \tag{5.23}$$

Now, let s be the state with max loss $V^{\hat{\pi}}_{\mathcal{M}}(s) - V^{\pi^*}_{\mathcal{M}}(s)$

$$\begin{aligned}&V^{\hat{\pi}}_{\mathcal{M}}(s) - V^{\pi^*}_{\mathcal{M}}(s) = \left\langle V^{\hat{\pi}}_{\mathcal{M}}, \hat{\pi}, \mathcal{M} \right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi^*, \mathcal{M} \right\rangle_H \\ &= \left(\left\langle V^{\hat{\pi}}_{\mathcal{M}}, \hat{\pi}, \mathcal{M} \right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \hat{\pi}, \mathcal{M} \right\rangle_H\right) + \left(\left\langle V^{\pi^*}_{\mathcal{M}}, \hat{\pi}, \mathcal{M} \right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi^*, \mathcal{M} \right\rangle_H\right) \\ &= \gamma^H \left(V^{\hat{\pi}}_{\mathcal{M}}(s_{H+1}) - V^{\pi^*}_{\mathcal{M}}(s_{H+1})\right) + \left(\left\langle V^{\pi^*}_{\mathcal{M}}, \hat{\pi}, \mathcal{M} \right\rangle_H - \left\langle V^{\pi^*}_{\mathcal{M}}, \pi^*, \mathcal{M} \right\rangle_H\right) \\ &\leq \gamma^H \left(V^{\hat{\pi}}_{\mathcal{M}}(s) - V^{\pi^*}_{\mathcal{M}}(s)\right) \\ &+ 2\left(\frac{\gamma(1-\gamma^{H-1})}{1-\gamma}\alpha H\left(\frac{c_{\max}-c_{\min}}{2}\right) + \gamma^H \alpha H\left(\frac{V_{\max}-V_{\min}}{2}\right) + \frac{1-\gamma^H}{1-\gamma}\alpha + \gamma^H \epsilon\right)\end{aligned} \tag{5.24}$$

where last inequality comes from applying Eq. equation 5.23 and the fact that s is the state with max loss. The final expression follows from simple algebraic manipulation. □

5.7.2 Experiment Details

Task Details

CartpoleSwingup

- Reward function: $x^2_{cart} + \theta^2_{pole} + 0.01 v_{cart} + 0.01\omega_{pole} + 0.01a^2$
- Observation: $[x_{cart}, \theta_{pole}, v_{cart}, \omega_{pole}]$ (4 dim)

SawyerPegInsertion We simulate sensor noise by placing a simulated position sensor at the target location in the MuJoCo physics engine that adds Gaussian noise with $\sigma = 10$cm to the observed 3D position vector. MPPI uses a deterministic model that does not take sensor noise into account for planning. Every episode lasts for 75 steps with a timestep of 0.02 seconds between steps

- Reward function: $-1.0 * ||x_{ee} - x_{target}||_1 - 5.0 * ||x_{ee} - x_{target||_2}$ $+5 * \mathbb{1}\left(||x_{ee} - x_{target}||_2 < 0.06\right)$

- Observation: $\left[q_{pos}, q_{vel}, x_{ee}, x_{target}, x_{ee} - x_{target}, ||x_{ee} - x_{target}||_1, ||x_{ee} - x_{target||_2}\right]$ (25 dim)

An episode is considered successful if the peg stays within the hole for atleast 5 steps.

InHandManipulation This environment was used without modification from the accompanying codebase for [384] and is available at https://bit.ly/3f6MNBP

- Reward function: $-||x_{obj} - x_{des}||_2 + z_{obj}^T z_{des}$+ Bonus for proximity to desired pos + orien, $z_{obj}^T z_{des}$ represents dot product between object axis and target axis to measure orientation similarity.

- Observation: $[q_{pos}, x_{obj}, v_{obj}, z_{obj}, z_{des}, x_{obj} - x_{des}, z_{obj} - z_{des}]$ (45 dim)

Every episode lasts 75 steps with a timestep of 0.01 seconds between steps. An episode is considered successful if the orientation of the pen stays within a specified range of desired orientation for atleast 20 steps. The orientation similarity is measured by the dot product between the pen's current longitudinal axis and desired with a threshold of 0.95.

Learning Details

Validation: Validation is performed after every N training episodes during training for N_{eval} episodes using a fixed set of start states that the environment is reset to. We ensure that the same start states are sampled at every validation iteration by setting the seed value to a pre-defined validation seed, which is kept constant across different runs of the algorithm with different training seeds. This helps ensure consistency in evaluating different runs of the algorithm. For all our experiments we set $N = 40$ and $N_{\text{eval}} = 30$.

MPQ(λ): For all tasks, we represent Q function using 2 layered fully-connected neural network with 100 units in each layer and ReLU activation. We use Adam [246] for optimization with a learning rate of 0.001 and discount factor $\gamma = 0.99$. The buffer size is 1500

Parameter	CartpoleSwingup	SawyerPegInsertion	InHandManipulation
Horizon	32	20	32
Num particles	60	100	100
Covariance (Σ)	0.45	0.25	0.3
Temperature(β)	0.1	0.1	0.15
Filter coeffs	[1.0, 0.0, 0.0]	[0.25, 0.8, 0.0]	[0.25, 0.8, 0.8]
Step size	1.0	0.9	1.0
γ	0.99	0.99	0.99

Table 5.1: MPPI Parameters

for CartpoleSwingup and 3000 for the others, with batch size of 64 for all. λ is smoothly decayed according to the following sublinear schedule

$$\lambda_t = \frac{\lambda_0}{1 + \kappa\sqrt{t}} \tag{5.25}$$

where the decay rate κ is calculated based on the desired final value of λ. For batch size we did a search from [16, 64] with a step size of 16 and buffer size was chosen from 1500, 3000, 5000. Batch size was tuned for cartpole and then fixed for the remaining two environments and buffer size was chosen independently.

Proximal Policy Optimization (PPO): Both policy and value functions are represented by feed forward networks with 2 layers each with 64 and 128 units for policy and value respectively. All other parameters are left to the default values. The number of trajectories collected per iteration is modified to correspond with the same number of samples collected between validation iterations for MPQ(λ). We collect 40 trajectories per iteration. Asymptotic performance is reported as average of last 10 validation iterations after 500 training iters in SawyerPegInsertion and 2k in InHandManipulation.

MPPI parameters MPPI parameters are reported in Table 5.1. In addition to the standard MPPI parameters, in certain cases we also use a step size as introduced by [494]. For InHandManipulation and SawyerPegInsertion we also apply autoregressive filtering to smoothen the sampled trajectories with tuned filter coefficients. This has been found

to be useful in prior works [312, 449] for high dimensional control tasks. The temperature, initial covariance and step size parameters were tuned using a grid search with true dynamics. Temperature and initial covariance were searched within $[0.0, 1.0]$ and step size from $[0.5, 1.0]$ with a discretization of 0.05. The number of particles were searched from $[40, 120]$ with a step size of 10 and the horizon was chosen from 4 different values $[16, 20, 32, 64]$. The best performing parameters were chosen based on average reward over 30 episodes with a fixed seed value to ensure reproducibility. The same parameters were then used in the case of biased dynamics and MPQ(λ), to clearly demonstrate that MPQ(λ) can overcome sources of error in the base MPC implementation.

Chapter 6

ADVERSARIAL MODEL-BASED OFFLINE REINFORCEMENT LEARNING

6.1 Introduction

The chapters so far have focused on planning and model-predictive control approaches that can enable robots to perform a variety of complex tasks. However, these methods require prior access to *accurate* and *fast* predictive models to succeed. This assumption can be violated in many real-world dynamic interactions such as fluid dynamical systems [491], contact-rich manipulation [19] or even modelling human behavior for autonomous driving. In such scenarios, we must rely on data-driven approaches for learning control policies. However, as discussed before, collecting robot experience is often challenging in terms of safety, time and money. Thus, data must be collected by qualified reference policies that satisfy certain safety and performance guarantees, such as human teleoperation or safe reference controllers. This poses severe challenges to learning based approaches based on RL, as the data collected by such policies is generally non-exploratory, sub-optimal and sparse. How can we develop practical methods for maximally utilizing such mixed-quality datasets? In this chapter, we explore this question and present a method to enable robust policy learning with strong performance guarantees.

We focus on the paradigm of offline reinforcement learning (ORL), which is a general framework for learning decision-making policies from logged datasets of interaction [224, 274, 291, 512]. In comparison with alternate learning techniques, such as off-policy RL and imitation learning (IL), offline RL reduces the data assumption needed to learn good policies and does not require collecting new data. Theoretically, offline RL can learn the best policy that the given data can explain: as long as the offline data includes the scenarios encountered by a near-optimal policy, an offline RL algorithm can learn such a near-optimal policy, even when the data is collected by highly sub-optimal policies and/or is not diverse.

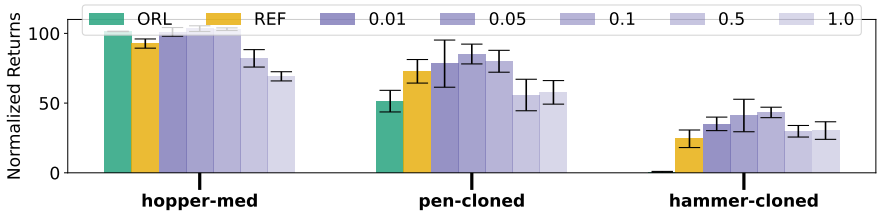


Figure 6.1: Robust Policy Improvement: ARMOR can improve performance over the reference policy (REF) over a broad range of pessimism hyperparameter (purple) regardless of data coverage. ORL denotes best offline RL policy without using the reference policy, and reference is obtained by behavior cloning on expert dataset.

Such robustness to data coverage makes offline RL a promising technique for solving real-world problems, as collecting diverse or expert-quality data in practice is often expensive or simply infeasible.

The fundamental principle behind offline RL is the concept of pessimism, which considers worst-case outcomes for scenarios without data. In algorithms, this is realized by (explicitly or implicitly) constructing performance lower bounds in policy learning which penalizes uncertain actions. Various designs have been proposed to construct such lower bounds, including behavior regularization [160, 162, 264, 277, 509], point-wise pessimism based on negative bonuses or truncation [224, 245], value penalty [265, 530], or two-player games [85, 481, 512]. Conceptually, the tighter the lower bound is, the better the learned policy would perform; see a detailed discussion of related work in Section 6.6.

Despite these advances, offline RL still has not been widely adopted to build learning-based decision systems beyond academic research. One important factor we posit is the issue of performance degradation: Usually, the systems we apply RL to have currently running policies, such as an engineered autonomous driving rule or a heuristic-based system for diagnosis, and the goal of applying a learning algorithm is often to further improve upon these baseline *reference policies*. As a result, it is imperative that the policy learned by the algorithm does not degrade the base performance. This criterion is especially critical for applications where poor decision outcomes cannot be tolerated.

However, running an offline RL algorithm based on pessimism, in general, is not free

from performance degradation. While there have been algorithms with policy improvement guarantees [85, 160, 162, 265, 277], such guarantees apply only to the behavior policy that collects the data, which might not necessarily be the reference policy. In fact, quite often these two policies are different. For example, in robotic manipulation, it is common to have a dataset of activities different from the target task. In such a scenario, comparing against the behavior policy is meaningless, as these policies do not have meaningful performance in the target task.

In this chapter, we propose a novel model-based offline RL framework, called Adversarial Model for Offline Rinforcement Learning (ARMOR), which can robustly learn policies that improve upon an arbitrary reference policy by adversarially training a Markov decision process (MDP) model, regardless of the data quality. ARMOR is designed based on the concept of relative pessimism [85], which aims to optimize for the worst-case relative performance over uncertainty. In theory, we prove that, owing to relative pessimism, the ARMOR policy never degrades the performance of the reference policy for a range of hyperparameters which is given beforehand, a property known as Robust Policy Improvement (RPI) [85]. In addition, when the right hyperparameter is chosen, and the reference policy is covered by the data, we prove that the ARMOR policy can also compete with any policy covered by the data in an absolute sense. To our knowledge, RPI property of offline RL has so far been limited to comparing against the data collection policy [85, 160, 162, 264, 277, 509]. In ARMOR, by adversarially training an MDP model, we extend the technique of relative pessimism to achieve RPI with *arbitrary* reference policies, regardless of whether they collected the data or not (Fig. 6.1). In addition to theory, we design a scalable deep-learning implementation of ARMOR to validate these claims that jointly trains an MDP model and the state-action value function to minimize the estimated performance difference between the policy and the reference using model-based rollouts. Our implementation achieves state-of-the-art (SoTA) performance on D4RL benchmarks [155], while using only a *single* model (in contrast to ensembles used in existing model-based offline RL works). This makes ARMOR a better framework for using high-capacity world models (e.g.[177]) for which building an ensemble is too expensive. We also empirically validate the RPI property of our implementation.

6.2 Preliminaries

We assume that the learner has access to a Markovian policy class Π and an MDP model class $\mathcal{M}$.

Assumption 1 (Realizability)**.** *We assume the ground truth model $M^\star$ is in the model class $\mathcal{M}$.*

In addition, we assume that we are provided a reference policy π_{ref}. In practice, such a reference policy represents a baseline whose performance we want to improve with offline RL and data.

Assumption 2 (Reference policy)**.** *We assume access to a reference policy π_{ref}, which can be queried at any state. We assume π_{ref} is realizable, i.e., $\pi_{\mathsf{ref}} \in \Pi$.*

If π_{ref} is not provided, we can still run ARMOR as a typical offline RL algorithm, by first performing behavior cloning on the data and setting the cloned policy as π_{ref}. In this case, ARMOR has RPI with respect to the behavior policy.

Robust Policy Improvement RPI is a notion introduced in [85], which means that the offline algorithm can learn to improve over the behavior policy, using hyperparameters within a known set. Algorithms with RPI are more robust to hyperparameter choices, and they are often derived from the principle of relative pessimism [85]. In this work, we extend the RPI concept to compare with an arbitrary reference (or baseline) policy, which can be different from the behavior policy and can take actions outside data support.

6.3 Adversarial Model for Offline Reinforcement Learning

ARMOR is a model-based offline RL algorithm designed with relative pessimism. The goal of ARMOR is to find a policy $\widehat{\pi}$ that maximizes the performance difference $J(\widehat{\pi}) - J(\pi_{\mathsf{ref}})$ to a given reference policy π_{ref}, while accounting for the uncertainty due to limited data coverage. ARMOR achieves this by solving a two-player game between a learner policy and an adversary MDP model:

$$\widehat{\pi} = \arg\max_{\pi \in \Pi} \min_{M \in \mathcal{M}_\alpha} J_M(\pi) - J_M(\pi_{\mathsf{ref}}) \tag{6.1}$$

based on a version space of MDP models

$$\mathcal{M}_\alpha = \{M \in \mathcal{M} : \mathcal{E}_\mathcal{D}(M) - \min_{M' \in \mathcal{M}} \mathcal{E}_\mathcal{D}(M') \leq \alpha\}, \tag{6.2}$$

where we define the model fitting loss as

$$\mathcal{E}_\mathcal{D}(M) \coloneqq -\textstyle\sum_\mathcal{D} \log P_M(s' \mid s, a) + {}^{(R_M(s,a)-r)^2}/_{V_{\max}^2} \tag{6.3}$$

and $\alpha \geq 0$ is a bound on statistical errors such that $M^\star \in \mathcal{M}_\alpha$. In this two-player game, ARMOR is optimizing a lower bound of the relative performance $J(\pi) - J(\pi_{\mathsf{ref}})$. This is due to the construction that $M^\star \in \mathcal{M}_\alpha$, which ensures $\min_{M \in \mathcal{M}_\alpha} J_M(\pi) - J_M(\pi_{\mathsf{ref}}) \leq J_{M^\star}(\pi) - J_{M^\star}(\pi_{\mathsf{ref}})$.

One interesting property that follows from optimizing the relative performance lower bound is that $\widehat{\pi}$ is guaranteed to always be no worse than π_{ref}, for a wide range of α and regardless of the relationship between π_{ref} and the data $\mathcal{D}$.

Proposition 8. *For any α large enough such that $M^\star \in \mathcal{M}_\alpha$, it holds that $J(\widehat{\pi}) \geq J(\pi_{\mathsf{ref}})$.*

This fact can be easily reasoned: Since $\pi_{\mathsf{ref}} \in \Pi$, we have $\max_{\pi \in \Pi} \min_{M \in \mathcal{M}_\alpha} J_M(\pi) - J_M(\pi_{\mathsf{ref}}) \geq \min_{M \in \mathcal{M}_\alpha} J_M(\pi_{\mathsf{ref}}) - J_M(\pi_{\mathsf{ref}}) = 0$. In other words, ARMOR achieves the RPI property with respect to any reference policy π_{ref} and offline dataset $\mathcal{D}$.

This RPI property of ARMOR is stronger than the RPI property in the literature. In comparison, previous algorithms with RPI [85, 160, 162, 264, 277, 509] are only guaranteed to be no worse than the behavior policy that collected the data. In Section 6.3.2, we will also show that when α is set appropriately, ARMOR can provably compete with the best data covered policy as well, as prior offline RL works [e.g., 85, 481, 512].

6.3.1 An Illustrative Toy Example

Why does ARMOR have the RPI property, even when the reference policy π_{ref} is not covered by the data $\mathcal{D}$? While we will give a formal analysis soon in Section 6.3.2, here we provide some intuitions as to why this is possible. First, notice that ARMOR has access to the reference policy π_{ref}. Therefore, a trivial way to achieve RPI with respect to π_{ref} is to just output π_{ref}. However, this naïve algorithm while never degrading π_{ref} cannot learn to improve from π_{ref}. ARMOR achieves these two features simultaneously by *1)* learning

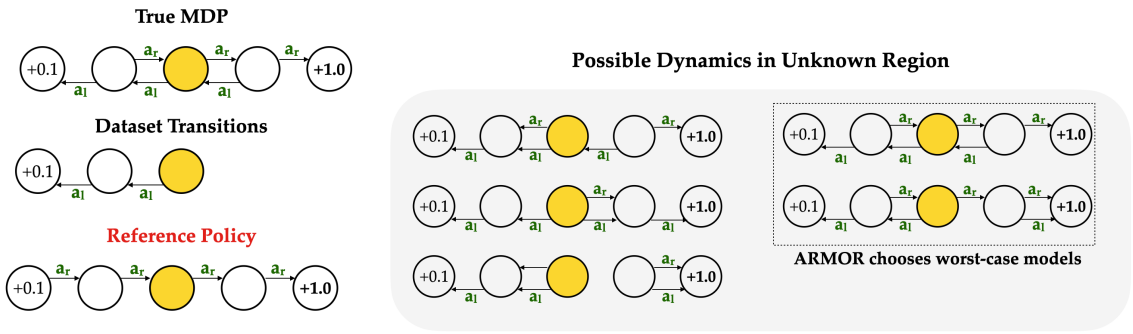


Figure 6.2: A toy MDP illustrating the RPI property of ARMOR. (Top) The true MDP has deterministic dynamics where taking the left (a_l) or right (a_r) actions takes the agent to corresponding states; start state is in yellow. The suboptimal behavior policy visits only the left part of the state space, and the reference policy demonstrates optimal behavior by always choosing a_r. (Bottom) A subset of possible data-consistent MDP models in the version space. The adversary always chooses the MDP that makes the reference maximally outperform the learner. In response, the learner will learn to mimic the reference outside data support to be competitive.

an MDP Model, and *2)* adversarially training this MDP model to minimize the relative performance difference to π_{ref} during policy optimization.

We illustrate this by a one-dimensional discrete MDP example with five possible states as shown in Figure 6.2. The dynamic is deterministic, and the agent always starts in the center cell. The agent receives a lower reward of 0.1 in the left-most state and a high reward of 1.0 upon visiting the right-most state. Say, the agent only has access to a dataset from a sub-optimal policy that always takes the left action to receive the 0.1 reward. Further, let's say we have access to a reference policy that demonstrates optimal behavior on the true MDP by always visiting the right-most state. However, it is unknown a priori that the reference policy is optimal. In such a case, typical offline RL methods can only recover the sub-optimal policy from the dataset as it is the best-covered policy in the data.

ARMOR can learn to recover the expert reference policy in this example by performing rollouts with the adversarially trained MDP model. From the realizability assumption (Assumption 1), we know that the version space of models contains the true model (i.e., $M^\star \in \mathcal{M}_\alpha$). The adversary can then choose a model from this version space where the refer-

ence policy π_{ref} maximally outperforms the learner. In this toy example, the model selected by the adversary would be the one allowing the expert policy to reach the right-most state. Now, optimizing relative performance difference with respect to this model will ensure that the learner can recover the expert behavior, since the only way for the learner to stay competitive with the reference policy is to mimic the reference policy in the region outside data support. In other words, the reason why ARMOR has RPI to π_{ref} is that **its adversarial model training procedure can augment the original offline data with new states and actions that would cover those generated by running the reference policy**. [1]

6.3.2 Theoretical Analysis

Now we make the above discussions formal and give theoretical guarantees on ARMOR's absolute performance and RPI property. To this end, we introduce a single-policy concentrability coefficient, which measures the distribution shift between a policy π and the data distribution μ.

Definition 1 (Generalized Single-policy Concentrability)**.** *We define the generalized single-policy concentrability for policy π, model class $\mathcal{M}$ and offline data distribution μ as $\mathfrak{C}_{\mathcal{M}}(\pi) \coloneqq \sup_{M\in\mathcal{M}} \frac{\mathbb{E}_{d^\pi}[\mathcal{E}^\star(M)]}{\mathbb{E}_{\mu}[\mathcal{E}^\star(M)]}$, where $\mathcal{E}^\star(M) = D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2 + (R_M(s,a) - R^\star(s,a))^2 / V_{\max}^2$.*

Note that $\mathfrak{C}_{\mathcal{M}}(\pi)$ is always upper bounded by the standard single-policy concentrability coefficient $\|d^\pi/\mu\|_\infty$ [e.g., 224, 392, 515], but it can be smaller in general with model class $\mathcal{M}$. It can also be viewed as a model-based analog of the one in [512]. A detailed discussion around $\mathfrak{C}_{\mathcal{M}}(\pi)$ can be found in [481].

First, we present the absolute performance guarantee of ARMOR, which holds for a well-tuned α.

Theorem 9 (Absolute performance)**.** *Under Assumption 1, there is an absolute constant c such that for any $\delta \in (0,1]$, if we set $\alpha = c \cdot (\log(|\mathcal{M}|/\delta))$ in Equation 6.2, then for any reference policy π_{ref} and comparator policy $\pi^\dagger \in \Pi$, with probability $1-\delta$, the policy $\widehat{\pi}$ learned*

[1] Note that ARMOR does not depend on knowledge of the true reward function and similar arguments hold in the case of learned rewards.

by ARMOR in Equation 6.1 satisfies that $J(\pi^\dagger) - J(\widehat{\pi})$ *is upper bounded by*

$$\mathcal{O}\left(\left(\sqrt{\mathfrak{C}_{\mathcal{M}}(\pi^\dagger)} + \sqrt{\mathfrak{C}_{\mathcal{M}}(\pi_{\mathsf{ref}})}\right) \frac{V_{\max}}{1-\gamma} \sqrt{\frac{\log(|\mathcal{M}|/\delta)}{n}}\right).$$

Roughly speaking, Theorem 9 shows that $\widehat{\pi}$ learned by ARMOR can compete with any policy $\pi^\dagger$ with a large enough dataset, as long as the offline data μ has good coverage on $\pi^\dagger$ (good coverage over π_{ref} can be automatically satisfied if we simply choose $\pi_{\mathsf{ref}} = \mu$, which yields $\mathfrak{C}_{\mathcal{M}}(\pi_{\mathsf{ref}}) = 1$). Compared to the closest model-based offline RL work [481], if we set $\pi_{\mathsf{ref}} = \mu$ (data collection policy), Theorem 9 leads to almost the same guarantee as [481, Theorem 1] up to constant factors.

In addition to absolute performance, below we show that, under Assumptions 1 and 2, ARMOR has the RPI property to π_{ref}: it always improves over $J(\pi_{\mathsf{ref}})$ for *a wide range of parameter* α. Compared with the model-free ATAC algorithm in [85, Proposition 6], the threshold for α in Theorem 10 does not depend on sample size N due to the model-based nature of ARMOR.

Theorem 10 (Robust strong policy improvement)**.** *Under Assumptions 1 and 2, there exists an absolute constant* c *such that for any* $\delta \in (0, 1]$*, if: i)* $\alpha \geq c \cdot (\log(|\mathcal{M}|/\delta))$ *in Equation 6.2; ii)* $\pi_{\mathsf{ref}} \in \Pi$*, then with probability* $1 - \delta$*, the policy* $\widehat{\pi}$ *learned by ARMOR in Equation 6.1 satisfies* $J(\widehat{\pi}) \geq J(\pi_{\mathsf{ref}})$*.*

The detailed proofs of Theorems 9 and 10, as well as the discussion on how to relax Assumptions 1 and 2 to the misspecified model and policy classes are deferred to Section 6.9.1.

6.4 Practical Implementation

In this section, we present a scalable implementation of ARMOR (Algorithm 7) that approximately solves the two-player game in Equation 6.1. We first describe the overall design principle and then the algorithmic details.

6.4.1 A Model-based Actor Critic Approach

For computational efficiency, we take a model-based actor critic approach and solve a regularized version of Equation 6.1. We construct this regularized version by relaxing the

Algorithm 7: ARMOR (Adversarial Model for Offline Reinforcement Learning)

Input: Batch data $\mathcal{D}_{\text{real}}$, policy π, MDP model M, critics f_1, f_2, horizon H, constants $\beta, \lambda \geq 0$, $\tau \in [0,1]$, $w \in [0,1]$,

Initialize target networks $\bar{f}_1 \leftarrow f_1$, $\bar{f}_2 \leftarrow f_2$ and $\mathcal{D}_{\text{model}} = \emptyset$ **for** $k = 0, \ldots, K-1$ **do**

Sample minibatch $\mathcal{D}_{\text{real}}^{\text{mini}}$ from dataset $\mathcal{D}_{\text{real}}$ and minibatch $\mathcal{D}_{\text{model}}^{\text{mini}}$ from dataset $\mathcal{D}_{\text{model}}$.

Construct transition tuples using model predictions

$$\mathcal{D}_M \coloneqq \left\{(s, a, r_M, s'_M) : r_M = R_M(s,a), s'_M \sim P_M(\cdot \mid s, a), (s,a) \in \mathcal{D}_{\text{real}}^{\text{mini}} \cup \mathcal{D}_{\text{model}}^{\text{mini}}\right\}$$

Update the adversary networks; for $i = 1, 2$,

$$l^{\text{adversary}}(f, M) \coloneqq \mathcal{L}_{\mathcal{D}_M}(f, \pi, \pi_{\mathsf{ref}}) + \beta \left(\mathcal{E}^w_{\mathcal{D}_M}(f, M, \pi) + \lambda \mathcal{E}_{\mathcal{D}_{\text{real}}^{\text{mini}}}(M)\right) \tag{6.4}$$

$$M \leftarrow M - \eta_{\text{fast}} \left(\nabla_M l^{\text{adversary}}(f_1, M) + \nabla_M l^{\text{adversary}}(f_2, M)\right)$$

$$f_i \leftarrow \text{Proj}_{\mathcal{F}}(f_i - \eta_{\text{fast}} \nabla_{f_i} l^{\text{adversary}}(f_i, M)) \quad \text{and} \quad \bar{f}_i \leftarrow (1-\tau)\bar{f}_i + \tau f_i$$

Update actor network with respect to the first critic network and the reference policy

$$l^{\text{actor}}(\pi) \coloneqq -\mathcal{L}_{\mathcal{D}_M}(f_1, \pi, \pi_{\mathsf{ref}}) \tag{6.5}$$

$$\pi \leftarrow \text{Proj}_{\Pi}(\pi - \eta_{\text{slow}} \nabla_\pi l^{\text{actor}}(\pi))$$

If $k\%H = 0$, then reset model state: $\bar{S}_\pi \leftarrow \{s \in \mathcal{D}_{\text{real}}^{\text{mini}}\}$ and $\bar{S}_{\pi_{\mathsf{ref}}} \leftarrow \{s \in \mathcal{D}_{\text{real}}^{\text{mini}}\}$

Query the MDP model to expand $\mathcal{D}_{\text{model}}$ and update model state

$$\bar{A}_\pi \coloneqq \{a : a \sim \pi(s), s \in \bar{S}_\pi\} \quad \text{and} \quad \bar{A}_{\pi_{\mathsf{ref}}} \coloneqq \{a : a \sim \pi_{\mathsf{ref}}(s), s \in \bar{S}_{\pi_{\mathsf{ref}}}\}$$

$$\mathcal{D}_{\text{model}} \coloneqq \mathcal{D}_{\text{model}} \cup \{\bar{S}_\pi, \bar{A}_\pi\} \cup \{\bar{S}_{\pi_{\mathsf{ref}}}, \bar{A}_{\pi_{\mathsf{ref}}}\}$$

$$\bar{S}_\pi \leftarrow \{s' \mid s' \sim \texttt{detach}(P_M(\cdot \mid s, a)), s \in \bar{S}_\pi, a \in \bar{A}_\pi\}$$

$$\bar{S}_{\pi_{\mathsf{ref}}} \leftarrow \{s' \mid s' \sim \texttt{detach}(P_M(\cdot \mid s, a)), s \in \bar{S}_{\pi_{\mathsf{ref}}}, a \in \bar{A}_{\pi_{\mathsf{ref}}}\}$$

constraint $M \in \mathcal{M}_\alpha$ in the inner minimization of Equation 6.1 to a regularization term and introducing an additional critic function. To clearly elaborate this, we first present the regularized objective in its complete form, and subsequently derive it from Equation 6.1.

Let $\mathcal{F} : \{f : \mathcal{S} \times \mathcal{A} \to [0, V_{\max}]\}$ be a class of critic functions. The regularized objective is given as

$$\tilde{\pi} \in \arg\max_{\pi \in \Pi} \mathcal{L}_{d_M^{\pi_{\text{ref}}}}(\pi, f) \tag{6.6}$$
$$\text{s.t. } f^\pi \in \arg\min_{M \in \mathcal{M}, f \in \mathcal{F}} \mathcal{L}_{d_M^{\pi_{\text{ref}}}}(\pi, f) + \beta \left(\mathcal{E}_{\rho_{\pi_{\text{ref}}, \pi}}(\pi, f, M) + \lambda \mathcal{E}_{\mathcal{D}}(M) \right)$$

where $\mathcal{E}_{\mathcal{D}}(M) = \sum_{\mathcal{D}} - \log P_M(s' \mid s, a) + {}^{(R_M(s,a)-r)^2}/_{V_{\max}^2}$ is the model-fitting error, $\mathcal{L}_{d_M^{\pi_{\text{ref}}}}(\pi, f) \coloneqq \mathbb{E}_{d_M^{\pi_{\text{ref}}}}[f(s, \pi) - f(s, \pi_{\text{ref}})]$ is equal to the performance difference $(1-\gamma)(J_M(\pi) - J_M(\pi_{\text{ref}}))$, $\mathcal{E}_{\rho_{\pi_{\text{ref}}, \pi}}(\pi, f, M)$ denotes the squared Bellman error on the distribution $\rho_{\pi_{\text{ref}}, \pi}$ that denotes the distribution generated by first running π_{ref} and then rolling out π in M (with a switching time sampled from a geometric distribution of γ), and β, λ act as the Lagrange multipliers.

This regularized formulation in Equation 6.6 can be derived as follows. Assuming $Q_M^\pi \in \mathcal{F}$, and using the facts that $J_M(\pi) = \mathbb{E}_{d_0}[Q_M^\pi(s, \pi)]$ and the Bellman equation $Q_M^\pi(s, a) = r_M(s, a) + \gamma \mathbb{E}_{s' \sim P_M(s,a)}[Q_M^\pi(s', \pi)]$, we can rewrite Equation 6.1 as

$$\max_{\pi \in \Pi} \min_{M \in \mathcal{M}, f \in \mathcal{F}} \quad \mathbb{E}_{d_M^{\pi_{\text{ref}}}}[f(s, \pi) - f(s, \pi_{\text{ref}})]. \tag{6.7}$$
$$\text{s.t.} \quad \mathcal{E}_{\mathcal{D}}(M) \leq \alpha + \min_{M' \in \mathcal{M}} \mathcal{E}_{\mathcal{D}}(M')$$
$$\forall s, a \in \text{supp}(\rho_{\pi_{\text{ref}}, \pi}), \quad f(s, a) = r_M(s, a) + \gamma \mathbb{E}_{s' \sim P_M(s,a)}[f(s', \pi)]$$

We then convert the constraints in Equation 6.7 into regularization terms in the inner minimization by introducing Lagrange multipliers (β, λ), following [85, 512], and drop the constants not affected by M, f, π, which results in Equation 6.6.

6.4.2 Algorithm Details

Algorithm 7 is an iterative solver for approximating the solution to Equation 6.6. Here we further approximate $d_M^{\pi_{\text{ref}}}$ and $\rho_{\pi_{\text{ref}}, \pi}$ in Eq. (6.6) using samples from the state-action buffer $\mathcal{D}_{\text{model}}$. We want ensure that $\mathcal{D}_{\text{model}}$ has a larger coverage than both $d_M^{\pi_{\text{ref}}}$ and $\rho_{\pi_{\text{ref}}, \pi}$. We do so heuristically, by constructing the model replay buffer $\mathcal{D}_{\text{model}}$ through repeatedly rolling

out π and π_{ref} with the adversarially trained MDP model M, such that $\mathcal{D}_{\text{model}}$ contains a diverse training set of state-action tuples.

Specifically, the algorithm takes as input an offline dataset $\mathcal{D}_{\text{real}}$, a policy π, an MDP model M and two critic networks f_1, f_2. At every iteration, the algorithm proceeds in two stages. First, the adversary is optimized to find a data-consistent model that minimizes the performance difference with the reference policy. We sample mini-batches of only states and actions $\mathcal{D}^{\text{mini}}_{\text{real}}$ and $\mathcal{D}^{\text{mini}}_{\text{model}}$ from the real and model-generated datasets respectively (Line 3). The MDP model M is queried on these mini-batches to generate next-state and reward predictions. The adversary then updates the model and Q-functions (Line 4) using the gradient of the loss described in Equation 6.4, where

$$\mathcal{L}_{\mathcal{D}_M}(f, \pi, \pi_{\mathsf{ref}}) \coloneqq \mathbb{E}_{\mathcal{D}_M}[f(s, \pi(s)) - f(s, \pi_{\mathsf{ref}}(s)]$$
$$\mathcal{E}^{w}_{\mathcal{D}_M}(f, M, \pi) \coloneqq (1-w)\mathcal{E}^{td}_{\mathcal{D}}(f, f, M, \pi) + w\mathcal{E}^{td}_{\mathcal{D}}(f, \bar{f}, M, \pi)$$
$$\mathcal{E}_{\mathcal{D}^{\text{mini}}_{\text{real}}}(M) \coloneqq \mathbb{E}_{\mathcal{D}^{\text{mini}}_{\text{real}}}[-\log P_M(s' \mid s, a) + {}^{(R_M(s,a)-r)^2}/_{V^2_{\max}}]$$

$\mathcal{L}_{\mathcal{D}_M}$ is the pessimistic loss term that forces the f to predict a lower value for the learner than the reference on the sampled states. $\mathcal{E}^{w}_{\mathcal{D}_M}$ is the Bellman surrogate to encourage the Q-functions to be consistent with the model-generated data $\mathcal{D}_M$. We use the double Q residual algorithm loss similar to [85], which is defined as a convex combination of the temporal difference losses with respect to the critic and the delayed target networks, $\mathcal{E}^{td}_{\mathcal{D}}(f, f', M, \pi) := \mathbb{E}_{\mathcal{D}}\left[(f(s,a) - r - \gamma f'(s', \pi))^2\right]$. $\mathcal{E}_{\mathcal{D}}(M)$ is the model-fitting loss that ensures the model is data-consistent. β and λ control the effect of the pessimistic loss, by constraining Q-functions and models the adversary can choose. Once the adversary is updated, we update the policy (Line 5) to maximize the pessimistic loss as defined in Eq. (6.5). Similar to [85], we choose one Q-function and a slower learning rate for the policy updates ($\eta_{\text{fast}} \gg \eta_{\text{slow}}$).

We remark that $\mathcal{E}^{w}_{\mathcal{D}_M}$ not only affects f_1, f_2, but also M, i.e., it forces the model to generate transitions where the Q-function is Bellman consistent. This allows the pessimistic loss to indirectly affect the model learning, thus making the model adversarial. Consider the special case where $\lambda = 0$ in the loss of Line 3. The model here is no longer forced to be data consistent, and the adversary can now freely update the model via $\mathcal{E}^{w}_{\mathcal{D}_M}$ such that the Q-function is always Bellman consistent. As a consequence, the algorithm becomes

equivalent to IL on the model-generated states. We empirically study this behavior in our experiments (Section 6.5).

Lines 6 and 7 describe our model-based rollout procedure. We incrementally rollout both π and π_{ref} from states in $\mathcal{D}^{\text{mini}}_{\text{real}}$ for a horizon H, and add the generated transitions to $\mathcal{D}_{\text{model}}$. The aim of this strategy is to generate a distribution with large coverage for training the adversary and policy, and we discuss this in detail in the next section.

Finally, it is important to note the fact that neither the pessimistic nor the Bellman surrogate losses uses the real transitions; hence our algorithm is completely model-based from a statistical point of view, that the value function f is solely an intermediate variable that helps in-model optimization and not directly fit from data.

6.5 Experiments

We test the efficacy of ARMOR on two major fronts: (1) performance comparison to existing offline RL algorithms, and (2) robust policy improvement over a reference policy that is not covered by the dataset, a novel setting that is not applicable to existing works[2]. We use the D4RL [155] continuous control benchmarks datasets for all our experiments and the code will be made public.

Experimental Setup: We parameterize π, f_1, f_2 and M using feedforward neural networks, and set $\eta_{\text{fast}} = 5e-4$, $\eta_{\text{slow}} = 5e-7$, $w = 0.5$ similar to [85]. In all our experiments, we vary only the β and λ parameters which control the amount of pessimism; others are fixed. Importantly, we set the rollout horizon to be the max episode horizon defined in the environment. The dynamics model is pre-trained for 100k steps using model-fitting loss on the offline dataset. ARMOR is then trained for 1M steps on each dataset. Refer to Section 6.9.7 for more details.

6.5.1 Comparison with Offline RL Baselines

By setting the reference policy to the behavior-cloned policy on the offline dataset, we can use ARMOR as a standard offline RL algorithm. Table 6.1 shows a comparison of

[2]In Section 6.9.7 we empirically show how imitation learning can be obtained as a special case of ARMOR

Dataset	ARMOR	MoREL	MOPO	RAMBO	COMBO	ATAC	CQL	IQL	BC
hopper-med	**101.4**	**95.4**	28.0	**92.8**	**97.2**	85.6	86.6	66.3	29.0
walker2d-med	**90.7**	77.8	17.8	**86.9**	**81.9**	**89.6**	74.5	78.3	6.6
halfcheetah-med	54.2	42.1	42.3	**77.6**	54.2	53.3	44.4	47.4	36.1
hopper-med-replay	**97.1**	**93.6**	67.5	**96.6**	89.5	**102.5**	48.6	**94.7**	11.8
walker2d-med-replay	**85.6**	49.8	39.0	**85.0**	56.0	**92.5**	32.6	73.9	11.3
halfcheetah-med-replay	50.5	40.2	53.1	**68.9**	55.1	48.0	46.2	44.2	38.4
hopper-med-exp	**103.4**	**108.7**	23.7	83.3	**111.1**	**111.9**	**111.0**	91.5	**111.9**
walker2d-med-exp	**112.2**	95.6	44.6	68.3	**103.3**	**114.2**	98.7	**109.6**	6.4
halfcheetah-med-exp	**93.5**	53.3	63.3	**93.7**	**90.0**	**94.8**	62.4	**86.7**	35.8
pen-human	**72.8**	-	-	-	-	53.1	37.5	**71.5**	34.4
hammer-human	1.9	-	-	-	-	1.5	**4.4**	1.4	1.5
door-human	6.3	-	-	-	-	2.5	**9.9**	4.3	0.5
relocate-human	**0.4**	-	-	-	-	0.1	0.2	0.1	0.0
pen-cloned	**51.4**	-	-	-	-	43.7	39.2	37.3	**56.9**
hammer-cloned	0.7	-	-	-	-	1.1	**2.1**	**2.1**	0.8
door-cloned	-0.1	-	-	-	-	**3.7**	0.4	1.6	-0.1
relocate-cloned	-0.0	-	-	-	-	**0.2**	-0.1	-0.2	-0.1
pen-exp	112.2	-	-	-	-	**136.2**	107.0	-	85.1
hammer-exp	**118.8**	-	-	-	-	**126.9**	86.7	-	**125.6**
door-exp	**98.7**	-	-	-	-	**99.3**	**101.5**	-	34.9
relocate-exp	**96.0**	-	-	-	-	**99.4**	**95.0**	-	**101.3**

Table 6.1: Performance comparison of ARMOR against baselines on the D4RL datasets. The values for ARMOR denote last iteration performance averaged over 4 random seeds, and baseline values were taken from their respective papers. The values denote normalized returns based on random and expert policy returns similar to [155]. Boldface denotes performance within 10% of the best performing algorithm. We report results with standard deviations in Section 6.9.7.

the performance of ARMOR against SoTA model-free and model-based offline RL baselines. In the former category, we consider ATAC [85], CQL [265] and IQL [255], and for the latter we consider MoREL [245], MOPO [530], and RAMBO [398]. We also compare against COMBO [529] which is a hybrid model-free and model-based algorithm. In these experiments, we initially warm start the optimization for 100k steps, by training the policy and Q-function using behavior cloning and temporal difference learning respectively on the offline dataset to ensure the learner policy is initialized to be the same as the reference. Overall, we observe that ARMOR consistently outperforms or is competitive with the best baseline algorithm on most datasets. Specifically, compared to other purely model-based baselines (MoREL, MOPO and RAMBO), there is a marked increase in performance in the *walker2d-med, hopper-med-exp* and *walker2d-med-exp* datasets. We would like to highlight two crucial elements about ARMOR, in contrast to other model-based baselines - (1) ARMOR achieves SoTA performance using only a *single* neural network to model the MDP, as opposed to complex network ensembles employed in previous model-based offline RL methods [245, 398, 529, 530], and (2) to the best of our knowledge, ARMOR is the only purely model-based offline RL algorithm that has shown performance comparable with model-free algorithms on the high-dimensional Adroit environments. The lower performance compared to RAMBO on *halfcheetah-med* and *halfcheetah-med-replay* may be attributed to that the much larger computational budget used by RAMBO is required for convergence on these datasets.

6.5.2 Robust Policy Improvement

Next, we test whether the practical version of ARMOR demonstrates RPI of the theoretical version. We consider a set of 14 datasets comprised of the *medium* and *medium-replay* versions of D4RL locomotion tasks, as well as the *human* and *cloned* versions of the Adroit tasks, with the reference policy set to be the stochastic behavior cloned policy on the expert dataset. We chose these combinations of dataset quality and reference, to ensure that the reference policy takes out-of-distribution actions with respect to the data. Unlike Sec. 6.5.1

[3]The variation in performance of the reference for different dataset qualities in the same environment is owing to different random seeds.

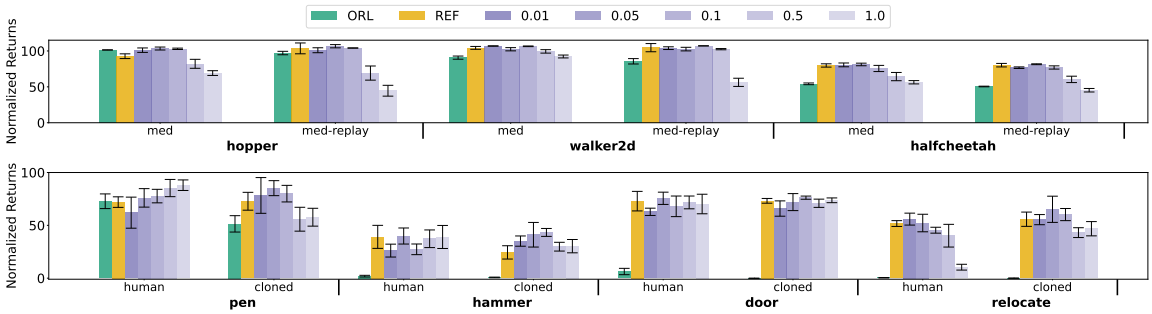


Figure 6.3: Verification of RPI over the reference policy for different β (purple). ORL denotes the performance of offline RL with ARMOR (Table 6.1), and REF is the performance of reference policy.[3]

here the reference policy is a black-box given as a part of the problem definition. This opens the question of how the learner should be initialized, since we can not trivially initialize the learner to be the reference as in the previous experiments.[4] In a similar spirit to Sec. 6.5.1, one might consider initializing the learner close to the reference by behavior cloning the reference policy on the provided dataset during warmstart, i.e, by replacing the dataset actions with reference actions. However, when the reference chooses out of support actions, this procedure will not provide a good global approximation of the reference policy, which can make the optimization problem harder. Instead, we propose to learn a residual policy where the learned policy outputs an additive correction to the reference [434]. This is an appropriate choice since ARMOR does not make any restrictive assumptions about the structure of the policy class. Figure 6.3 shows the normalized return achieved by ARMOR for different β, with fixed values for remaining hyperparameters. We observe that ARMOR is able to achieve performance comparable or better than the reference policy for a range of β values uniformly across all datasets, thus verifying the RPI property in practice. Specifically, there is significant improvement via RPI in the *hammer*, *door* and *relocate* domains, where running ARMOR as a pure offline RL algorithm(Section 6.5.1) does not show any progress [5]. Overall, we note the following metrics:

[4]In Section 6.9.7 we provide further experiments for different choices of reference policies.

[5]We provide comparisons when using a behavior cloning initialization for the learner in Section 6.9.7.

- In 14/14 datasets, ARMOR shows RPI (i.e., ARMOR policy is no worse than the reference when measured by overlap of confidence intervals). Further, considering the difference between ORL and REF as a rough indication of whether the reference is within data support, we note that in 12/14 cases REF is strictly better than ORL, and in all those cases ARMOR demonstrates RPI.

- In 5/14 datasets, the ARMOR policy is strictly better than the reference. (Criterion: the lower confidence of ARMOR performance is better than upper confidence of REF). It is important to note that this metric is highly dependent on the quality of the reference policy. Since the reference is near-expert, it can be hard for some environments to improve significantly over it.

6.6 Related Work

There has been an extensive line of works on reinforcement with offline/batch data, especially for the case with the data distribution is rich enough to capture the state-action distribution for any given policy [26, 75, 146, 274, 310, 339, 340, 513, 514]. However, this assumption is not practical since the data distribution is typically restricted by factors such as the quality of available policies, safety concerns, and existing system constraints, leading to narrower coverage. As a result, recent offline RL works in both theoretical and empirical literature have focused on systematically addressing datasets with inadequate coverage.

Modern offline reinforcement learning approaches can be broadly categorized into two groups for the purpose of learning with partial coverage. The first type of approaches rely on behavior regularization, where the learned policy is encouraged to be close to the behavior policy in states where there is insufficient data [e.g., 161, 264, 277, 429]. These algorithms ensure that the learned policy performs at least as well as the behavior policy while striving to improve it when possible, providing a form of safe policy improvement guarantees. These and other studies [160, 255, 509] have provided compelling empirical evidence for the benefits of these approaches.

The second category of approaches that has gained prevalence relies on the concept of *pessimism under uncertainty* to construct lower-bounds on policy performance without

explicitly constraining the policy. Recently, there have been several model-free and model-based algorithms based on this concept that have shown great empirical performance on high dimensional continuous control tasks. Model-free approaches operate by constructing lower bounds on policy performance and then optimizing the policy with respect to this lower bound [255, 265]. The model-based counterparts first learn a world model and the optimize a policy using model-based rollouts via off-the-shelf algorithms such as Natural Policy Gradient [230] or Soft-Actor Critic [176]. Pessimism is introduced by either terminating model rollouts using uncertainty estimation from an ensemble of neural network models [245] or modifying the reward function to penalize visiting uncertain regions [530]. [529] propose a hybrid model-based and model-free approach that integrates model-based rollouts into a model-free algorithm to construct tighter lower bounds on policy performance. On the more theoretical side, the offline RL approaches built upon the pessimistic concept [e.g., 224, 311, 392, 427, 481, 512, 533] also illustrate desired theoretical efficacy under various of setups.

Another class of approaches employs an adversarial training framework, where offline RL is posed a two player game between an adversary that chooses the worst-case hypothesis (e.g., a value function or an MDP model) from a hypothesis class, and a policy player that tried to maximize the adversarially chosen hypothesis. [512] propose the concept of Bellman-consistent pessimism to constrain the class of value functions to be Bellman consistent on the data. [85] extend this framework by introducing a relative pessimism objective which allows for robust policy improvement over the data collection policy μ for a wide range of hyper-parameters. Our approach can be interpreted as a model-based extension of [85]. These approaches provide strong theoretical guarantees even with general function approximators while making minimal assumptions about the function class (realizability and Bellman completeness). [78] provide an adversarial model learning method that uses an adversarial policy to generate a data-distribution where the model performs poorly and iteratively updating the model on the generated distribution. There also exist model-based approaches based on the same principle [398, 481] for optimizing the absolute performance. Of these, [398] is the closest to our approach, as they also aim to find an adversarial MDP model that minimizes policy performance. They use a policy gradient approach to train the model, and demonstrate great empirical performance. However, their approach is based on

absolute pessimism and does not enjoy the same RPI property as ARMOR.

6.7 Discussion

The RPI of ARMOR is highly valuable as it allows easy tuning of the pessimism hyperparameter without performance degradation. We believe that leveraging this property can pave the way for real-world deployment of offline RL. Thus, we next present a discussion of RPI.

When does RPI actually improve over the reference policy?

Given ARMOR's ability to improve over an arbitrary policy, the following question naturally arises: Can ARMOR nontrivially improve the output policy of other offline algorithms, including itself?

If this were true, can we repeatedly run ARMOR to improve over itself and obtain the *best* policy any algorithm can learn offline? Unfortunately, the answer is negative. Not only can ARMOR not improve over itself, but it also cannot improve over a variety of algorithms (e.g., absolute pessimism or minimax regret). In fact, the optimal policy of an *arbitrary* model in the version space $\mathcal{M}_\alpha$ is provably unimprovable (Corollary 17; Section 6.9.6). With a deep dive into when RPI gives nontrivial improvement (Section 6.9.6), we found some interesting observations, which we highlight here.

Return maximization and regret minimization are *different* in offline RL These objectives generally produce different policies, even though they are equivalent in online RL. Their equivalence in online RL relies on the fact that online exploration can eventually resolve any uncertainty. In offline RL with an arbitrary data distribution, there will generally be model uncertainty that cannot be resolved, and the worst-case reasoning over such model uncertainty (i.e., $\mathcal{M}_\alpha$) leads to definitions that are no longer equivalent. Moreover, it is impossible to compare return maximization and regret minimization and make a claim about which is better. *They are not simply an algorithm design choice, but are definitions of the learning goals and guarantees themselves*—and are thus incomparable: if we care about obtaining a guarantee for the worst-case return, the return maximization is optimal by

definition; if we are more interested in a guarantee for the worst-case regret, then regret minimization is optimal. We also note that analyzing algorithms under a metric that is different from the one they are designed for can lead to unusual conclusions, e.g., [511] show that optimistic/neutral/pessimistic algorithms are equally minimax-optimal in terms of their regret guarantees in offline multi-armed bandits. However, the algorithms they consider are optimistic/pessimistic with respect to the *return* (as commonly considered in the offline RL literature) not the *regret* which is the performance metric they are interested in analyzing.

π_{ref} is more than a hyperparameter—it defines the performance metric and learning goal Corollary 17 in Section 6.9.6 shows that ARMOR has many different fixed points: when π_{ref} is chosen from these fixed points, the solution to Equation 6.1 is also π_{ref}. Furthermore, some of them may seem quite unreasonable for offline learning (e.g., the greedy policy to an arbitrary model in $\mathcal{M}_\alpha$ or even the optimistic policy). This is not a defect of the algorithm. Rather, because of the unresolvable uncertainty in the offline setting, there are many different performance metrics/learning goals that are generally incompatible/incomparable, and the agent designer must make a conscious choice among them and convey the intention to the algorithm. In ARMOR, such a choice is explicitly conveyed by π_{ref}, which makes ARMOR subsume return maximization and regret minimization as special cases.

6.8 Conclusion

In this chapter, we presented a model-based offline RL framework, ARMOR, that can improve over arbitrary reference policies regardless of data coverage, by using the concept of relative pessimism. ARMOR provides strong theoretical guarantees with general function approximators, and exhibits robust policy improvement over the reference policy for a wide range of hyper-parameters. We also presented a scalable deep learning instantiation of the theoretical algorithm. Our empirical evaluations demonstrate that ARMOR indeed enjoys the RPI property, and has competitive performance with several SoTA model-free and model-based offline RL algorithms, while employing a simpler model architecture (a single MDP model) than other model-based baselines that rely on ensembles. This also opens the opportunity to leverage high-capacity world models [177] with offline RL in the

future. However, the presented approach also has some **limitations**. While RPI holds for the pessimism parameter, the others still need to be tuned. In practice, the non-convexity of the optimization can also make solving the two-player game challenging. For instance, if the adversary is not strong enough (i.e., far from solving the inner minimization), RPI would break. Further, runtime of ARMOR is slightly slower than model-free algorithms owing to extra computations for model rollouts.

6.9 Appendix

6.9.1 Proofs for Section 6.3

Technical Tools

Lemma 11 (Simulation lemma)**.** *Consider any two MDP model M and M', and any $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, we have*

$$|J_M(\pi) - J_{M'}(\pi)| \le \frac{V_{\max}}{1-\gamma}\mathbb{E}_{d^\pi}\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M'}(\cdot \mid s,a)\right)\right] + \frac{1}{1-\gamma}\mathbb{E}_{d^\pi}\left[|R_M(s,a) - R_{M'}(s,a)|\right].$$

Lemma 11 is the standard simulation lemma in model-based reinforcement learning literature, and its proof can be found in, e.g., [481, Lemma 7].

6.9.2 MLE Guarantees

We use $\ell_{\mathcal{D}}(M)$ to denote the likelihood of model $M = (P, R)$ with offline data $\mathcal{D}$, where

$$\ell_{\mathcal{D}}(M) = \prod_{(s,a,r,s') \in \mathcal{D}} P_M(s' \mid s,a). \tag{6.8}$$

For the analysis around maximum likelihood estimation, we largely follow the proving idea of [11, 309], which is inspired by [538].

The next lemma shows that the ground truth model $M^\star$ has a comparable log-likelihood compared with MLE solution.

Lemma 12. *Let $M^\star$ be the ground truth model. Then, with probability at least $1-\delta$, we have*

$$\max_{M \in \mathcal{M}} \log \ell_{\mathcal{D}}(M) - \log \ell_{\mathcal{D}}(M^\star) \le \log(|\mathcal{M}|/\delta).$$

Proof of Lemma 12. The proof of this lemma is obtained by a standard argument of MLE [see, e.g., 485]. For any $M \in \mathcal{M}$,

$$\begin{aligned}
\mathbb{E}\left[\exp\left(\log \ell_{\mathcal{D}}(M) - \log \ell_{\mathcal{D}}(M^\star)\right)\right] &= \mathbb{E}\left[\frac{\ell_{\mathcal{D}}(M)}{\ell_{\mathcal{D}}(M^\star)}\right] \\
&= \mathbb{E}\left[\frac{\prod_{(s,a,r,s')\in\mathcal{D}} P_M(s' \mid s,a)}{\prod_{(s,a,r,s')\in\mathcal{D}} P_{M^\star}(s' \mid s,a)}\right] \\
&= \mathbb{E}\left[\prod_{(s,a,r,s')\in\mathcal{D}} \frac{P_M(s' \mid s,a)}{P_{M^\star}(s' \mid s,a)}\right] \\
&= \mathbb{E}\left[\prod_{(s,a)\in\mathcal{D}} \mathbb{E}\left[\frac{P_M(s' \mid s,a)}{P_{M^\star}(s' \mid s,a)} \;\middle|\; s,a\right]\right] \\
&= \mathbb{E}\left[\prod_{(s,a)\in\mathcal{D}} \sum_{s'} P_M(s' \mid s,a)\right] \\
&= 1. \qquad (6.9)
\end{aligned}$$

Then by Markov's inequality, we obtain

$$\begin{aligned}
&\mathbb{P}\left[\left(\log \ell_{\mathcal{D}}(M) - \log \ell_{\mathcal{D}}(M^\star)\right) > \log(1/\delta)\right] \\
&\leq \underbrace{\mathbb{E}\left[\exp\left(\log \ell_{\mathcal{D}}(M) - \log \ell_{\mathcal{D}}(M^\star)\right)\right]}_{=1 \text{ by Equation } 6.9} \cdot \exp\left[-\log(1/\delta)\right] = \delta.
\end{aligned}$$

Therefore, taking a union bound over $\mathcal{M}$, we obtain

$$\mathbb{P}\left[\left(\log \ell_{\mathcal{D}}(M) - \log \ell_{\mathcal{D}}(M^\star)\right) > \log(|\mathcal{M}|/\delta)\right] \leq \delta.$$

This completes the proof. □

The following lemma shows that, the on-support error of any model $M \in \mathcal{M}$ can be captured via its log-likelihood (by comparing with the MLE solution).

Lemma 13. *For any model M, we have with probability at least $1-\delta$,*

$$\mathbb{E}_\mu\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2\right] \leq \mathcal{O}\left(\frac{\log \ell_{\mathcal{D}}(M^\star) - \log \ell_{\mathcal{D}}(M) + \log(|\mathcal{M}|/\delta)}{n}\right),$$

where $\ell_{\mathcal{D}}(\cdot)$ is defined in Equation 6.8.

Proof of Lemma 13. By [11, Lemma 25], we have

$$\mathbb{E}_{\mu}\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2\right] \leq \\ -2\log \mathbb{E}_{\mu\times P_{M^\star}}\left[\exp\left(-\frac{1}{2}\log\left(\frac{P_{M^\star}(s' \mid s,a)}{P_M(s' \mid s,a)}\right)\right)\right], \tag{6.10}$$

where $\mu \times P_{M^\star}$ denote the ground truth offline joint distribution of (s,a,s').

Let $\widetilde{\mathcal{D}} = \{(\widetilde{s}_i, \widetilde{a}_i, \widetilde{r}_i, \widetilde{s}'_i)\}_{i=1}^n \sim \mu$ be another offline dataset that is independent to $\mathcal{D}$. Then,

$$\begin{aligned} & -n\cdot\log \mathbb{E}_{\mu\times P_{M^\star}}\left[\exp\left(-\frac{1}{2}\log\left(\frac{P_{M^\star}(s' \mid s,a)}{P_M(s' \mid s,a)}\right)\right)\right] \\ = & -\sum_{i=1}^n \log \mathbb{E}_{(\widetilde{s}_i,\widetilde{a}_i,\widetilde{s}'_i)\sim\mu}\left[\exp\left(-\frac{1}{2}\log\left(\frac{P_{M^\star}(\widetilde{s}'_i \mid \widetilde{s}_i,\widetilde{a}_i)}{P_M(\widetilde{s}'_i \mid \widetilde{s}_i,\widetilde{a}_i)}\right)\right)\right] \\ = & -\log \mathbb{E}_{\widetilde{\mathcal{D}}\sim\mu}\left[\exp\left(\sum_{i=1}^n -\frac{1}{2}\log\left(\frac{P_{M^\star}(\widetilde{s}'_i \mid \widetilde{s}_i,\widetilde{a}_i)}{P_M(\widetilde{s}'_i \mid \widetilde{s}_i,\widetilde{a}_i)}\right)\right) \;\middle|\; \mathcal{D}\right] \\ = & -\log \mathbb{E}_{\widetilde{\mathcal{D}}\sim\mu}\left[\exp\left(\sum_{(s,a,s')\in\widetilde{\mathcal{D}}} -\frac{1}{2}\log\left(\frac{P_{M^\star}(s' \mid s,a)}{P_M(s' \mid s,a)}\right)\right) \;\middle|\; \mathcal{D}\right]. \end{aligned} \tag{6.11}$$

We use $\ell_M(s,a,s')$ as the shorthand of $-\frac{1}{2}\log\left(\frac{P_{M^\star}(s'|s,a)}{P_M(s'|s,a)}\right)$, for any $(s,a,s') \in \mathcal{S}\times\mathcal{A}\times\mathcal{S}$. By [11, Lemma 24] [see also 309, Lemma 15], we know

$$\mathbb{E}_{\mathcal{D}\sim\mu}\left[\exp\left(\sum_{(s,a,s')\in\mathcal{D}} \ell_M(s,a,s') - \log \mathbb{E}_{\widetilde{\mathcal{D}}\sim\mu}\left[\exp\left(\sum_{(s,a,s')\in\widetilde{\mathcal{D}}} \ell_M(s,a,s')\right) \;\middle|\; \mathcal{D}\right] - \log|\mathcal{M}|\right)\right] \\ \leq 1.$$

Thus, we can use Chernoff method as well as a union bound on the equation above to obtain the following exponential tail bound: with probability at least $1-\delta$, we have for all $(P,R) = M \in \mathcal{M}$,

$$-\log \mathbb{E}_{\widetilde{\mathcal{D}}\sim\mu}\left[\exp\left(\sum_{(s,a,s')\in\widetilde{\mathcal{D}}} \ell_M(s,a,s')\right) \;\middle|\; \mathcal{D}\right] \leq -\sum_{(s,a,s')\in\mathcal{D}} \ell_M(s,a,s') + 2\log(|\mathcal{M}|/\delta). \tag{6.12}$$

Plugging back the definition of ℓ_M and combining Eq. 6.10, Eq. 6.11, Eq. 6.12, we obtain

$$n\cdot\mathbb{E}_{\mu}\left[D_{\mathrm{TV}}\left(P(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2\right] \leq \frac{1}{2}\sum_{(s,a,s')\in\mathcal{D}} \log\left(\frac{P_{M^\star}(s' \mid s,a)}{P(s' \mid s,a)}\right) + 2\log(|\mathcal{M}|/\delta).$$

Therefore, we obtain

$$\begin{aligned}
& n \cdot \mathbb{E}_{\mu}\left[D_{\mathrm{TV}}\left(P(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2\right] \\
\lesssim & \sum_{(s,a,s')\in\mathcal{D}} \log\left(\frac{P_{M^\star}(s' \mid s,a)}{P(s' \mid s,a)}\right) + \log(|\mathcal{M}|/\delta) \\
= & \log \ell_{\mathcal{D}}(M^\star) - \log \ell_{\mathcal{D}}(M) + \log(|\mathcal{M}|/\delta). \qquad (\ell_{\mathcal{D}}(\cdot) \text{ is defined in Equation 6.8})
\end{aligned}$$

This completes the proof. □

6.9.3 Guarantees about Model Fitting Loss

Lemma 14. *Let $M^\star$ be the ground truth model. Then, with probability at least $1-\delta$, we have*

$$\mathcal{E}_{\mathcal{D}}(M^\star) - \min_{M\in\mathcal{M}} \mathcal{E}_{\mathcal{D}}(M) \le \mathcal{O}\left(\log(|\mathcal{M}|/\delta)\right),$$

where $\mathcal{E}_{\mathcal{D}}$ is defined in Eq. (6.3).

Proof of Lemma 14. By defition, we know

$$\mathcal{E}_{\mathcal{D}}(M) = -\log \ell_{\mathcal{D}}(M) + {}^{(R_M(s,a)-r)^2}/_{V_{\max}^2}$$

By Lemma 12, we know

$$\max_{M\in\mathcal{M}} \log \ell_{\mathcal{D}}(M) - \log \ell_{\mathcal{D}}(M^\star) \le \log(|\mathcal{M}|/\delta). \tag{6.13}$$

In addition, by [512, Theorem A.1] (with setting $\gamma = 0$), we know w.p. $1-\delta$,

$$\sum_{(s,a,r,s')\in\mathcal{D}} \left(R^\star(s,a) - r\right)^2 - \min_{M\in\mathcal{M}} \sum_{(s,a,r,s')\in\mathcal{D}} \left(R_M(s,a) - r\right)^2 \lesssim \log(|\mathcal{M}|/\delta). \tag{6.14}$$

Combining Eqs. (6.13) and (6.14) and using the fact of $V_{\max} \ge 1$, we have w.p. $1-\delta$,

$$\begin{aligned}
& \mathcal{E}_{\mathcal{D}}(M^\star) - \min_{M\in\mathcal{M}} \mathcal{E}_{\mathcal{D}}(M) \\
\le & \max_{M\in\mathcal{M}} \log \ell_{\mathcal{D}}(M) - \min_{M\in\mathcal{M}} \sum_{(s,a,r,s')\in\mathcal{D}} {}^{(R_M(s,a)-r)^2}/_{V_{\max}^2} + \mathcal{E}_{\mathcal{D}}(M^\star) \\
\lesssim & \log(|\mathcal{M}|/\delta).
\end{aligned}$$

This completes the proof. □

Lemma 15. *For any $M \in \mathcal{M}$, we have with probability at least $1-\delta$,*

$$\mathbb{E}_\mu\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2 + {}^{(R_M(s,a)-R^\star(s,a))^2}/_{V_{\max}^2}\right] \le \mathcal{O}\left(\frac{\mathcal{E}_{\mathcal{D}}(M) - \mathcal{E}_{\mathcal{D}}(M^\star) + \log({}^{|\mathcal{M}|}/_{\delta})}{n}\right),$$

where $\mathcal{E}_{\mathcal{D}}$ is defined in Eq. (6.3).

Proof of Lemma 15. By Lemma 13, we have w.p. $1-\delta$,

$$n \cdot \mathbb{E}_\mu\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2\right] \lesssim \log \ell_{\mathcal{D}}(M^\star) - \log \ell_{\mathcal{D}}(M) + \log({}^{|\mathcal{M}|}/_{\delta}). \tag{6.15}$$

Also, we have

$$\begin{aligned}
& n \cdot \mathbb{E}_\mu\left[\left(R_M(s,a) - R^\star(s,a)\right)^2\right] \qquad (6.16)\\
&= n \cdot \mathbb{E}_\mu\left[\left(R_M(s,a) - r\right)^2\right] - n \cdot \mathbb{E}_\mu\left[\left(R^\star(s,a) - r\right)^2\right] \\
&\qquad\qquad \text{([see, e.g., 512, Eq. (A.10) with } \gamma = 0\text{])} \\
&\lesssim \sum_{(s,a,r,s')\in\mathcal{D}} \left(R_M(s,a) - r\right)^2 - \sum_{(s,a,r,s')\in\mathcal{D}} \left(R^\star(s,a) - r\right)^2 + \log({}^{|\mathcal{M}|}/_{\delta}),
\end{aligned}$$

where the last inequality is a direct implication of [512, Lemma A.4]. Combining Eqs. (6.15) and (6.16) and using the fact of $V_{\max} \ge 1$, we obtain

$$\begin{aligned}
& n \cdot \mathbb{E}_\mu\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2 + {}^{(R_M(s,a)-R^\star(s,a))^2}/_{V_{\max}^2}\right] \\
&\lesssim \log \ell_{\mathcal{D}}(M^\star) - \sum_{(s,a,r,s')\in\mathcal{D}} {}^{(R^\star(s,a)-r)^2}/_{V_{\max}^2} - \log \ell_{\mathcal{D}}(M) + \sum_{(s,a,r,s')\in\mathcal{D}} {}^{(R_M(s,a)-r)^2}/_{V_{\max}^2} + \log({}^{|\mathcal{M}|}/_{\delta}) \\
&= \mathcal{E}_{\mathcal{D}}(M) - \mathcal{E}_{\mathcal{D}}(M^\star) + \log({}^{|\mathcal{M}|}/_{\delta}).
\end{aligned}$$

This completes the proof. □

6.9.4 Proof of Main Theorems

Proof of Theorem 9. By the optimality of $\widehat{\pi}$ (from Eq. (6.1)), we have

$$\begin{aligned}
J(\pi^\dagger) - J(\widehat{\pi}) &= J(\pi^\dagger) - J(\pi_{\mathsf{ref}}) - \left[J(\widehat{\pi}) - J(\pi_{\mathsf{ref}})\right] \\
&\le J(\pi^\dagger) - J(\pi_{\mathsf{ref}}) - \min_{M\in\mathcal{M}_\alpha} \left[J_M(\widehat{\pi}) - J_M(\pi_{\mathsf{ref}})\right] \qquad (\star) \\
&\le J(\pi^\dagger) - J(\pi_{\mathsf{ref}}) - \min_{M\in\mathcal{M}_\alpha} \left[J_M(\pi^\dagger) - J_M(\pi_{\mathsf{ref}})\right], \qquad (6.17)
\end{aligned}$$

where step equation $\star$ follows from Lemma 12 so that we have $M^\star \in \mathcal{M}_\alpha$, and the last step is because of $\pi^\dagger \in \Pi$. By the simulation lemma (Lemma 11), we know for any policy π and any $M \in \mathcal{M}_\alpha$,

$$
\begin{aligned}
|J(\pi) - J_M(\pi)| &\le \frac{V_{\max}}{1-\gamma}\mathbb{E}_{d^\pi}\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)\right] + \frac{1}{1-\gamma}\mathbb{E}_{d^\pi}\left[|R_M(s,a) - R^\star(s,a)|\right] \\
&\le \frac{V_{\max}}{1-\gamma}\sqrt{\mathbb{E}_{d^\pi}\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2\right]} + \frac{V_{\max}}{1-\gamma}\sqrt{\mathbb{E}_{d^\pi}\left[\left(R_M(s,a)-R^\star(s,a)\right)^2/V_{\max}^2\right]} \\
&\lesssim \frac{V_{\max}}{1-\gamma}\sqrt{\mathbb{E}_{d^\pi}\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2 + \left(R_M(s,a)-R^\star(s,a)\right)^2/V_{\max}^2\right]} \\
&\qquad\qquad (a \lesssim b \text{ means } a \le \mathcal{O}(b)) \\
&\le \frac{V_{\max}\sqrt{\mathfrak{C}_{\mathcal{M}}(\pi)}}{1-\gamma}\sqrt{\mathbb{E}_{\mu}\left[D_{\mathrm{TV}}\left(P_M(\cdot \mid s,a), P_{M^\star}(\cdot \mid s,a)\right)^2 + \left(R_M(s,a)-R^\star(s,a)\right)^2/V_{\max}^2\right]} \\
&\lesssim \frac{V_{\max}\sqrt{\mathfrak{C}_{\mathcal{M}}(\pi)}}{1-\gamma}\sqrt{\frac{\mathcal{E}_{\mathcal{D}}(M) - \mathcal{E}_{\mathcal{D}}(M^\star) + \log(|\mathcal{M}|/\delta)}{n}} \qquad \text{(by Lemma 15)} \\
&\lesssim \frac{V_{\max}\sqrt{\mathfrak{C}_{\mathcal{M}}(\pi)}}{1-\gamma}\sqrt{\frac{\mathcal{E}_{\mathcal{D}}(M) - \min_{M' \in \mathcal{M}}\mathcal{E}_{\mathcal{D}}(M') + \log(|\mathcal{M}|/\delta)}{n}} \qquad (\ddagger) \\
&\lesssim \frac{V_{\max}\sqrt{\mathfrak{C}_{\mathcal{M}}(\pi)}}{1-\gamma}\sqrt{\frac{\log(|\mathcal{M}|/\delta)}{n}} \qquad (6.18)
\end{aligned}
$$

where the step equation $\ddagger$ follows from the assumption of $M^\star \in \mathcal{M}$, and last step is because $\mathcal{E}_{\mathcal{D}}(M) - \min_{M' \in \mathcal{M}}\mathcal{E}_{\mathcal{D}}(M') \le \alpha = \mathcal{O}(\log(|\mathcal{M}|/\delta)$ by Eq. (6.2).

Combining Eqs. (6.17) and (6.18), we obtain

$$
J(\pi^\dagger) - J(\widehat{\pi}) \lesssim \left[\sqrt{\mathfrak{C}_{\mathcal{M}}(\pi^\dagger)} + \sqrt{\mathfrak{C}_{\mathcal{M}}(\pi_{\mathsf{ref}})}\right] \cdot \frac{V_{\max}}{1-\gamma}\sqrt{\frac{\log(|\mathcal{M}|/,\delta)}{n}}.
$$

This completes the proof. □

Note that, over the proof above, only steps equation $\star$ and equation $\ddagger$ have used the realizability assumption of $M^\star \in \mathcal{M}$. To extend that to the misspecification case, where there only exists an $\widetilde{M}^\star \in \mathcal{M}$ such that $\widetilde{M}^\star$ is close to $M^\star$ up to some misspecification error, we just need the following straightforward accommodations:

(1) A variant of Lemma 12—to ensure that $\widetilde{M}^\star$ is included in the version space $\mathcal{M}_\alpha$. By doing so, the misspecification error should also be included in the radius of the version space.

(2) Upper bound $|J(\pi) - J_{\widetilde{M}^\star}(\pi)|$ for any π using misspecification error. This is a standard argument, and by combining with the item above, equation $\star$ becomes $J(\widehat{\pi}) - J(\pi_{\mathsf{ref}}) \ge \min_{M \in \mathcal{M}_\alpha}\left[J_M(\widehat{\pi}) - J_M(\pi_{\mathsf{ref}})\right] - \mathsf{misspecification\ error}$.

(3) Upper bound difference in model-fitting error $\mathcal{E}_{\mathcal{D}}(\widetilde{M}^\star) - \mathcal{E}_{\mathcal{D}}(M^\star)$ using misspecification error. Then, step equation ‡ becomes, for $M \in \mathcal{M}_\alpha$,

$$\begin{aligned}\mathcal{E}_{\mathcal{D}}(M) - \mathcal{E}_{\mathcal{D}}(M^\star) &= \mathcal{E}_{\mathcal{D}}(M) - \mathcal{E}_{\mathcal{D}}(\widetilde{M}^\star) + \mathcal{E}_{\mathcal{D}}(\widetilde{M}^\star) - \mathcal{E}_{\mathcal{D}}(M^\star) \\ &\leq \mathcal{E}_{\mathcal{D}}(M) - \min_{M' \in \mathcal{M}} \mathcal{E}_{\mathcal{D}}(M') + \textsf{misspecification error} \\ &\lesssim \log\left(|\mathcal{M}|/\delta\right) + \textsf{misspecification error}.\end{aligned}$$

Due to the unboundedness of the likelihood, we conjecture that naively defining misspecification error using total variation without any accommodation on the MLE loss may be insufficient for the steps above. To resolve that, we may adopt an alternate misspecification definition, e.g., $\left|\log P_{M^\star}(s' \mid s, a) - \log P_{\widetilde{M}^\star}(s' \mid s, a)\right| \leq \varepsilon,\ \forall (s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, or add extra smoothing to the MLE loss with regularization.

Proof of Theorem 10.

$$\begin{aligned}J(\pi_{\textsf{ref}}) - J(\widehat{\pi}) &= J(\pi_{\textsf{ref}}) - J(\pi_{\textsf{ref}}) - [J(\widehat{\pi}) - J(\pi_{\textsf{ref}})] \\ &\leq -\min_{M \in \mathcal{M}_\alpha} [J_M(\widehat{\pi}) - J_M(\pi_{\textsf{ref}})] \qquad \text{(by Lemma 12, we have } M^\star \in \mathcal{M}_\alpha) \\ &= -\max_{\pi \in \Pi} \min_{M \in \mathcal{M}_\alpha} [J_M(\pi) - J_M(\pi_{\textsf{ref}})] \\ &\qquad\qquad \text{(by the optimality of } \widehat{\pi} \text{ from Eq. (6.1))} \\ &\leq -\min_{M \in \mathcal{M}_\alpha} [J_M(\pi_{\textsf{ref}}) - J_M(\pi_{\textsf{ref}})] \qquad (\pi_{\textsf{ref}} \in \Pi) \\ &= 0.\end{aligned}$$

□

A misspecified version of Theorem 10 can be derived similarly to what we discussed about that of Theorem 9. If the policy class is also misspecified, where there exists only $\widetilde{\pi}_{\textsf{ref}} \in \Pi$ that is close to $\pi_{\textsf{ref}}$ up to some misspecification error, the second last step of the proof of Theorem 10 becomes $-\min_{M \in \mathcal{M}_\alpha} [J_M(\widetilde{\pi}_{\textsf{ref}}) - J_M(\pi_{\textsf{ref}})] \leq \textsf{misspecification error}$ by simply applying the performance difference lemma on the difference between $\widetilde{\pi}_{\textsf{ref}}$ and $\pi_{\textsf{ref}}$.

6.9.5 Proofs for Section 6.7

Proof of Lemma 16. We prove the result by contradiction. First notice $\min_{M\in\mathcal{M}} J_M(\pi') - J_M(\pi') = 0$. Suppose there is $\overline{\pi} \in \Pi$ such that $\min_{M\in\mathcal{M}_\alpha} J_M(\bar{\pi}) - J_M(\pi') > 0$, which implies that $J_M(\bar{\pi}) > J_M(\pi')$, $\forall M \in \mathcal{M}_\alpha$. Since $\mathcal{M} \subseteq \mathcal{M}_\alpha$, we have

$$\min_{M\in\mathcal{M}} J_M(\bar{\pi}) + \psi(M) > \min_{M\in\mathcal{M}} J_M(\pi') + \psi(M) = \max_{\pi\in\Pi} \min_{M\in\mathcal{M}} J_M(\pi) + \psi(M)$$

which is a contradiction of the maximin optimality. Thus $\max_{\pi\in\Pi} \min_{M\in\mathcal{M}_\alpha} J_M(\bar{\pi}) - J_M(\pi') = 0$, which means π' is a solution.

For the converse statement, suppose π is a fixed point. We can just let $\psi(M) = -J_M(\pi)$. Then this pair of π and ψ by definition of the fixed point satisfies Equation 6.19. □

6.9.6 A Deeper Discussion of Robust Policy Improvement

How to formally define RPI?

Improving over some reference policy has been long studied in the literature. To highlight the advantage of ARMOR, we formally give the definition of different policy improvement properties.

Definition 2 (Robust policy improvement)**.** *Suppose $\widehat{\pi}$ is the learned policy from an algorithm. We say the algorithm has the policy improvement (PI) guarantee if $J(\pi_{\mathsf{ref}}) - J(\widehat{\pi}) \leq {}^{o(N)}/_{N}$ is guaranteed for some reference policy π_{ref} with offline data $\mathcal{D} \sim \mu$, where $N = |\mathcal{D}|$. We use the following two criteria w.r.t. π_{ref} and μ to define different kinds PI:*

(i) The PI is strong if π_{ref} can be selected arbitrarily from policy class Π regardless of the choice data-collection policy μ; otherwise, PI is weak (i.e., $\pi_{\mathsf{ref}} \equiv \mu$ is required).

(ii) The PI is robust if it can be achieved by a range of hyperparameters with a known subset.

Weak policy improvement is also known as *safe policy improvement* in the literature [162, 277]. It requires the reference policy to be also the behavior policy that collects the offline data. In comparison, strong policy improvement imposes a stricter requirement, which

requires policy improvement *regardless* of how the data were collected. This condition is motivated by the common situation where the reference policy is not the data collection policy. Finally, since we are learning policies offline, without online interactions, it is not straightforward to tune the hyperparameter directly. Therefore, it is desirable that we can design algorithms with these properties in a robust manner in terms of hyperparameter selection. Formally, Definition 2 requires the policy improvement to be achievable by a set of hyperparameters that is known before learning.

Theorem 10 indicates the robust strong policy improvement of ARMOR. On the other hand, algorithms with robust weak policy improvement are available in the literature [85, 160, 162, 264, 277, 509]; this is usually achieved by designing the algorithm to behave like IL for a known set of hyperparameter (e.g., behavior regularization algorithms have a weight that can turn off the RL behavior and regress to IL). However, deriving guarantees of achieving the best data-covered policy of the IL-like algorithm is challenging due to its imitating nature. To our best knowledge, ATAC [85] is the only algorithm that achieves both robust (weak) policy improvement as well as guarantees absolute performance.

When RPI actually improves?

Given ARMOR's ability to improve over an arbitrary policy, the following questions naturally arise: *Can ARMOR nontrivially improve the output policy of other algorithms (e.g., such as those based on* absolute pessimism *[512]), including itself?* Note that outputting π_{ref} itself always satisfies RPI, but such result is trivial. By "nontrivially" we mean a non-zero worst-case improvement. If the statement were true, we would be able to repeatedly run ARMOR to improve over itself and then obtain the *best* policy any algorithm can learn offline.

Unfortunately, the answer is negative. Not only ARMOR cannot improve over itself, but it also cannot improve over a variety of algorithms. In fact, the optimal policy of an *arbitrary* model in the version space is unimprovable (see Corollary 17)! Our discussion reveals some interesting observations (e.g., how equivalent performance metrics for online RL can behave very differently in the offline setting) and their implications (e.g., how we should choose π_{ref}

for ARMOR). Despite their simplicity, we feel that many in the offline RL community are not actively aware of these facts (and the unawareness has led to some confusion), which we hope to clarify below.

Setup We consider an abstract setup where the learner is given a version space $\mathcal{M}_\alpha$ that contains the true model and needs to choose a policy $\pi \in \Pi$ based on $\mathcal{M}_\alpha$. We use the same notation $\mathcal{M}_\alpha$ as before, but emphasize that it does not have to be constructed as in Eq. 6.2 and Eq. 6.3. In fact, for the purpose of this discussion, the data distribution, sample size, data randomness, and estimation procedure for constructing $\mathcal{M}_\alpha$ are **all irrelevant**, as our focus here is how decisions should be made with a given $\mathcal{M}_\alpha$. This makes our setup very generic and the conclusions widely applicable.

To facilitate discussion, we define the *fixed point* of ARMOR's relative pessimism step:

Definition 3. *Consider Equation 6.1 as an operator that maps an arbitrary policy π_{ref} to $\widehat{\pi}$. A fixed point of this* relative pessimism *operator is, therefore, any policy $\pi \in \Pi$ such that $\pi \in \arg\max_{\pi' \in \Pi} \min_{M \in \mathcal{M}_\alpha} J_M(\pi') - J_M(\pi)$.*

Given the definition, relative pessimism cannot improve over a policy if it is already a fixed point. Below we show a sufficient and necessary condition for being a fixed point, and show a number of concrete examples (some of which may be surprising) that are fixed points and thus unimprovable.

Lemma 16 (Fixed-point Lemma)**.** *For any $\mathcal{M} \subseteq \mathcal{M}_\alpha$ and any $\psi : \mathcal{M} \to \mathbb{R}$, consider the policy*

$$\pi \in \arg\max_{\pi' \in \Pi} \min_{M \in \mathcal{M}} J_M(\pi') + \psi(M) \tag{6.19}$$

Then π is a fixed point in Definition 3. Conversely, for any fixed point π in Definition 3, there is a $\psi : \mathcal{M} \to \mathbb{R}$ such that π is a solution to Equation 6.19.

Corollary 17. *The following are fixed points of relative pessimism (Definition 3):*

1. *Absolute-pessimism policy, i.e., $\psi(M) = 0$.*

2. *Relative-pessimism policy for any reference policy, i.e., $\psi(M) = -J_M(\pi_{\mathsf{ref}})$.*

3. *Regret-minimization policy, i.e.,* $\psi(M) = -J_M(\pi^\star_M)$*, where* $\pi^\star_M \in \arg\max_{\pi\in\Pi} J_M(\pi)$.

4. *Optimal policy of an* arbitrary *model* $M \in \mathcal{M}_\alpha$*,* $\pi^\star_M$*, i.e.,* $\mathcal{M} = \{M\}$*. This would include the optimistic policy, that is,* $\arg\max_{\pi\in\Pi, M\in\mathcal{M}_\alpha} J_M(\pi)$

Return maximization and regret minimization are *different* in offline RL We first note that these four examples generally produce different policies, even though some of them optimize for objectives that are traditionally viewed as equivalent in online RL (the "worst-case over $\mathcal{M}_\alpha$" part of the definition does not matter in online RL), e.g., absolute pessimism optimizes for $J_M(\pi)$, which is the same as minimizing the regret $J_M(\pi^\star_M) - J_M(\pi)$ for a fixed M. However, their equivalence in online RL relies on the fact that online exploration can eventually resolve any model uncertainty when needed, so we only need to consider the performance metrics w.r.t. the true model $M = M^\star$. In offline RL with an arbitrary data distribution (since we do not make any coverage assumptions), there will generally be model uncertainty that cannot be resolved, and worst-case reasoning over such model uncertainty (i.e., $\mathcal{M}_\alpha$) separates apart the definitions that are once equivalent.

Moreover, it is impossible to compare return maximization and regret minimization and make a claim about which one is better. They are not simply an algorithm design choice, but are definitions of the learning goals and the guarantees themselves—thus incomparable: if we care about obtaining a guarantee for the worst-case *return*, the return maximization is optimal by definition; if we are more interested in obtaining a guarantee for the worst-case *regret*, then again, regret minimization is trivially optimal. We also note that analyzing algorithms under a metric that is different from the one they are designed for can lead to unusual conclusions. For example, [511] show that optimistic/neutral/pessimistic algorithms[6] are equally minimax-optimal in terms of their regret guarantees in offline multi-armed bandits. However, the algorithms they consider are optimistic/pessimistic w.r.t. the return—as commonly considered in the offline RL literature—not w.r.t. the regret which is the performance metric they are interested in analyzing.

[6]Incidentally, optimistic/neutral policies correspond to #4 in Corollary 17.

π_{ref} is more than a hyperparameter—it defines the performance metric and learning goal Corollary 17 shows that ARMOR (with relative pessimism) has many different fixed points, some of which may seem quite unreasonable for offline learning, such as greedy w.r.t. an arbitrary model or even optimism (#4). From the above discussion, we can see that this is not a defect of the algorithm. Rather, in the offline setting with unresolvable model uncertainty, there are many different performance metrics/learning goals that are generally incompatible/incomparable with each other, and the agent designer must make a choice among them and convey the choice to the algorithm. In ARMOR, such a choice is explicitly conveyed by the choice of π_{ref}, which subsumes return maximization and regret minimization as special cases (#2 and #3 in Corollary 17)

6.9.7 Further Experimental Details

Experimental Setup and Hyper-parameters

We represent our policy π, Q-functions f_1, f_2 and MDP model M as standard fully connected neural networks. The policy is parameterized as a Gaussian with a state-dependent covariance, and we use a tanh transform to limit the actions to the action space bound similar to [176]. The MDP model learns to predict the next state distribution, rewards and terminal states, where the reward and next-state distributions part are parameterized as Gaussians with state-dependent covariances. The model fitting loss consists of negative log-likelihood for the next-state and reward and binary cross entropy for the terminal flags. In all our experiments we use the same model architecture and a fixed value of λ. We use Adam optimizer [246] with fixed learning rates η_{fast} and η_{slow} similar to [85]. Also similar to prior work [245], we let the MDP model network predict delta differences to the current state. The rollout horizon is always set to the maximum episode steps per environment. A complete list of hyper-parameters can be found in Table 6.3.

Compute: Each run of ARMOR has access to 4CPUs with 28GB RAM and a single Nvidia T4 GPU with 16GB memory. With these resources each run tasks around 6-7 hours to complete. Including all runs for 4 seeds, and ablations this amounts to approximately 2500 hours of GPU compute.

Hyperparameter	Value
model_num_layers	3
model_hidden_size	512
model_nonlinearity	swish
policy_num_layers	3
policy_hidden_size	256
policy_nonlinearity	relu
f_num_layers	3
f_hidden_size	256
f_nonlinearity	relu

Table 6.2: Model Architecture Details

Hyperparameter	Value
η_{fast}	5e-4
η_{slow}	5e-7
discount factor	0.99
rollout horizon	max episode steps
model buffer size	10^6
batch size	125
model batch size	125
num warmstart steps	10^5
τ	$5e-3$

Table 6.3: List of Hyperparameters.

Detailed Performance Comparison and RPI Ablations

In Table 6.4 we show the performance of ARMOR compared to model-free and model-based offline RL baselines with associate standard deviations over 8 seeds. For ablation, here we also include ARMOR†, which is running ARMOR in Alg. 7 but without the model optimizing for the Bellman error (that is, the model is not adversarial). Although ARMOR† does not have any theoretical guarantees (and indeed in the worst case its performance can be arbitrarily bad), we found that ARMOR† in these experiments is performing surprisingly well. Compared with ARMOR, ARMOR† has less stable performance when the dataset is diverse (e.g. *-med-replay* datasets) and larger learning variance. Nonetheless, ARMOR† using a single model is already pretty competitive with other algorithms. We conjecture that this is due to that Alg. 7 also benefits from pessimism due to adversarially trained critics. Since the model buffer would not cover all states and actions (they are continuous in these problems), the adversarially trained critic still controls the pessimism for actions not in the model buffer, as a safe guard. As a result, the algorithm can tolerate the model quality more.

Dataset	ARMOR	ARMOR†	ARMOR^{re}	MoREL	MOPO	RAMBO	COMBO	ATAC	CQL	IQL	BC
hopper-med	**101.4 ± 0.3**	**100.4 ± 1.7**	65.3 ± 4.8	**95.4**	28.0 ± 12.4	**92.8 ± 6.0**	**97.2 ± 2.2**	85.6	86.6	66.3	29.0
walker2d-med	**90.7 ± 4.4**	**91.0 ± 10.4**	79.0 ± 2.2	77.8	17.8 ± 19.3	**86.9 ± 2.7**	**81.9 ± 2.8**	**89.6**	74.5	78.3	6.6
halfcheetah-med	54.2 ± 2.4	56.3 ± 0.5	45.2 ± 0.2	42.1	42.3 ± 1.6	**77.6 ± 1.5**	54.2 ± 1.5	53.3	44.4	47.4	36.1
hopper-med-replay	**97.1 ± 4.8**	82.7 ± 23.1	68.4 ± 5.2	**93.6**	67.5 ± 24.7	**96.6 ± 7.0**	89.5 ± 1.8	**102.5**	48.6	**94.7**	11.8
walker2d-med-replay	**85.6 ± 7.5**	78.4 ± 1.9	50.3 ± 5.7	49.8	39.0 ± 9.6	**85.0 ± 15.0**	56.0 ± 8.6	**92.5**	32.6	73.9	11.3
halfcheetah-med-replay	50.5 ± 0.9	49.5 ± 0.9	36.8 ± 1.5	40.2	53.1 ± 2.0	**68.9 ± 2.3**	55.1 ± 1.0	48.0	46.2	44.2	38.4
hopper-med-exp	**103.4 ± 5.9**	100.1 ± 10.0	89.3 ± 3.2	**108.7**	23.7 ± 6.0	83.3 ± 9.1	**111.1 ± 2.9**	**111.9**	**111.0**	91.5	**111.9**
walker2d-med-exp	**112.2 ± 1.7**	**110.5 ± 1.4**	**105.8 ± 1.4**	95.6	44.6 ± 12.9	68.3 ± 15.0	**103.3 ± 5.6**	**114.2**	98.7	**109.6**	6.4
halfcheetah-med-exp	**93.5 ± 0.5**	**93.4 ± 0.3**	61.8 ± 3.75	53.3	63.3 ± 38.0	**93.7 ± 10.5**	**90.0 ± 5.6**	**94.8**	62.4	**86.7**	35.8
pen-human	**72.8 ± 13.9**	50.0 ± 15.6	62.3 ± 8.35	-	-	-	-	53.1	37.5	**71.5**	34.4
hammer-human	1.9 ± 1.6	1.1 ± 1.4	3.1 ± 1.9	-	-	-	-	1.5	**4.4**	1.4	1.5
door-human	6.3 ± 6.0	3.9 ± 2.4	5.9 ± 2.75	-	-	-	-	2.5	**9.9**	4.3	0.5
relocate-human	**0.4 ± 0.4**	**0.4 ± 0.6**	**0.3 ± 0.25**	-	-	-	-	0.1	0.2	0.1	0.0
pen-cloned	**51.4 ± 15.5**	45.2 ± 15.8	40.0 ± 8.25	-	-	-	-	43.7	39.2	37.3	**56.9**
hammer-cloned	0.7 ± 0.6	0.3 ± 0.0	**2.7 ± 0.15**	-	-	-	-	1.1	**2.1**	**2.1**	0.8
door-cloned	-0.1 ± 0.0	-0.1 ± 0.1	0.5 ± 0.4	-	-	-	-	**3.7**	0.4	1.6	-0.1
relocate-cloned	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-	-	-	-	**0.2**	-0.1	-0.2	-0.1
pen-exp	112.2 ± 6.3	113.0 ± 11.8	92.8 ± 9.25	-	-	-	-	**136.2**	107.0	-	85.1
hammer-exp	**118.8 ± 5.6**	**115.3 ± 9.3**	51.0 ± 11.05	-	-	-	-	**126.9**	86.7	-	**125.6**
door-exp	**98.7 ± 4.1**	**97.1 ± 4.9**	88.4 ± 3.05	-	-	-	-	**99.3**	**101.5**	-	34.9
relocate-exp	**96.0 ± 6.8**	90.7 ± 6.3	64.2 ± 7.3	-	-	-	-	**99.4**	**95.0**	-	**101.3**

Table 6.4: Performance comparison of ARMOR against baselines on the D4RL datasets. The values for ARMOR denote last iteration performance averaged over 4 random seeds along with standard deviations, and baseline values were taken from their respective papers. Boldface denotes performance within 10% of the best performing algorithm.

Effect of Residual Policy

In Fig. 6.4, we show the effect on RPI of different schemes for initializing the learner for several D4RL datasets. Specifically, we compare using a residual policy(Section 6.5) versus behavior cloning the reference policy on the provided offline dataset for learner initialization. Note that this offline dataset is the suboptimal one used in offline RL and is different from the expert-level dataset used to train and produce the reference policy. We observe that using a residual policy (purple) consistently shows RPI across all datasets. However, with behavior cloning initialization (pink), there is a large variation in performance across datasets. While RPI is achieved with behavior cloning initialization on *hopper*, *walker2d* and *hammer* datasets, performance can be arbitrarily bad compared to the reference on other problems. As an ablation, we also study the effect of using a residual policy in the offline

RL case where no explicit reference is provided, and the behavior cloning policy is used as the reference similar to Section 6.5.1. We include the results in Table 6.4 as ARMOR^{re}, where we observe that using a residual policy overall leads to worse performance across all datasets. This lends evidence to the fact that using a residual policy is a *compromise* in instances where initializing the learner exactly to the reference policy is not possible.

Connection to Imitation Learning

Dataset	ARMOR-IL	BC
hopper-exp	111.6	111.7
walker2d-exp	108.1	108.5
halfcheetah-exp	93.9	94.7

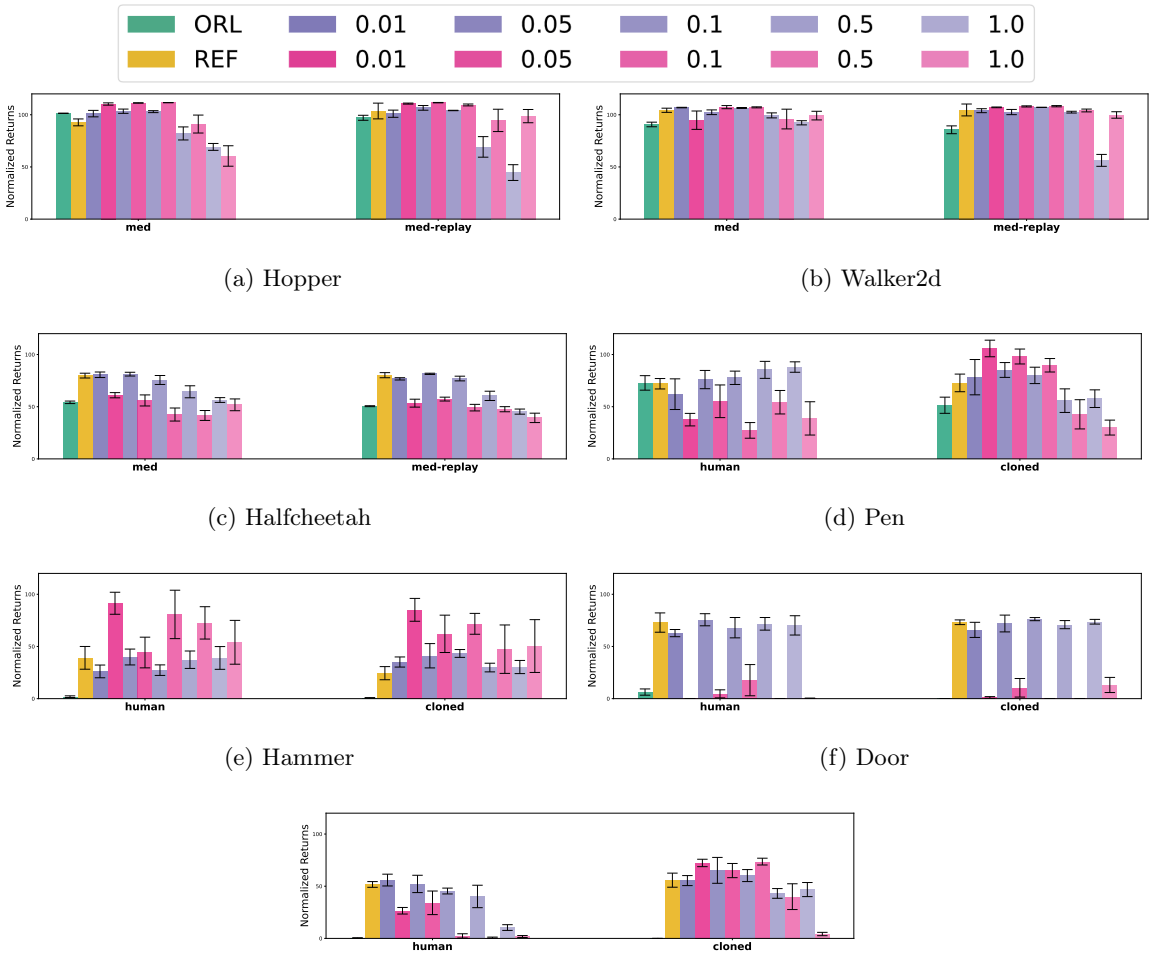
Table 6.5: ARMOR-IL on expert datasets. By setting $\lambda = 0$, $\beta > 0$ we recover IL.

As mentioned in Section 6.4.2, IL is a special case of ARMOR with $\lambda = 0$. In this setting, the Q-function can fully affect the adversarial MDP model, so the best strategy of the policy is to mimic the reference. We test this on the *expert* versions of the D4RL locomotion tasks in Table 6.5, and observe that ARMOR can indeed perform IL to match expert performance.

Ablation Study: RPI for Different Reference Policies

Here we provide ablation study results for robust policy improvement under different reference policies for a wide range of β values (pessimism hyper-parameter). For all the considered reference policies we present average normalized scroes for ARMOR and reference (REF) over multiple random seeds, and observe that ARMOR can consistently outperform the reference for a large range of β values.

Random Dataset Reference We use a reference policy obtained by running behavior cloning on the RANDOM versions of different datasets. This is equivalent to using a randomly initialized neural network as the reference.



(a) Hopper

(b) Walker2d

(c) Halfcheetah

(d) Pen

(e) Hammer

(f) Door

(g) Relocate

Figure 6.4: Comparison of different policy initializations for RPI with varying pessimism hyper-parameter β . ORL denotes the performance of offline RL with ARMOR (Table 6.1), and REF is the performance of reference policy. Purple represents residual policy initialization and pink is initialization using behavior cloning of the reference on the suboptimal offline RL dataset.

Dataset	0.01	0.05	0.1	0.5	1.0	10.0	100.0	200.0	500.0	1000.0	REF
hopper-med	1.3	1.5	4.7	9.6	20.4	34.8	25.8	40.8	25.6	28.9	1.2
walker2d-med	0.0	0.0	0.2	1.5	4.0	17.4	23.6	12.4	12.1	20.1	0.0
halfcheetah-med	0.0	0.1	0.1	0.7	1.2	-0.1	-0.7	0.1	1.1	-0.3	-0.1
hopper-med-replay	1.3	3.0	8.2	13.3	39.5	57.6	48.0	34.4	51.0	32.9	1.2
walker2d-med-replay	0.0	0.0	0.1	4.0	5.9	13.1	10.8	14.1	16.5	16.6	0.0
halfcheetah-med-replay	-0.2	0.4	0.3	0.7	0.9	7.7	5.8	8.2	4.9	6.1	-0.2

Hand-designed Reference In this experiment we use a hand-designed reference policy called RandomBangBang, that selects either the minimum of maximum action in the data with 0.5 probability each.

Dataset	0.01	0.05	0.1	0.5	1.0	10.0	100.0	200.0	500.0	1000.0	REF
hopper-med	8.5	15.0	15.8	5.0	8.1	11.1	37.0	19.6	12.4	25.6	1.2
walker2d-med	22.7	36.1	31.6	42.6	12.0	55.9	33.6	37.8	36.6	49.6	0.1
halfcheetah-med	10.6	9.8	19.1	11.7	13.1	15.6	14.3	3.8	8.3	11.1	-1.0
hopper-med-replay	33.9	54.8	62.6	66.7	55.3	57.0	67.7	79.5	70.7	62.0	1.3
walker2d-med-replay	11.6	35.5	47.5	40.4	42.7	47.6	64.8	49.9	44.7	33.0	0.1
halfcheetah-med-replay	13.1	10.1	11.3	9.7	12.4	17.6	9.0	14.5	11.1	9.1	-1.3

Chapter 7

CONCLUSION

In this thesis, we demonstrated how a principled combination of model-based optimization with data-driven approaches can enable robots to be efficient, adaptive and improve their performance over time. To this end, we developed both algorithmic frameworks with strong theoretical guarantees and empirical performance, as well as practical systems that are deployed on real-world robots.

First, we focused on long-horizon robot motion planning with perfectly known models of the robot and the environment, showing how past experience can be leveraged to enhance the real-time performance. Specifically, in Chapter 2, we examined the problem of minimizing computationally expensive edge evaluations in lazy search and presented an algorithm to learn edge selection policies by imitating oracle edge selectors. We theoretically analyzed our proposed framework in the Bayesian setting drawing a novel connection to Bayesian Active Learning. In Chapter 3, we studied how motion planning via trajectory optimization can be improved across changing environments. We presented dGPMP2, an end-to-end trainable motion planning algorithm, by reformulating GPMP2 as a differentiable computational graph. Our experimental results show that these strategies can enable flexible planning algorithms that seamlessly operate across diverse scenarios.

Second, we considered the setting of *approximate* prior models. Chapter 4 presented STORM, a GPU-accelerated, sampling-based MPC framework that can leverages an approximate forward model to rapidly optimize complex task objectives, and demonstrated its effectiveness on real-world reactive manipulation tasks. We also introduced novel sampling strategies and intuitive cost terms to encourage desirable behaviors like smoothness and constraint satisfaction. In Chapter 5, we studied how the MPC can be improved from experience via model-free RL to address biases from approximate models. Here, we presented a general framework to blend model-free value estimates with MPC, thus systematically mitigating

the effects of model-bias.

Finally, we studied the case of learning policies with only access to prior static datasets. In Chapter 6 presented ARMOR, a model-based offline RL framework based on relative pessimism that can improve over arbitrary reference policies regardless of data coverage. We provided a rigorous theoretical analysis and extensive empirical evaluation on benchmark tasks, demonstrating strong performance and robustness to hyperparameter settings.

To conclude, I would like to outline two exciting directions for future research

Multi-task offline RL: ARMOR enables robust policy learning in the single-task setting, however, as robots enter the real-world they are expected to perform a diverse set of downstream tasks. As more robotics data becomes available, there is an increasing interest in large scale learning of generalist multi-task policies [14, 356]. The application of offline RL to this problem is promising as it can enable us to leverage diverse data sources that are more commonly available in robotics [443]. However, a key question to be addressed is exploration. A generalist agent must not only perform well on data it has seen before, but also be able to explore to gather information when faced with novel situations. Efficiently combining offline RL with exploration can enable robots to gather their own training data in an autonomous fashion, and is an exciting avenue for interesting research.

Integrating complex perceptual models with MPC: MPC approaches, while strong, are usually restricted by the kinds of representations and task descriptions they can use. MPC algorithms often assume access to accurate state estimates and physics-based models which might not always be available as environments get more complex. They also require complex tasks to be described in terms of cost functions requiring domain expertise. Recent works on learning world models directly from sensory inputs [177] and using natural language description of tasks to specify reward functions [531] offer a promising path forward. While in our work on STORM we take a first step in showing how learned collision checking model can be leveraged in sampling-based MPC, deeply integrating more complex perceptual models with MPC optimizers is interesting an future direction that can enable robots to operate across a much broader range of environments.

BIBLIOGRAPHY

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[2] Yasin Abbasi-Yadkori, Peter Bartlett, Kush Bhatia, Nevena Lazic, Csaba Szepesvari, and Gellért Weisz. Politex: Regret bounds for policy iteration using expert prediction. In *International Conference on Machine Learning*, pages 3692–3702, 2019.

[3] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[4] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19, 2006.

[5] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[6] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[7] Pieter Abbeel and Andrew Y Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 1–8, 2005.

[8] Abbas Abdolmaleki, Jost Tobias Springenberg, Jonas Degrave, Steven Bohez, Yuval Tassa, Dan Belov, Nicolas Heess, and Martin Riedmiller. Relative entropy regularized policy iteration. *arXiv preprint arXiv:1812.02256*, 2018.

[9] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

[10] Alekh Agarwal, Mikael Henaff, Sham Kakade, and Wen Sun. Pc-pg: Policy cover directed exploration for provable policy gradient learning. *Advances in neural information processing systems*, 33:13399–13412, 2020.

[11] Alekh Agarwal, Sham Kakade, Akshay Krishnamurthy, and Wen Sun. Flambe: Structural complexity and representation learning of low rank mdps. *Advances in Neural Information Processing Systems*, 33:20095–20107, 2020.

[12] Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, 22(98):1–76, 2021.

[13] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[14] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[15] Sandip Aine and Maxim Likhachev. Search portfolio with sharing. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.

[16] Sandip Aine, Charupriya Sharma, and Maxim Likhachev. Learning to search more efficiently from experience: A multi-heuristic approach. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

[17] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic a*. In *Seventh Annual Symposium on Combinatorial Search*, 2014.

[18] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic a. *The International Journal of Robotics Research*, 2016.

[19] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[20] Matthias Althoff, Olaf Stursberg, and Martin Buss. Model-based probabilistic collision detection in autonomous driving. *IEEE ITS*, 2009.

[21] Kalyan Vasudev Alwala and Mustafa Mukadam. Joint sampling and trajectory optimization over graphs for online motion planning. *arXiv preprint arXiv:2011.07171*, 2020.

[22] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, pages 8299–8310, 2018.

[23] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 136–145. JMLR. org, 2017.

[24] Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems 29*, pages 3981–3989. 2016.

[25] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pages 5360–5370, 2017.

[26] András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

[27] Shahab Jabbari Arfaee, Sandra Zilles, and Robert C Holte. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 2011.

[28] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[29] Sankalp Arora and Sebastian Scherer. Randomized algorithm for informative path planning with budget constraints. In *ICRA*, 2017.

[30] Oktay Arslan and Panagiotis Tsiotras. Machine learning guided exploration for sampling-based motion planning algorithms. In *IROS*, 2015.

[31] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[32] John Asmuth and Michael L Littman. Approaching bayes-optimalilty using monte-carlo tree search. In *Proc. 21st Int. Conf. Automat. Plan. Sched., Freiburg, Germany*, 2011.

[33] Bernardo Ávila Pires, Csaba Szepesvari, and Mohammad Ghavamzadeh. Cost-sensitive multiclass classification risk bounds. In *ICML*, 2013.

[34] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[35] Moses Bangura and Robert Mahony. Real-time model predictive control for quadrotors. *IFAC Proceedings Volumes*, 47(3):11773–11780, 2014.

[36] Tim D Barfoot, Chi Hay Tong, and Simo Särkkä. Batch continuous-time trajectory estimation as exactly sparse gaussian process regression. Citeseer, 2014.

[37] Shai Ben-David, Nicolo Cesa-Bianchi, David Haussler, and Philip M Long. Characterizations of learnability for classes of $\{0, \ldots, n\}$-valued functions. *Journal of Computer and System Sciences*, 50(1):74–86, 1995.

[38] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE conference on decision and control*, volume 1, pages 560–564. IEEE, 1995.

[39] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996.

[40] Mohak Bhardwaj, Byron Boots, and Mustafa Mukadam. Differentiable gaussian process motion planning. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 10598–10604. IEEE, 2020.

[41] Mohak Bhardwaj, Sanjiban Choudhury, and Byron Boots. Blending mpc & value function approximation for efficient reinforcement learning. *arXiv preprint arXiv:2012.05909*, 2020.

[42] Mohak Bhardwaj, Sanjiban Choudhury, Byron Boots, and Siddhartha Srinivasa. Leveraging experience in lazy search. *arXiv preprint arXiv:1907.07238*, 2019.

[43] Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian Scherer. Learning heuristic search via imitation. In *CoRL*, 2017.

[44] Mohak Bhardwaj, Ankur Handa, Dieter Fox, and Byron Boots. Information theoretic model predictive q-learning. In *Learning for Dynamics and Control*, pages 840–850, 2020.

[45] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. Storm: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *Conference on Robot Learning*, pages 750–759. PMLR, 2022.

[46] Mohak Bhardwaj, Tengyang Xie, Byron Boots, Nan Jiang, and Ching-An Cheng. Adversarial model for offline reinforcement learning. *arXiv preprint arXiv:2302.11048*, 2023.

[47] Joshua Bialkowski, Michael Otte, and Emilio Frazzoli. Free-configuration biased sampling for motion planning. In *IROS*, 2013.

[48] Joshua Bialkowski, Michael Otte, Sertac Karaman, and Emilio Frazzoli. Efficient collision checking in sampling-based motion planning via safety certificates. *The International Journal of Robotics Research*, 35(7):767–796, 2016.

[49] Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav S Sukhatme. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33(6):1273–1291, 2017.

[50] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *ICRA*, 2000.

[51] Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research*, 30(7):954–966, 2011.

[52] Vivek S Borkar. Stochastic approximation with two time scales. *Systems & Control Letters*, 29(5):291–294, 1997.

[53] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023.

[54] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[55] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[56] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[57] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alex Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *arXiv preprint arXiv:2307.15818*, 2023.

[58] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[59] Niv Buchbinder, Moran Feldman, Joseph Seffi, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing*, 2015.

[60] Ruben Burger, Mukunda Bharatheesha, Marc van Eert, and Robert Babuška. Automated tuning and configuration of path planning algorithms. In *2017 IEEE Inter-*

national Conference on Robotics and Automation (ICRA), pages 4371–4376. IEEE, 2017.

[61] Brendan Burns and Oliver Brock. Sampling-based motion planning using predictive models. In *ICRA*, 2005.

[62] Arunkumar Byravan, Felix Lceb, Franziska Meier, and Dieter Fox. Se3-pose-nets: Structured deep dynamics models for visuomotor control. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[63] Arunkumar Byravan, Felix Leeb, Franziska Meier, and Dieter Fox. Se3-pose-nets: Structured deep dynamics models for visuomotor planning and control. *arXiv preprint arXiv:1710.00489*, 2017.

[64] Qi Cai, Zhuoran Yang, Chi Jin, and Zhaoran Wang. Provably efficient exploration in policy optimization. In *International Conference on Machine Learning*, pages 1283–1294. PMLR, 2020.

[65] John Canny. *The complexity of robot motion planning*. MIT press, 1988.

[66] José Cano, Yiming Yang, Bruno Bodin, Vijay Nagarajan, and Michael O'Boyle. Automatic parameter tuning of motion planning algorithms. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8103–8109. IEEE, 2018.

[67] Andrea Cassioli, D Di Lorenzo, Marco Locatelli, Fabio Schoen, and Marco Sciandrone. Machine learning for global optimization. *Computational Optimization and Applications*, 2012.

[68] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[69] Venkatesan T Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, Pranjal Awasthi, and Mukesh Mohania. Decision trees for entity identification: Approximation algorithms

and hardness results. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 53–62, 2007.

[70] Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical Science*, pages 273–304, 1995.

[71] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. Learning to search better than your teacher. In *ICML*, 2015.

[72] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. Information-theoretic planning with trajectory optimization for dense 3d mapping. In *RSS*, 2015.

[73] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[74] Chandra Chekuri and Martin Pal. A recursive greedy algorithm for walks in directed graphs. In *FOCS*, 2005.

[75] Jinglin Chen and Nan Jiang. Information-theoretic considerations in batch reinforcement learning. In *International Conference on Machine Learning*, pages 1042–1051, 2019.

[76] Min Chen, Emilio Frazzoli, David Hsu, and Wee Sun Lee. Pomdp-lite for robust robot planning under uncertainty. *arXiv:1602.04875*, 2016.

[77] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6572–6583, 2018.

[78] Xiong-Hui Chen, Yang Yu, Zheng-Mao Zhu, Zhihua Yu, Zhenjun Chen, Chenghe Wang, Yinan Wu, Hongqiu Wu, Rong-Jun Qin, Ruijin Ding, et al. Adversarial counterfactual environment model learning. *arXiv preprint arXiv:2206.04890*, 2022.

[79] Yuxin Chen, S. Hamed Hassani, and Andreas Krause. Near-optimal bayesian active learning with correlated and noisy tests. *CoRR*, abs/1605.07334, 2016.

[80] Yuxin Chen, Shervin Javdani, Amin Karbasi, Drew Bagnell, Siddhartha Srinivasa, and Andreas Krause. Submodular surrogates for value of information. In *AAAI*, 2015.

[81] Yuxin Chen, Shervin Javdani, Amin Karbasi, J. Bagnell, Siddhartha Srinivasa, and Andreas Krause. Submodular surrogates for value of information. In *AAAI*, 2015.

[82] Ching-An Cheng, Andrey Kolobov, and Alekh Agarwal. Policy improvement via imitation of multiple oracles. *Advances in Neural Information Processing Systems*, 33, 2020.

[83] Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. Heuristic-guided reinforcement learning. *Advances in Neural Information Processing Systems*, 34:13550–13563, 2021.

[84] Ching-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan Ratliff. RMPflow: A computational graph for automatic motion policy generation. In *The 13th International Workshop on the Algorithmic Foundations of Robotics*, 2018.

[85] Ching-An Cheng, Tengyang Xie, Nan Jiang, and Alekh Agarwal. Adversarially trained actor critic for offline reinforcement learning. *International Conference on Machine Learning*, 2022.

[86] Ching-An Cheng, Xinyan Yan, Nathan Ratliff, and Byron Boots. Predictor-corrector policy optimization. In *International Conference on Machine Learning*, pages 1151–1161. PMLR, 2019.

[87] Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. Fast policy learning through imitation and reinforcement. *arXiv preprint arXiv:1805.10413*, 2018.

[88] S. Choudhury, S. Javdani, S. Srinivasa, and S. Scherer. Near-optimal edge evaluation

in explicit generalized binomial graphs. In *Advances in neural information processing systems*, 2017.

[89] S. Choudhury, S.S. Srinivasa, and S. Scherer. Bayesian active edge evaluation on expensive graphs. In *IJCAI*, 2018.

[90] Sanjiban Choudhury. Learning to gather information via imitation: Proofs. goo.gl/GJfg7r, 2016.

[91] Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety. In *AHS 70th Annual Forum*, 2014.

[92] Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety. In *AHS 70th Annual Forum, Montreal, Quebec, Canada*, 2014.

[93] Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. The planner ensemble: Motion planning by executing diverse algorithms. In *IEEE Conference on Robotics and Automation*, page 2389 – 2395, May 2015.

[94] Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. The Planner Ensemble: Motion planning by executing diverse algorithms. In *IEEE International Conference on Robotics and Automation, ICRA*, 2015.

[95] Sanjiban Choudhury, Mohak Bhardwaj, Sankalp Arora, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. Data-driven planning via imitation learning. *IJRR*, 2017.

[96] Sanjiban Choudhury, Jonathan D. Gammell, Timothy D. Barfoot, Siddhartha Srinivasa, and Sebastian Scherer. Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning. In *ICRA*, 2016.

[97] Sanjiban Choudhury, Shervin Javdani, Siddhartha Srinivasa, and Sebastian Scherer. Near-optimal edge evaluation in explicit generalized binomial graphs. In *NIPS*, 2017.

[98] Sanjiban Choudhury, Shervin JAvdani, Siddhartha Srinivasa, and Sebastian Scherer. Near-optimal edge evaluation in explicit generalized binomial graphs. *Arxiv*, 2017.

[99] Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, and Debadeepta Dey. Learning to gather information via imitation. In *ICRA*, 2017.

[100] Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, and Debadeepta Dey. Adaptive information gathering via imitation learning. In *RSS*, 2017.

[101] Shushman Choudhury, Christopher M Dellin, and Siddhartha S Srinivasa. Pareto-optimal search over configuration space beliefs for anytime motion planning. In *IROS*, 2016.

[102] Shushman Choudhury, Oren Salzman, Sanjiban Choudhury, and Siddhartha S Srinivasa. Densification strategies for anytime motion planning over large dense roadmaps. In *ICRA*, 2017.

[103] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[104] Ronald Clark, Michael Bloesch, Jan Czarnowski, Stefan Leutenegger, and Andrew J Davison. Learning to solve nonlinear least squares for monocular stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 284–299, 2018.

[105] Benjamin Cohen, Mike Phillips, and Maxim Likhachev. Planning single-arm manipulations with n-arm robots. In *Eigth Annual Symposium on Combinatorial Search*, 2015.

[106] William W Cohen and Vitor R Carvalho. Stacked sequential learning. In *International Joint Conference on Artificial Intelligence*. Citeseer, 2005.

[107] Lin Cong, Michael Görner, Philipp Ruppel, Hongzhuo Liang, Norman Hendrich, and Jianwei Zhang. Self-adapting recurrent models for object pushing from learning in simulation. *arXiv preprint arXiv:2007.13421*, 2020.

[108] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*, pages 82–90, 2006.

[109] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[110] Hugh Cover, Sanjiban Choudhury, Sebastian Scherer, and Sanjiv Singh. Sparse tangential network (spartan): Motion planning for micro aerial vehicles. In *ICRA*. IEEE, 2013.

[111] Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song. Sbeed: Convergent reinforcement learning with nonlinear function approximation. In *International Conference on Machine Learning*, pages 1125–1134, 2018.

[112] Michael Danielczuk, Arsalan Mousavian, Clemens Eppner, and Dieter Fox. Object rearrangement using learned implicit collision functions. *arXiv preprint arXiv:2011.10726*, 2020.

[113] Amit Daniely, Sivan Sabato, Shai Ben-David, and Shai Shalev-Shwartz. Multiclass learnability and the erm principle. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 207–232. JMLR Workshop and Conference Proceedings, 2011.

[114] Christoph Dann, Gerhard Neumann, Jan Peters, et al. Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research*, 15:809–883, 2014.

[115] Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *NIPS*, 2004.

[116] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[117] Todor Davchev, Oleg Olegovich Sushkov, Jean-Baptiste Regli, Stefan Schaal, Yusuf Aytar, Markus Wulfmeier, and Jon Scholz. Wish you were here: Hindsight goal selec-

tion for long-horizon dexterous manipulation. In *International Conference on Learning Representations*, 2021.

[118] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[119] Frank Dellaert and Michael Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006.

[120] Christopher Dellin. *Completing Manipulation Tasks Efficiently in Complex Environments*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, September 2016.

[121] Christopher M Dellin and Siddhartha S Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *ICAPS*, 2016.

[122] Christopher M Dellin, Kyle Strabala, G Clark Haynes, David Stager, and Siddhartha S Srinivasa. Guided manipulation planning at the darpa robotics challenge trials. In *Experimental Robotics*, 2016.

[123] Jory Denny, Marco Morales, Samuel Rodriguez, and Nancy M Amato. Adapting rrt growth for heterogeneous environments. In *IROS*, 2013.

[124] Vishnu R Desaraju and Nathan Michael. Fast nonlinear model predictive control via partial enumeration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1243–1248. IEEE, 2016.

[125] Debadeepta Dey, Tian Yu Liu, Martial Hebert, and J Andrew Bagnell. Contextual sequence prediction with application to control library optimization. *Robotics*, 2013.

[126] Rosen Diankov and James Kuffner. Randomized statistical path planning. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1–6. IEEE, 2007.

[127] J. Dick, F. Y. Kuo, and I. H. Sloan. High-dimensional integration: The quasi-monte carlo way. *Acta Numerica*, 22:133–288, 2013.

[128] Alexander Dietrich, Thomas Wimbock, Alin Albu-Schaffer, and Gerd Hirzinger. Reactive whole-body control: Dynamic mobile manipulation using a large number of actuated degrees of freedom. *IEEE Robotics & Automation Magazine*, 19(2):20–33, 2012.

[129] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *AAAI*, 2008.

[130] Jing Dong, Mustafa Mukadam, Byron Boots, and Frank Dellaert. Sparse gaussian processes on matrix lie groups: A unified framework for optimizing continuous-time trajectories. In *Proceedings of the 2018 IEEE Conference on Robotics and Automation (ICRA)*, 2018.

[131] Jing Dong, Mustafa Mukadam, Frank Dellaert, and Byron Boots. Motion planning as probabilistic inference using Gaussian processes and factor graphs. In *Proceedings of Robotics: Science and Systems (RSS-2016)*, 2016.

[132] Avner Dor. The greedy search algorithm on binary vectors. *Journal of Algorithms*, 27(1):42–60, 1998.

[133] Avner Dor and Eitan Greenshtein. An almost-greedy search on random binary vectors and random graphs. *Journal of Algorithms*, 40(1):102–133, 2001.

[134] Anca Dragan, Geoffrey J Gordon, and Siddhartha Srinivasa. Learning from experience in manipulation planning: Setting the right goals. 2011.

[135] Paul Drews, Grady Williams, Brian Goldfain, Evangelos A Theodorou, and James M

Rehg. Aggressive deep driving: Combining convolutional neural networks and model predictive control. In *Conference on Robot Learning*, pages 133–142. PMLR, 2017.

[136] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[137] Yaqi Duan, Zeyu Jia, and Mengdi Wang. Minimax-optimal off-policy evaluation with linear function approximation. In *International Conference on Machine Learning*, pages 2701–2709. PMLR, 2020.

[138] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. 1957.

[139] Vishal Dugar, Sanjiban Choudhury, and Sebastian Scherer. A κite in the wind: Smooth trajectory optimization in a moving reference frame. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 109–116. IEEE, 2017.

[140] Vishal Dugar, Sanjiban Choudhury, and Sebastian Scherer. Smooth trajectory optimization in wind: First results on a full-scale helicopter. In *AHS International 73rd Annual Forum, Forth Worth, Texas, USA*, volume 1, 2017.

[141] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov. An integrated system for real-time model predictive control of humanoid robots. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 292–299, 2013.

[142] Tom Erez, Kendall Lowrey, Yuval Tassa, Vikash Kumar, Svetoslav Kolev, and Emanuel Todorov. An integrated system for real-time model predictive control of humanoid robots. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 292–299. IEEE, 2013.

[143] Eyal Even-Dar, Sham M Kakade, and Yishay Mansour. Online markov decision processes. *Mathematics of Operations Research*, 34(3):726–736, 2009.

[144] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza. Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5774–5781, 2017.

[145] Dixia Fan, Liu Yang, Zhicheng Wang, Michael S. Triantafyllou, and George Em Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proceedings of the National Academy of Sciences*, 117(42):26091–26098, 2020.

[146] Amir Massoud Farahmand, Rémi Munos, and Csaba Szepesvári. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, 2010.

[147] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018.

[148] Uriel Feige. A threshold of ln n for approximating set cover. *JACM*, 1998.

[149] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[150] Adam Fishman, Chris Paxton, Wei Yang, Dieter Fox, Byron Boots, and Nathan Ratliff. Collaborative Behavior Models for Optimized Human-Robot Teamwork. *arXiv e-prints*, page arXiv:1910.04339, October 2019.

[151] Dylan J Foster, Claudio Gentile, Mehryar Mohri, and Julian Zimmert. Adapting to misspecification in contextual bandits. *Advances in Neural Information Processing Systems*, 33:11478–11489, 2020.

[152] Roy Fox, Ari Pakman, and Naftali Tishby. Taming the noise in reinforcement learning via soft updates. *arXiv preprint arXiv:1512.08562*, 2015.

[153] Janick V Frasch, Andrew Gray, Mario Zanon, Hans Joachim Ferreau, Sebastian Sager, Francesco Borrelli, and Moritz Diehl. An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. In *2013 European Control Conference (ECC)*, pages 4136–4141. IEEE, 2013.

[154] Alan Frieze and Michał Karoński. *Introduction to random graphs*. Cambridge Press, 2015.

[155] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[156] Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4019–4026. IEEE, 2016.

[157] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep whole-body control: Learning a unified policy for manipulation and locomotion. In *Conference on Robot Learning (CoRL)*, 2022.

[158] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep whole-body control: learning a unified policy for manipulation and locomotion. In *Conference on Robot Learning*, pages 138–149. PMLR, 2023.

[159] Zipeng Fu, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Minimizing energy consumption leads to the emergence of gaits in legged robots. *arXiv preprint arXiv:2111.01674*, 2021.

[160] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34:20132–20145, 2021.

[161] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[162] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062, 2019.

[163] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 3067–3074, 2015.

[164] Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to rank for synthesizing planning heuristics.

[165] Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. A theory of regularized markov decision processes. In *International Conference on Machine Learning*, pages 2160–2169. PMLR, 2019.

[166] Daniel Golovin and Andreas Krause. Adaptive submodularity: A new approach to active learning and stochastic optimization. *CoRR*, abs/1003.3967, 2010.

[167] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 2011.

[168] Daniel Golovin, Andreas Krause, and Debajyoti Ray. Near-optimal bayesian active learning with noisy observations. In *NIPS*, 2010.

[169] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[170] Geoffrey J Gordon. Stable function approximation in dynamic programming. In *Machine Learning Proceedings 1995*, pages 261–268. Elsevier, 1995.

[171] Andrew Guillory and Jeff A. Bilmes. Interactive submodular set cover. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.

[172] Anupam Gupta, Viswanath Nagarajan, and R Ravi. Approximation algorithms for optimal decision trees and adaptive tsp problems. In *International Colloquium on Automata, Languages, and Programming*, 2010.

[173] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017.

[174] Nico Gürtler, Felix Widmaier, Cansu Sancaktar, Sebastian Blaes, Pavel Kolev, Stefan Bauer, Manuel Wüthrich, Markus Wulfmeier, Martin Riedmiller, Arthur Allshire, et al. Real robot challenge 2022: Learning dexterous manipulation from offline data in the real world. In *NeurIPS 2022 Competition Track*, pages 133–150. PMLR, 2022.

[175] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org, 2017.

[176] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[177] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[178] Nika Haghtalab, Simon Mackenzie, Ariel D. Procaccia, Oren Salzman, and Siddhartha S. Srinivasa. The Provable Virtue of Laziness in Motion Planning. In *ICAPS*, pages 106–113, 2018.

[179] J. H. Halton and G. B. Smith. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications ACM*, 7(12):701–702, 1964.

[180] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar

Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. *arXiv preprint arXiv:2210.13702*, 2022.

[181] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[182] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics*, 1968.

[183] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *ICRA*, 2015.

[184] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[185] David Haussler. Sphere packing numbers for subsets of the boolean n-cube with bounded vapnik-chervonenkis dimension. *Journal of Combinatorial Theory, Series A*, 69(2):217–232, 1995.

[186] David Haussler and Philip M Long. A generalization of sauer's lemma. *Journal of Combinatorial Theory, Series A*, 71(2):219–240, 1995.

[187] Jesse Haviland and Peter Corke. A purely-reactive manipulability-maximising motion controller. *arXiv e-prints*, pages arXiv–2002, 2020.

[188] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

[189] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[190] Lionel Heng, Alkis Gotovos, Andreas Krause, and Marc Pollefeys. Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. In *ICRA*, 2015.

[191] Yale T Herer. Submodularity and the traveling salesman problem. *European journal of operational research*, 1999.

[192] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[193] Todd Hester, Michael Quinlan, and Peter Stone. Generalized model learning for reinforcement learning on a humanoid robot. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2369–2374. IEEE, 2010.

[194] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

[195] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[196] Matthew W Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Nikola Momchev, Danila Sinopalnikov, Piotr Stańczyk, Sabela Ramos, Anton Raichuk, Damien Vincent, et al. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

[197] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 2001.

[198] Francois R Hogan and Alberto Rodriguez. Reactive planar non-prehensile manipulation with hybrid model predictive control. *The International Journal of Robotics Research*, 39(7):755–773, 2020.

[199] Geoffrey A Hollinger, Brendan Englot, Franz S Hover, Urbashi Mitra, and Gaurav S Sukhatme. Active planning for underwater inspection and the benefit of adaptivity. *IJRR*, 2012.

[200] Geoffrey A Hollinger, Urbashi Mitra, and Gaurav S Sukhatme. Active classification: Theory and application to underwater inspection. In *Robotics Research*, pages 95–110. Springer, 2017.

[201] Geoffrey A Hollinger and Gaurav S Sukhatme. Sampling-based motion planning for robotic information gathering. In *RSS*, 2013.

[202] Ronald A Howard. Information value theory. *IEEE Tran. Systems Science Cybernetics*, 1966.

[203] David Hsu, J-C Latcombe, and Stephen Sorkin. Placing a robot manipulator amid obstacles for optimized execution. In *Assembly and Task Planning, 1999.(ISATP'99) Proceedings of the 1999 IEEE International Symposium on*, pages 280–285. IEEE, 1999.

[204] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *ICRA*, 1997.

[205] Eric Huang, Mustafa Mukadam, Zhen Liu, and Byron Boots. Motion planning with graph-based trajectories and Gaussian process inference. In *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[206] Jinwook Huh and Daniel D Lee. Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees. In *ICRA*, 2016.

[207] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[208] Phillip Hyatt and Marc D Killpack. Real-time nonlinear model predictive control of robots using a graphics processing unit. *IEEE Robotics and Automation Letters*, 5(2):1468–1475, 2020.

[209] Phillip Hyatt, Connor S Williams, and Marc D Killpack. Parameterized and gpu-

parallelized real-time model predictive control for high degree of freedom robots. *arXiv preprint arXiv:2001.04931*, 2020.

[210] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. *arXiv preprint arXiv:1709.05448*, 2017.

[211] Ingenieurgemeinschaft IgH GmbH. EtherLab EtherCat Master, 2024.

[212] Koji Ishihara, Takeshi D Itoh, and Jun Morimoto. Full-body optimal control toward versatile and agile behaviors in a humanoid robot. *IEEE Robotics and Automation Letters*, 5(1):119–126, 2019.

[213] Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. Dynamic multi-heuristic a*. In *ICRA*, 2015.

[214] Stefan Isler, Reza Sabzevari, Jeffrey Delmerico, and Davide Scaramuzza. An information gain formulation for active volumetric 3d reconstruction. In *ICRA*, 2016.

[215] Rishabh K Iyer and Jeff A Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *NIPS*, 2013.

[216] Shervin Javdani, Yuxin Chen, Amin Karbasi, Andreas Krause, Drew Bagnell, and Siddhartha Srinivasa. Near optimal bayesian active learning for decision making. In *AISTATS*, 2014.

[217] Shervin Javdani, Yuxin Chen, Amin Karbasi, Andreas Krause, J. Andrew Bagnell, and Siddhartha Srinivasa. Near optimal bayesian active learning for decision making. In *AISTATS*, 2014.

[218] Shervin Javdani, Matthew Klingensmith, J. Andrew Bagnell, Nancy Pollard, and Siddhartha Srinivasa. Efficient touch based localization through submodularity. In *ICRA*, 2013.

[219] Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization. In *RSS*, 2015.

[220] Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, 2013.

[221] Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. Contextual decision processes with low bellman rank are pac-learnable. In *International Conference on Machine Learning*, pages 1704–1713. PMLR, 2017.

[222] Sergio Jiménez, Tomás De La Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. A review of machine learning for automated planning. *The Knowledge Engineering Review*, 2012.

[223] Chi Jin, Zhuoran Yang, Zhaoran Wang, and Michael I Jordan. Provably efficient reinforcement learning with linear function approximation. In *Conference on Learning Theory*, pages 2137–2143. PMLR, 2020.

[224] Ying Jin, Zhuoran Yang, and Zhaoran Wang. Is pessimism provably efficient for offline rl? In *International Conference on Machine Learning*, pages 5084–5096. PMLR, 2021.

[225] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[226] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 1998.

[227] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *ICRA*, 2017.

[228] Sham Kakade, Michael J Kearns, and John Langford. Exploration in metric state spaces. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 306–312, 2003.

[229] Sham Kakade and John Langford. Approximately optimal approximate reinforce-

ment learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274. Morgan Kaufmann Publishers Inc., 2002.

[230] Sham M Kakade. A natural policy gradient. *Advances in Neural Information Processing Systems*, 14, 2001.

[231] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011.

[232] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *ICRA*, 2011.

[233] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[234] D. Kappler, F. Meier, J. Issac, J. Mainprice, C. G. Cifuentes, M. Wüthrich, V. Berenz, S. Schaal, N. Ratliff, and J. Bohg. Real-time perception meets reactive motion generation. *IEEE Robotics and Automation Letters*, 3(3):1864–1871, 2018.

[235] Daniel Kappler, Franziska Meier, Jan Issac, Jim Mainprice, Cristina Garcia Cifuentes, Manuel Wüthrich, Vincent Berenz, Stefan Schaal, Nathan Ratliff, and Jeannette Bohg. Real-time perception meets reactive motion generation. *IEEE Robotics and Automation Letters*, 3(3):1864–1871, 2018.

[236] Sertac Karaman. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104(2).

[237] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[238] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483. IEEE, 2011.

[239] Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. *arXiv preprint arXiv:1703.06692*, 2017.

[240] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[241] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, 1996.

[242] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

[243] Michael J Kearns, Yishay Mansour, and Andrew Y Ng. Approximate planning in large pomdps via reusable trajectories. In *Advances in Neural Information Processing Systems*, pages 1001–1007, 2000.

[244] A. Kelly and B. Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *International Journal of Robotics Research*, 22(7-8):583–601, 2003.

[245] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement llearning. In *Advances in Neural Information Processing Systems*, 2020.

[246] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[247] Charles A Klein and Bruce E Blaho. Dexterity measures for the design and control of kinematically redundant manipulators. *The international journal of robotics research*, 6(2):72–83, 1987.

[248] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[249] Marin Kobilarov. Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871, 2012.

[250] Marin Kobilarov. Cross-entropy randomized motion planning. In *Robotics: Science and Systems*, volume 7, pages 153–160, 2012.

[251] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

[252] Jonas Koenemann, Andrea Del Prete, Yuval Tassa, Emanuel Todorov, Olivier Stasse, Maren Bennewitz, and Nicolas Mansard. Whole-body model-predictive control applied to the hrp-2 humanoid. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3346–3351. IEEE, 2015.

[253] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014. Citeseer, 2000.

[254] Igor Kononenko. Machine learning for medical diagnosis: History, state of the art and perspective. *Artificial Intelligence in Medicine*, 2001.

[255] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[256] Michael Koval, Nancy Pollard, and Siddhartha Srinivasa. Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty. In *RSS*, 2014.

[257] Michael C Koval, Nancy S Pollard, and Siddhartha S Srinivasa. Pre-and post-contact policy decomposition for planar contact manipulation under uncertainty. *The International Journal of Robotics Research*, 35(1-3):244–264, 2016.

[258] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 2012.

[259] Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, 2007.

[260] Andreas Krause and Carlos Guestrin. Optimal value of information in graphical models. *Journal of Artificial Intelligence Research*, 35:557–591, 2009.

[261] Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 2008.

[262] James J Kuffner and Steven M LaValle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation, ICRA*, 2000.

[263] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[264] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32:11784–11794, 2019.

[265] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[266] Vikash Kumar, Yuval Tassa, Tom Erez, and Emanuel Todorov. Real-time behaviour synthesis for dynamic hand-manipulation. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6808–6815. IEEE, 2014.

[267] Alan Kuntz, Chris Bowen, and Ron Alterovitz. Fast anytime motion planning in point clouds by interleaving sampling and interior point optimization. In *Robotics Research*, pages 929–945. Springer, 2020.

[268] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *RSS*, 2008.

[269] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[270] Bakir Lacevic, Dinko Osmankovic, and Adnan Ademovic. Burs of free c-space: a novel structure for path planning. In *ICRA*. IEEE, 2016.

[271] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[272] Alexander Lambert, Mustafa Mukadam, Balakumar Sundaralingam, Nathan Ratliff, Byron Boots, and Dieter Fox. Joint inference of kinematic and force trajectories with visuo-tactile sensing. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, 2019.

[273] Nathan Lambert, Markus Wulfmeier, William Whitney, Arunkumar Byravan, Michael Bloesch, Vibhavari Dasagi, Tim Hertweck, and Martin Riedmiller. The challenges of exploration for offline reinforcement learning. *arXiv preprint arXiv:2201.11861*, 2022.

[274] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. *Reinforcement learning: State-of-the-art*, pages 45–73, 2012.

[275] John Langford and Alina Beygelzimer. Sensitive error correcting output codes. In *COLT*, volume 3559, pages 158–172. Springer, 2005.

[276] A Larcher and E Hachem. A review on deep reinforcement learning for fluid mechanics: an update. *Physics of Fluids*, 34(11):111301, 2022.

[277] Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pages 3652–3661. PMLR, 2019.

[278] Jean-Paul Laumond, S Sekhavat, and F Lamiraux. Guidelines in nonholonomic motion planning for mobile robots. In *Robot motion planning and control*, pages 1–53. Springer, 1998.

[279] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

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

[281] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *IJRR*, 2001.

[282] Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Finite-sample analysis of least-squares policy iteration. *Journal of Machine Learning Research*, 13:3041–3074, 2012.

[283] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. Data center cooling using model-predictive control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[284] Alex X Lee, Coline Manon Devin, Yuxiang Zhou, Thomas Lampe, Konstantinos Bousmalis, Jost Tobias Springenberg, Arunkumar Byravan, Abbas Abdolmaleki, Nimrod Gileadi, David Khosid, et al. Beyond pick-and-place: Tackling robotic stacking of diverse shapes. In *Conference on Robot Learning*, pages 1089–1131. PMLR, 2022.

[285] Alex X. Lee, Coline Manon Devin, Yuxiang Zhou, Thomas Lampe, Konstantinos Bousmalis, Jost Tobias Springenberg, Arunkumar Byravan, Abbas Abdolmaleki, Nimrod Gileadi, David Khosid, Claudio Fantacci, Jose Enrique Chen, Akhil Raju, Rae Jeong, Michael Neunert, Antoine Laurens, Stefano Saliceti, Federico Casarini, Martin Riedmiller, raia hadsell, and Francesco Nori. Beyond pick-and-place: Tackling robotic stacking of diverse shapes. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1089–1131. PMLR, 08–11 Nov 2022.

[286] Gilwoo Lee, Brian Hou, Sanjiban Choudhury, and Siddhartha S Srinivasa. Bayesian residual policy optimization: Scalable bayesian reinforcement learning with clairvoyant experts. *arXiv preprint arXiv:2002.03042*, 2020.

[287] Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[288] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

[289] Sergey Levine and Vladlen Koltun. Continuous inverse optimal control with locally optimal examples. In *ICML*, 2012.

[290] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

[291] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[292] Anqi Li, Mustafa Mukadam, Magnus Egerstedt, and Byron Boots. Multi-objective policy generation for multi-robot systems using riemannian motion policies. *arXiv preprint arXiv:1902.05177*, 2019.

[293] Hui Li, Xuejun Liao, and Lawrence Carin. Multi-task reinforcement learning in partially observable stochastic environments. *Journal of Machine Learning Research*, 2009.

[294] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[295] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, 2010.

[296] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement llearning, 2018.

[297] Andy Liaw and Matthew Wiener. Classification and regression by randomforest.

[298] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 2009.

[299] 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.

[300] 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.

[301] Zhan Wei Lim, David Hsu, and Wee Sun Lee. Adaptive stochastic optimization: From sets to paths. In *NIPS*. 2015.

[302] Zhan Wei Lim, David Hsu, and Wee Sun Lee. Adaptive informative path planning in metric spaces. *IJRR*, 2016.

[303] Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[304] Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 1994.

[305] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML*, 1995.

[306] Michael L Littman and Richard S Sutton. Predictive representations of state. In *Advances in neural information processing systems*, pages 1555–1561, 2002.

[307] Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. An mpc algorithm with combined speed and steering control for obstacle avoidance in autonomous ground vehicles. In *ASME 2015 dynamic systems and control conference*. American Society of Mechanical Engineers Digital Collection, 2015.

[308] Miao Liu, Xuejun Liao, and Lawrence Carin. Online expectation maximization for reinforcement learning in pomdps. In *IJCAI*, 2013.

[309] Qinghua Liu, Alan Chung, Csaba Szepesvári, and Chi Jin. When is partially observable reinforcement learning not scary? In *Conference on Learning Theory*, volume 178, pages 5175–5220. PMLR, 2022.

[310] Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Off-policy policy gradient with stationary distribution correction. In *Uncertainty in Artificial Intelligence*, pages 1180–1190. PMLR, 2020.

[311] Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Provably good batch off-policy reinforcement learning without great exploration. *Advances in Neural Information Processing Systems*, 33:1264–1274, 2020.

[312] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*, 2018.

[313] Jingru Luo and Kris Hauser. Robust trajectory optimization under frictional contact with iterative learning. *Autonomous Robots*, 41(6):1447–1461, 2017.

[314] Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(4):1–11, 2018.

[315] Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey, and Maxim Likhachev. Path planning for noncircular microaerial vehicles in constrained environments. In *ICRA*, 2013.

[316] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5–34, 2003.

[317] Hamid Reza Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *NIPS*, pages 1204–1212, 2009.

[318] Aditya Mahajan and Demosthenis Teneketzis. Multi-armed bandit problems. In *Foundations and Applications of Sensor Management*. 2008.

[319] Aditya Mandalika, Oren Salzman, and Siddhartha Srinivasa. Lazy Receding Horizon A* for Efficient Path Planning in Graphs with Expensive-to-Evaluate Edges. In *ICAPS*, pages 476–484, 2018.

[320] David Q Mayne, Erric C Kerrigan, EJ Van Wyk, and Paola Falugi. Tube-based robust nonlinear model predictive control. *International Journal of Robust and Nonlinear Control*, 21(11):1341–1353, 2011.

[321] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[322] David A McAllester and Satinder Singh. Approximate planning for factored pomdps using belief state simplification. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 409–416. Morgan Kaufmann Publishers Inc., 1999.

[323] Tad McGeer. Passive Dynamic Walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990.

[324] Nik A Melchior and Reid Simmons. Particle rrt for path planning with uncertainty. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1617–1624. IEEE, 2007.

[325] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020.

[326] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[327] 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.

[328] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[329] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[330] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[331] Marco Morales, Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*. 2005.

[332] Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5307–5314. IEEE, 2015.

[333] Mustafa Mukadam, Ching An Cheng, Xinyan Yan, and Byron Boots. Approximately

optimal continuous-time motion planning and control via probabilistic inference. In *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[334] Mustafa Mukadam, Jing Dong, Frank Dellaert, and Byron Boots. Simultaneous trajectory estimation and planning via probabilistic inference. In *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

[335] Mustafa Mukadam, Jing Dong, Frank Dellaert, and Byron Boots. STEAP: simultaneous trajectory estimation and planning. In *Autonomous Robots (AURO)*, volume 43, pages 415–434, 2018.

[336] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time Gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research (IJRR)*, 37(11):1319–1340, 2018.

[337] Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9–15, 2016.

[338] Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 1997.

[339] Rémi Munos. Error bounds for approximate policy iteration. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, pages 560–567, 2003.

[340] Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(5), 2008.

[341] Sean Murray, Will Floyd-Jones, Ying Qi, Daniel J Sorin, and George Dimitri Konidaris. Robot motion planning on a chip. In *Robotics: Science and Systems*, 2016.

[342] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[343] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[344] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[345] Jun Nakanishi, Rick Cory, Michael Mistry, Jan Peters, and Stefan Schaal. Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757, 2008.

[346] Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. Improved multi-heuristic a* for searching with uncalibrated heuristics. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

[347] Venkatraman Narayanan and Maxim Likhachev. Heuristic search on graphs with existence priors for expensive-to-evaluate edges. In *ICAPS*, 2017.

[348] Erik Nelson and Nathan Michael. Information-theoretic occupancy grid compression for high-speed information-based exploration. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4976–4982. IEEE, 2015.

[349] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 1978.

[350] Gergely Neu, Anders Jonsson, and Vicenç Gómez. A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798*, 2017.

[351] Andrew Y Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

[352] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*, volume 63. Society for Industrial and Applied Mathematics, Philadelphia, US, 1992.

[353] Christian L Nielsen and Lydia E Kavraki. A 2 level fuzzy prm for manipulation planning. In *IROS*, 2000.

[354] Guido Novati, Hugues Lascombes de Laroussilhe, and Petros Koumoutsakos. Automating turbulence modelling by multi-agent reinforcement learning. *Nature Machine Intelligence*, 3(1):99–99, 2021.

[355] Robert Nowak. Generalized binary search. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, 2008.

[356] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. https://octo-models.github.io, 2023.

[357] Masashi Okada and Tadahiro Taniguchi. Variational inference mpc for bayesian model-based reinforcement learning. In *Conference on Robot Learning*, pages 258–272, 2020.

[358] Ian Osband, Dan Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *NIPS*, 2013.

[359] Brian Paden, Valerio Varricchio, and Emilio Frazzoli. Verification and synthesis of admissible heuristics for kinodynamic motion planning. *IEEE Robotics and Automation Letters*, 2(2):648–655, 2017.

[360] Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.

[361] Jia Pan, Zhuo Chen, and Pieter Abbeel. Predicting initialization effectiveness for trajectory optimization. In *ICRA*.

[362] Jia Pan, Zhuo Chen, and Pieter Abbeel. Predicting initialization effectiveness for trajectory optimization. 2014.

[363] Jia Pan, Sachin Chitta, and Dinesh Manocha. Faster sample-based motion planning using instance-based learning. In *WAFR*. Springer Verlag, 2012.

[364] Sandeep Pandey, Deepayan Chakrabarti, and Deepak Agarwal. Multi-armed bandit problems with dependent arms. In *ICML*, 2007.

[365] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[366] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[367] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

[368] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

[369] 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 1–8. IEEE, 2018.

[370] Mathew Penrose. *Random geometric graphs*. Oxford University Press, 2003.

[371] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *IROS*, 2006.

[372] Jan Peters and Stefan Schaal. Learning to control in operational space. *The International Journal of Robotics Research*, 27(2):197–212, 2008.

[373] Mike Phillips, Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. E-graphs: Bootstrapping planning with experience graphs. 2012.

[374] Mike Phillips, Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. Efficient search with an ensemble of heuristics. In *IJCAI*, 2015.

[375] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. *arXiv preprint arXiv:2008.06389*, 2020.

[376] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[377] Bernardo Ávila Pires and Csaba Szepesvári. Statistical linear estimation with penalized estimators: an application to reinforcement learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1755–1762, 2012.

[378] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[379] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

[380] Ira Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 1970.

[381] Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[382] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Réglade, and Nicolas Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, 2019.

[383] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pages 1177–1184, 2007.

[384] Aravind Rajeswaran*, Vikash Kumar*, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[385] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A game theoretic framework for model based reinforcement learning. In *International Conference on Machine Learning*, pages 7953–7963. PMLR, 2020.

[386] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[387] Fabio Ramos and Lionel Ott. Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent. *IJRR*, 2016.

[388] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv preprint arXiv:1906.01728*, 2019.

[389] M. Asif Rana, Mustafa Mukadam, S. Reza Ahmadzadeh, Sonia Chernova, and Byron Boots. Towards robust skill generalization: Unifying learning from demonstration and motion planning. In *Proceedings of the 2017 Conference on Robot Learning (CoRL)*, 2017.

[390] M. Asif Rana, Mustafa Mukadam, S. Reza Ahmadzadeh, Sonia Chernova, and Byron Boots. Learning generalizable robot skills from demonstrations in cluttered environments. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[391] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

[392] Paria Rashidinejad, Banghua Zhu, Cong Ma, Jiantao Jiao, and Stuart Russell. Bridging offline reinforcement learning and imitation learning: A tale of pessimism. *Advances in Neural Information Processing Systems*, 34:11702–11716, 2021.

[393] Nathan Ratliff, Matthew Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *ICRA*, 2009.

[394] Nathan D Ratliff, Jan Issac, Daniel Kappler, Stan Birchfield, and Dieter Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.

[395] Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

[396] Gautam Reddy, Antonio Celani, Terrence J Sejnowski, and Massimo Vergassola. Learning to soar in turbulent environments. *Proceedings of the National Academy of Sciences*, 113(33):E4877–E4884, 2016.

[397] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[398] Marc Rigter, Bruno Lacerda, and Nick Hawes. Rambo-rl: Robust adversarial model-based offline reinforcement learning. *Advances in Neural Information Processing Systems*, 35:16082–16097, 2022.

[399] Ugo Rosolia and Francesco Borrelli. Learning model predictive control for iterative

tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 63(7):1883–1896, 2017.

[400] Stephane Ross and J Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. *arXiv preprint arXiv:1203.1007*, 2012.

[401] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv*, 2014.

[402] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011.

[403] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for pomdps. *JAIR*, 2008.

[404] Stephane Ross, Jiaji Zhou, Yisong Yue, Debadeepta Dey, and J Andrew Bagnell. Learning policies for contextual submodular prediction. *JMLR*, 2013.

[405] RY Rubinstein and DP Kroese. The cross-entropy method: A unified approach to combinatorial optimization, monte-carlo simulation and machine learning. 2004.

[406] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[407] Wheeler Ruml and Minh Binh Do. Best-first utility-guided search. In *IJCAI*, 2007.

[408] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[409] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[410] Bruno Scherrer. Approximate policy iteration schemes: a comparison. In *International Conference on Machine Learning*, pages 1314–1322. PMLR, 2014.

[411] Ralf Schoknecht and Artur Merke. Td (0) converges provably faster than the residual gradient algorithm. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 680–687, 2003.

[412] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[413] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[414] John Schulman, Xi Chen, and Pieter Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

[415] John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *RSS*, 2013.

[416] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[417] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[418] 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.

[419] 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.

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

[421] Devin Schwab, Jost Tobias Springenberg, Murilo Fernandes Martins, Michael Neunert, Thomas Lampe, Abbas Abdolmaleki, Tim Hertweck, Roland Hafner, Francesco Nori, and Martin A. Riedmiller. Simultaneously learning vision and feature-based control policies for real-world ball-in-a-cup. In Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson, editors, *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, 2019.

[422] Jacob T Schwartz and Micha Sharir. On the "piano movers'" problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on pure and applied mathematics*, 36(3):345–398, 1983.

[423] N. Scianca, D. De Simone, L. Lanari, and G. Oriolo. Mpc for humanoid gait generation: Stability and feasibility. *IEEE Transactions on Robotics*, 36(4):1171–1188, 2020.

[424] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[425] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 2013.

[426] Lior Shani, Yonathan Efroni, Aviv Rosenberg, and Shie Mannor. Optimistic policy optimization with bandit feedback. In *International Conference on Machine Learning*, pages 8604–8613. PMLR, 2020.

[427] Laixi Shi, Gen Li, Yuting Wei, Yuxin Chen, and Yuejie Chi. Pessimistic q-learning for offline reinforcement learning: Towards optimal sample complexity. In *International Conference on Machine Learning*, pages 19967–20025. PMLR, 2022.

[428] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International Conference on Machine Learning*, pages 5779–5788, 2019.

[429] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*, 2020.

[430] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[431] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[432] David Silver, Richard S Sutton, and Martin Müller. Temporal-difference search in computer go. *Machine learning*, 87(2):183–219, 2012.

[433] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *NIPS*, 2010.

[434] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[435] Marwaan Simaan and Jose B Cruz. On the stackelberg strategy in nonzero-sum games. *Journal of Optimization Theory and Applications*, 11(5):533–555, 1973.

[436] Amarjeet Singh, Andreas Krause, Carlos Guestrin, William Kaiser, and Maxim Batalin. Efficient planning of informative paths for multiple robots. In *IJCAI*, 2007.

[437] Amarjeet Singh, Andreas Krause, and William J. Kaiser. Nonmyopic adaptive informative path planning for multiple robots. In *IJCAI*, 2009.

[438] Satinder P Singh and Richard C Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

[439] Satinder P Singh and Richard C Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

[440] Trey Smith and Reid Simmons. Point-based pomdp algorithms: Improved analysis and implementation. *arXiv:1207.1412*, 2012.

[441] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In *NIPS*, 2013.

[442] Matthijs TJ Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for pomdps. *Journal of artificial intelligence research*, 24:195–220, 2005.

[443] Jost Tobias Springenberg, Abbas Abdolmaleki, Jingwei Zhang, Oliver Groth, Michael Bloesch, Thomas Lampe, Philemon Brakel, Sarah Bechtle, Steven Kapturowski, Roland Hafner, et al. Offline actor-critic reinforcement learning scales to large models. *arXiv preprint arXiv:2402.05546*, 2024.

[444] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, pages 4739–4748, 2018.

[445] Matthew Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *NIPS*, 2009.

[446] Freek Stulp and Olivier Sigaud. Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*, 2012.

[447] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IROS*, 2012.

[448] Ioan A Sucan and Sachin Chitta. Moveit!, 2013.

[449] Colin Summers, Kendall Lowrey, Aravind Rajeswaran, Siddhartha Srinivasa, and Emanuel Todorov. Lyceum: An efficient and scalable ecosystem for robot learning. In *Learning for Dynamics and Control*, pages 793–803. PMLR, 2020.

[450] Wen Sun, J Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning & imitation learning. *arXiv preprint arXiv:1805.11240*, 2018.

[451] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *ICML*, 2017.

[452] Balakumar Sundaralingam and Tucker Hermans. Relaxed-Rigidity Constraints: Kinematic Trajectory Optimization and Collision Avoidance for In-Grasp Manipulation. *Autonomous Robots*, 43(2):469–483, February 2019.

[453] Giovanni Sutanto, Austin Wang, Yixin Lin, Mustafa Mukadam, Gaurav Sukhatme, Akshara Rai, and Franziska Meier. Encoding physical constraints in differentiable newton-euler algorithm. volume 120 of *Proceedings of Machine Learning Research*, pages 804–813, The Cloud, 10–11 Jun 2020. PMLR.

[454] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

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

[456] Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000, 2009.

[457] Gokul Swamy, Sanjiban Choudhury, J Andrew Bagnell, and Steven Wu. Of moments and matching: A game-theoretic framework for closing the imitation gap. In *International Conference on Machine Learning*, pages 10022–10032. PMLR, 2021.

[458] Csaba Szepesvári and Rémi Munos. Finite time bounds for sampling based fitted value iteration. In *Proceedings of the 22nd international conference on Machine learning*, pages 880–887, 2005.

[459] Abhijeet Tallavajhula, Sanjiban Choudhury , Sebastian Scherer, and Alonzo Kelly . List prediction applied to motion planning. In *ICRA*, 2016.

[460] Abhijeet Tallavajhula and Sanjiban Choudhury. List prediction for motion planning: Case studies. Technical Report CMU-RI-TR-15-25, Robotics Institute, Pittsburgh, PA, September 2015.

[461] Abhijeet Tallavajhula, Sanjiban Choudhury, Sebastian Scherer, and Alonzo Kelly. List prediction applied to motion planning. In *ICRA*, 2016.

[462] Aviv Tamar, Sergey Levine, and Pieter Abbeel. Value iteration networks. *NIPS*, 2016.

[463] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan–episodic mpc improvement. *arXiv preprint arXiv:1609.09001*, 2016.

[464] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan—episodic mpc improvement. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 336–343. IEEE, 2017.

[465] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[466] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[467] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind control suite. Technical report, DeepMind, January 2018.

[468] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014.

[469] Russ Tedrake, Zack Jackowski, Rick Cory, John William Roberts, and Warren Hoburg. Learning to fly like a bird. Citeseer.

[470] Jordan Tyler Thayer, Austin J Dionne, and Wheeler Ruml. Learning inadmissible heuristics during search. In *ICAPS*, 2011.

[471] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[472] Evangelos A Theodorou and Emanuel Todorov. Relative entropy and free energy dualities: Connections to path integral and kl control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 1466–1473. IEEE, 2012.

[473] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[474] T. Tielman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[475] Dhruva Tirumala, Thomas Lampe, José Enrique Chen, Tuomas Haarnoja, Sandy Huang, Guy Lever, Ben Moran, Tim Hertweck, Leonard Hasenclever, Martin Riedmiller, Nicolas Manfred Otto Heess, and Markus Wulfmeier. Replay across experiments: A natural extension of off-policy rl. 2023.

[476] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[477] Emanuel Todorov. Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, 106(28):11478–11483, 2009.

[478] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[479] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.

[480] Luis G Torres, Alan Kuntz, Hunter B Gilbert, Philip J Swaney, Richard J Hendrick, Robert J Webster, and Ron Alterovitz. A motion planning approach to automatic obstacle avoidance during concentric tube robot teleoperation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2361–2367. IEEE, 2015.

[481] Masatoshi Uehara and Wen Sun. Pessimistic model-based offline reinforcement learning under partial coverage. In *International Conference on Learning Representations*, 2021.

[482] Masatoshi Uehara, Xuezhou Zhang, and Wen Sun. Representation learning for online and offline rl in low-rank mdps. *arXiv preprint arXiv:2110.04652*, 2021.

[483] Jes ús Virseda, Daniel Borrajo, and Vidal Alcázar. Learning heuristic functions for cost-based planning. *Planning and Learning*, 2013.

[484] Nikolaus Vahrenkamp, Tamim Asfour, Giorgio Metta, Giulio Sandini, and Rüdiger Dillmann. Manipulability analysis. In *2012 12th ieee-ras international conference on humanoid robots (humanoids 2012)*, pages 568–573. IEEE, 2012.

[485] Sara A van de Geer. *Empirical Processes in M-estimation*, volume 6. Cambridge university press, 2000.

[486] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[487] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[488] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[489] Arun Venkatraman, Byron Boots, Martial Hebert, and J Andrew Bagnell. Data as demonstrator with applications to system identification. In *ALR Workshop, NIPS*, 2014.

[490] Arun Venkatraman, Martial Hebert, and J Andrew Bagnell. Improving multi-step prediction of learned time series models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[491] Siddhartha Verma, Guido Novati, and Petros Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23):5849–5854, 2018.

[492] Heinrich Von Stackelberg. *Market structure and equilibrium*. Springer Science & Business Media, 2010.

[493] Nolan Wagener, Ching-An Cheng, Jacob Sacks, and Byron Boots. An online learning approach to model predictive control. 2019.

[494] Nolan Wagener, Ching-An Cheng, Jacob Sacks, and Byron Boots. An online learning approach to model predictive control. *arXiv preprint arXiv:1902.08967*, 2019.

[495] Martin J Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge University Press, 2019.

[496] Ruosong Wang, Dean Foster, and Sham M Kakade. What are the statistical limits of offline rl with linear function approximation? In *International Conference on Learning Representations*, 2020.

[497] Ruosong Wang, Russ R Salakhutdinov, and Lin Yang. Reinforcement learning with general value function approximation: Provably efficient approach via bounded eluder dimension. *Advances in Neural Information Processing Systems*, 33:6123–6135, 2020.

[498] Ruosong Wang, Yifan Wu, Ruslan Salakhutdinov, and Sham Kakade. Instabilities of offline rl with pre-trained neural representation. In *International Conference on Machine Learning*, pages 10948–10960. PMLR, 2021.

[499] Zhikang T Wang and Masahito Ueda. A convergent and efficient deep q network algorithm. *arXiv preprint arXiv:2106.15419*, 2021.

[500] Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *Advances in Neural Information Processing Systems*, 33:7768–7778, 2020.

[501] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

[502] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[503] Gellért Weisz, Philip Amortila, and Csaba Szepesvári. Exponential lower bounds for planning in mdps with linearly-realizable optimal action-value functions. In *Algorithmic Learning Theory*, pages 1237–1264. PMLR, 2021.

[504] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017.

[505] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[506] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

[507] Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

[508] Christopher Makoto Wilt and Wheeler Ruml. Building a heuristic for greedy search. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

[509] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[510] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[511] Chenjun Xiao, Yifan Wu, Jincheng Mei, Bo Dai, Tor Lattimore, Lihong Li, Csaba Szepesvari, and Dale Schuurmans. On the optimality of batch policy optimization algorithms. In *International Conference on Machine Learning*, pages 11362–11371. PMLR, 2021.

[512] Tengyang Xie, Ching-An Cheng, Nan Jiang, Paul Mineiro, and Alekh Agarwal. Bellman-consistent pessimism for offline reinforcement learning. *Advances in neural information processing systems*, 34:6683–6694, 2021.

[513] Tengyang Xie and Nan Jiang. Q* approximation schemes for batch reinforcement learning: A theoretical comparison. In *Conference on Uncertainty in Artificial Intelligence*, pages 550–559. PMLR, 2020.

[514] Tengyang Xie and Nan Jiang. Batch value-function approximation with only realizability. In *International Conference on Machine Learning*, pages 11404–11413. PMLR, 2021.

[515] Tengyang Xie, Nan Jiang, Huan Wang, Caiming Xiong, and Yu Bai. Policy finetuning: Bridging sample-efficient offline and online reinforcement learning. *Advances in Neural Information Processing Systems*, 34:27395–27407, 2021.

[516] Jie Xu, Tao Du, Michael Foshey, Beichen Li, Bo Zhu, Adriana Schulz, and Wojciech Matusik. Learning to fly: computational controller design for hybrid uavs with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[517] Yuehua Xu, Alan Fern, and Sung Wook Yoon. Discriminative learning of beam-search heuristics for planning. In *IJCAI*, 2007.

[518] Yuehua Xu, Alan Fern, and Sung Wook Yoon. Iterative learning of weighted rule sets for greedy search. In *ICAPS*, 2010.

[519] Yuehua Xu, Alan Fern, and Sungwook Yoon. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research*, 2009.

[520] Zhenjia Xu, Cheng Chi, Benjamin Burchfiel, Eric Cousineau, Siyuan Feng, and Shuran Song. Dextairity: Deformable manipulation can be a breeze, 2022.

[521] Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani. Data efficient reinforcement learning for legged robots. In *Conference on Robot Learning*, pages 1–10. PMLR, 2020.

[522] Yuxiang Yang, Tingnan Zhang, Erwin Coumans, Jie Tan, and Byron Boots. Fast and efficient locomotion via learned gait transitions. In *Conference on Robot Learning*, pages 773–783. PMLR, 2022.

[523] Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, Pieter Abbeel, Alessandro Lazaric, and Lerrel Pinto. Don't change the algorithm, change the data: Ex-

ploratory data for offline reinforcement learning. *arXiv preprint arXiv:2201.13425*, 2022.

[524] Sung Wook Yoon, Alan Fern, and Robert Givan. Learning heuristic functions from relaxed plans. In *ICAPS*, 2006.

[525] Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, volume 7, pages 352–359, 2007.

[526] Sungwook Yoon, Alan Fern, and Robert Givan. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 2008.

[527] Takayuki Yoshizumi, Teruhisa Miura, and Toru Ishida. A* with partial expansion for large branching factor problems. In *AAAI/IAAI*, pages 923–929, 2000.

[528] Jingjin Yu, Mac Schwager, and Daniela Rus. Correlated orienteering problem and its application to informative path planning for persistent monitoring tasks. In *IROS*, 2014.

[529] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *Advances in Neural Information Processing Systems*, 34:28954–28967, 2021.

[530] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020.

[531] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

[532] Andrea Zanette. Exponential lower bounds for batch reinforcement learning: Batch rl can be exponentially harder than online rl. In *International Conference on Machine Learning*, pages 12287–12297. PMLR, 2021.

[533] Andrea Zanette, Martin J Wainwright, and Emma Brunskill. Provable benefits of actor-critic methods for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[534] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020.

[535] Haifeng Zhang and Yevgeniy Vorobeychik. Submodular optimization with routing constraints. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[536] Siyuan Zhang and Nan Jiang. Towards hyperparameter-free policy selection for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[537] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *ICRA*, 2016.

[538] Tong Zhang. From ε-entropy to kl-entropy: Analysis of minimum information complexity density estimation. *The Annals of Statistics*, 34(5):2180–2210, 2006.

[539] Liyuan Zheng, Tanner Fiez, Zane Alumbaugh, Benjamin Chasnov, and Lillian J Ratliff. Stackelberg actor-critic: Game-theoretic reinforcement learning algorithms. *arXiv preprint arXiv:2109.12286*, 2021.

[540] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov. Value function approximation and model predictive control. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 100–107, 2013.

[541] Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. Value function approximation and model predictive control. In *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, pages 100–107. IEEE, 2013.

[542] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[543] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.

[544] Matthew Zucker. A data-driven approach to high level planning. Technical Report CMU-RI-TR-09-42, Robotics Institute, Pittsburgh, PA, January 2009.