Planning and Execution using Inaccurate Models with Provable Guarantees on Task Completeness

Anirudh Vemula

CMU-RI-TR-22-05
February 2 2022

Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee
Maxim Likhachev, Co-Chair
J. Andrew Bagnell, Co-Chair
Oliver Kroemer
Leslie Pack Kaelbling, *Massachusetts Institute of Technology*

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Keywords: Robotics, Planning, Reinforcement Learning, Numerical Optimization

For my mother, who sacrificed her own dreams to help me pursue mine

Abstract

Modern planning methods are effective in computing feasible and optimal plans for robotic tasks when given access to accurate dynamical models. However, robots operating in the real world often face situations that cannot be modeled perfectly before execution. Thus, we only have access to simplified but potentially inaccurate models. This imperfect modeling can lead to highly suboptimal plans or even the inability to reach the goal during execution. Existing approaches present a learning-based solution where real-world experience is used to learn a complex dynamical model that is subsequently used for planning. However, this requires a prohibitively large amount of experience over the entire state space, and can be wasteful if we are interested in completing the task and not in modeling the dynamics accurately. Furthermore, real robots often have operating constraints and cannot spend hours acquiring experience to learn dynamics. This thesis argues that by updating the behavior of the planner and not the dynamics of the model, we can leverage simplified and potentially inaccurate models and significantly reduce the amount of real-world experience needed to provably guarantee that the robot completes the task.

We support this argument from an algorithmic perspective by presenting two novel algorithms. The first algorithm Cmax guarantees that the robot reaches the goal using the inaccurate model without any resets. This is achieved by biasing the planner away from transitions whose dynamics are discovered to be inaccurately modeled during online execution. However, Cmax requires strong assumptions on the accuracy of the model used for planning and fails to improve the quality of solution over repetitions of the same task. The second algorithm Cmax++ leverages real-world experience to improve the quality of resulting plans over successive repetitions of a robotic task. Cmax++ achieves this by integrating model-free learning using acquired experience with model-based planning using the potentially inaccurate model. As a consequence of this in addition to completeness, Cmax++ also guarantees asymptotic convergence to the optimal path cost as the number of repetitions increases under relaxed assumptions. Crucially, both algorithms do not require any updates to the dynamics of the model unlike any existing method for planning using inaccurate models.

From a theoretical perspective, this thesis presents a performance analysis for methods that leverage inaccurate models in optimal control of linearized systems with quadratic costs. Our analysis shows that naively using inaccurate models can lead to large suboptimality gaps when modeling errors are significant, while updating the behavior of the planner during execution, like Cmax and Cmax++, can substantially reduce the suboptimality gap. The thesis concludes by exploring the paradigm of updating the dynamics of the model and presents an algorithm Toms that directly optimizes task performance rather than prediction error. We show that in the online setting where the robot does not have access to any resets and collects data as it executes, Toms outperforms prior works that either optimize a maximum likelihood objective or rely on an offline collected dataset with good coverage.

Acknowledgments

I believe PhD is all about the journey and not about the destination. I was extremely fortunate to have a wonderful journey with amazing people who have been my collaborators, confidantes, friends, mentors, and family. I will try my best to thank each and every one of them but alas, my memory often fails me. So if I forget someone below, know that I am lucky to have known them and they made my journey all the more fun.

First and foremost, I would like to thank my advisors Max and Drew. I always wanted to work with both of them, and forging this joint collaboration remains as the most important decision I have made in my PhD. They were the yin and yang in my PhD, and the body of work you read in this thesis was a direct result of this delicate balance. Max always ensured that my ideas were grounded and would work on a robotic system, which was especially useful as I often flew away with idealistic ideas. I admire his extreme patience during our meetings and his approach of understanding complex ideas from fundamentals. If not these amazing skills, I hope I learned how to always stay humble and listen to collaborators from him. Drew has been a constant source of inspiration throughout my journey and allowed me to chase crazy ideas purely because they were interesting. For someone who is a CTO of a self-driving company, his passion for pure fundamental research is very inspiring and has driven me to work harder on ideas that I was excited about. I loved all our phone calls, with me speaking cryptic equations while he somehow miraculously deciphers them, our walks on campus discussing ideas and papers, and our terse emails with the entire content in the subject and no body. The most fun projects in my PhD were a result of brainstorming with Drew, and for that I am extremely grateful.

I take this opportunity to also thank my committee members, Oliver and Leslie. I interacted with Oliver a lot of times during my PhD both as an RI student who was interested in his work, and as his teaching assistant for two courses. I admire his straightforward way of thinking and insights in to the robot manipulation problem, which influenced my own research. Leslie has always been one of my idols and for someone entering into the field of robot planning and learning, some of her works were foundational. I am extremely lucky to have them both on my committee and thankful for all the feedback I have received from them on my thesis.

My journey also took me to some wonderful places besides CMU. I was fortunate to work with Martin Levihn, Chris Clark, Humphrey Hu, and Jakub Dworakowski at Apple Special Projects Group. All of them have been incredible mentors both professionally and personally. Their insights and guidance has influenced both this thesis and my future beyond PhD. I also worked with Vladlen Koltun and Ozan Sener at Intel Labs, where I learned how to tackle extremely difficult problems and the value of collaboration. I'd like to thank Ozan for his patience while I slowly grasped the challenges of the problem.

I would be remiss if I did not thank Jean Oh and Katharina Muelling who were my mentors at the start of this journey. They showed me how exciting research could be, and encouraged me to apply for a PhD. Jean is one of the nicest people I know, and I am fortunate to have been her student.

During my time at CMU, I was a part of two wonderful labs - SBPL and Lairlab. I would like to thank my officemates, Dhruv Saxena, Yash Oza, and Tushar Kusnur for putting up with my constant scribbles on the whiteboard and my loud discussions. I would like to thank Dhruv especially for always lending an ear when I needed it. For the first 3 years of my PhD, I was part of a truck unloading project and worked with fellow SBPL members - Oren Salzman, Fahad Islam, Sung-Kyun Kim, and Andrew Dornbush, who were all insanely helpful whenever I got stuck and helped me learn motion planning and belief space planning fundamentals. Besides work, I had loads of fun with other SBPL friends - Venkatraman Narayanan, Jacky Liang, Anahita Mohseni Kabir, Shivam Vats, Ram Natarajan, Manash

Pratim Das, Vinitha Ranganeni, Shohin Mukherjee, and Rishi Veerapaneni. While SBPL was where I physically most of my time, Lairlab was my go to for brainstorming on machine learning research. Wen Sun was my close collaborator initially in my PhD, and he almost became my third advisor for a year. Fellow "Bagnellians" - Allie Del Giorno, Hanzhang Hu, Jiaji Zhou, Wen Sun, Arun Venkatraman, and Gokul Swamy were wonderful labmates and I will always cherish the lab lunches with them which usually had heated discussions on topics ranging from research to the latest gossip. Special shout out to Allie for making my defense super fun, and helping me with reviewing all my papers and presentations.

The robotics institute at CMU has a wonderful student community. As a part of RoboOrg, and as a RI Masters and PhD student, I had the good fortune to know some of the most amazing and talented individuals. These friendships lasted over 6 years and will last for a lot more years to come. I would like to thank Puneet Puri, Vishal Dugar, Jerry Hsiung, Senthil Purushwalkam, Ankit Bhatia, Adithya Murali, Alex Spitzer, Xuning Yang, Achal Dave, Debidatta Dwibedi, Rogerio Bonatti, Zhi Tan, Rosario Scalise, Eric Huang, Arjav Desai, Tabitha Lee, Yifan Hou, Brian Okorn, Tanmay Shankar, Roberto Shu, Sibi Venkatesan, Nick Gisolfi, Abhijeet Tallavajhula, Ratnesh Madaan, Sankalp Arora, and Sanjiban Choudhury. A special shout-out to NSH 2204 for being legendary, you know you are. I would like to give special thanks to Sanjiban who was my first mentor at CMU, and remained a friend I always reach out to for guidance on work, life, and beyond.

Suzanne Muth and Jean Harpley made my life at RI feel seamless. Suzanne worked hard behind the scenes to ensure that RI PhDs could focus solely on research. Jean Harpley is one of the most amazing people in RI. She helped RoboOrg and the student community in RI at every step, even though it was not a part of her job description. For that and many more, I would like to thank both of them.

RI was too small to contain all the wonderful people I know, and I met people from other departments who played a big part in my journey. Abilash Subbaraman was always down to climb with me and do the craziest, and often the most dangerous, adventure sports with me. Ben Eysenbach helped me understand how my research fits with the broader RL literature. Shefali Umrania and Ananya Uppal were close friends who made my grad school life extremely fun. The Explorers club of Pittsburgh introduced me to several inspiring and fun people with whom I went backpacking, climbing, and biking with. All work and no play could have made this journey dull, but these folks ensured it was anything but dull.

I am the person I am today because of my family. My dad instilled the value of working hard in me, and pushed me to always have bigger ambitions. My brother is my constant partner-in-crime and was a huge inspiration in my career choices. He has always prioritized my choices over his own, and baked a fear of mediocrity in me, that has always pushed me to go further. My mom is the reason for all my successes, and her countless sacrifices gave me the opportunity to focus on my education while she took care of far more important issues. I hope I say it enough, but if I don't here it goes - I am extremely grateful to be your son, brother, and a part of this family.

I said PhD is all about the journey. This journey started with me meeting my partner Pragna Mannam, and she enriched it far beyond what it would have been without her. She always believed in me, when I did not. She brought me to the surface, when I sank to the depths. She supported me unconditionally, when I needed it. She is my rock, and the shining star in my sky. At the end of this journey, I am at a loss of words to thank her, and can only say "Cutie, we did it!"

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Remember that all models are wrong; the practical question is how wrong do they have to be to not be useful.

George Box (1987)

1.1 Motivation

Robotic planning algorithms have been widely successful in computing feasible and optimal plans, or sequence of decisions, for tasks involving robots operating in known environments or under known conditions [LaV06]. A large part of this success can be attributed to principled algorithms that effectively "search" the space of all plans by exploiting the known structure in the form of dynamical models to quickly compute the solution [Cho+05]. For example in the field of robot motion planning, there have been various developments in designing planning algorithms that exploit forward models to effectively discretize the state space into a graph and compute a feasible plan using graph search techniques [Lat91]. This enables planning algorithms to guarantee task completeness, which is a requirement on the solution plan to complete the task, and be efficient in the amount of computational resources needed to find the solution [Hau12].

However for robots to operate in unstructured environments such as homes, offices and disaster sites, planning algorithms have to reason about how to deal with the lack of complete knowledge of the environment while ensuring task completeness [KLC98]. To retain their effectiveness, these planning algorithms will have to utilize partial knowledge of the environment and the task in the form of simplified and *inaccurate* dynamical models [AQN06b]. Naively using these inaccurate dynamical models for planning can result in highly suboptimal plans and in some cases, plans that do not complete the task during execution [Kol10]. An example of such behavior is shown in Figure 1.1. In this example, a robotic arm is performing a pick-and-place task while avoiding collision with an obstacle. In the first scenario (the first three figures from the left in Figure 1.1,) the arm is interacting with a light object whose mass is accurately captured by the dynamical model used by the planner. This results in a computed trajectory for the arm that grasps the object, lifts it above the obstacle and takes it to the goal location. While this scenario has highlighted the effectiveness of the planning algorithm to complete the task when given access to an accurate dynamical model, consider the second scenario (the last figure on the right in

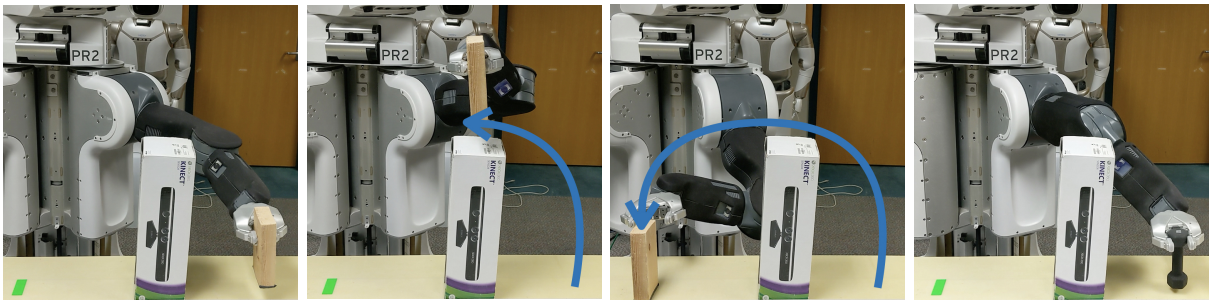

Figure 1.1: A robotic arm picking an object from its start location and placing it at a goal location while avoiding collision with the intermediate obstacle during motion. The first three (from left) figures show an execution with a light object (wooden block) and a plan (blue trajectory) computed using an accurate dynamical model which captures the weight of the object correctly. The last figure (rightmost) shows an instance of the same task but with a heavy object (black dumbbell) and same dynamical model as before which models the object as light. This results in the planner computing the same plan as before, which the robot cannot execute as lifting the heavy object requires joint torques that are beyond the robot's capabilities. Thus, the plan is not task complete.

Figure 1.1) where the arm is interacting with a heavy object which is modeled as a light object by the dynamical model. Since the model is same as before, the planner computes the same trajectory which lifts the object above the obstacle. However, while executing the trajectory the arm cannot lift the heavy object and cannot command the joint torques required because they are beyond the arm's capabilities. Thus, the computed plan is not successful in completing the task. The above example highlights the ineffectiveness of naively using these inaccurate models for planning. This ineffectiveness can be tackled broadly in two directions: updating either the dynamical model or the behavior of the planner, using the accumulated experience during execution.

1.1.1 Updating the Dynamical Model

The former direction of using online experience to update existing dynamical models or learning new dynamical models from scratch has been explored in the Reinforcement Learning (RL) framework. This framework enables autonomous agents, such as robots, to learn how to operate in an unknown environment by interacting with it and compute an optimal plan that minimizes total cost [SB98]. With partial or no prior knowledge about the environment, the agent needs to explore to discover low cost actions or regions where dynamics are inaccurately modeled. The exploration strategies leveraged by these agents require a large amount of interactions with the environment before we can compute plans that guarantee task completeness [Kak03]. This is a major reason why RL, despite being a very general framework, has mostly seen success in domains where we can afford to collect large amounts of interactions with little effort: video games and simulations [Sil+17; Ber+19; Hes+18].

Most methods in the RL framework can be categorized as either model-based or model-free (Figure 1.2). As the name suggests, model-based methods rely on using a model as input to a planning procedure to compute the solution plan for a given task. These methods use the experience gained online during execution to update the dynamics of the model and replan to

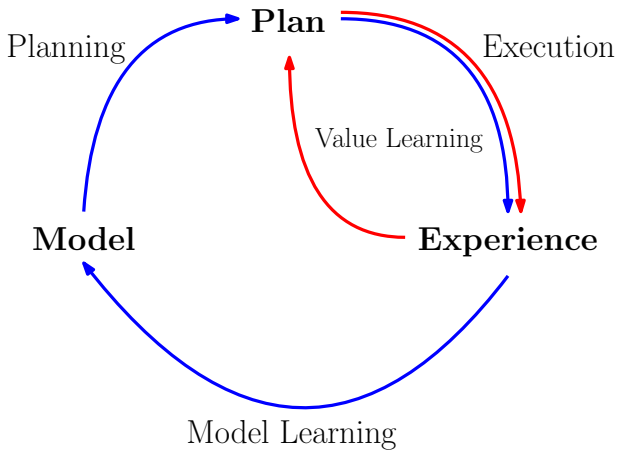

Figure 1.2: Operation of Model-based (blue) and Model-Free RL (red) methods while executing in unknown environments and collecting experience to complete a task. Figure inspired from Dyna [Sut91]

compute a new solution plan [Sut91]. In contrast, model-free methods directly use the accumulated experience to compute an updated solution plan without ever using a dynamical model. These methods utilize the experience to estimate value functions, which are essentially cost-to-go estimates, and compute a plan using the estimated values [WD92]. Both methods have advantages and disadvantages. Model-based methods relatively require fewer amounts of experience to compute a plan of the same quality as the plan computed by a model-free method [Sun+19a]. On the other hand, model-free methods are not affected by the biases inherent in the design of the model [DR10]. This thesis will primarily focus on model-based methods as they allow us to exploit existing domain knowledge in the form of inaccurate models. However, we also explore integrating model-free methods with model-based planning to combine the advantages of both approaches.

A primary disadvatange of methods that use the accumulated experience from execution to update the dynamics of the model is that in tasks with complex dynamics, learning the true dynamics can require a exhorbitantly large number of executions [Wan+19; Nag+18; Sch+19]. Furthermore, there might be no model in the model class considered that can accurately represent the true dynamics [Jos+13]. In such cases, prior works have shown that finding the model with the lowest prediction error need not guarantee task success when subsequently used for planning, and could have worse performance when compared to a model with higher prediction error [Far18; Gri+20]. Thus, it is non-trivial to guarantee task completeness for these approaches. Intuitively, there is an inherent mismatch of objectives where the model is updated to improve its prediction accuracy, while the planner uses the model to find a plan that completes the task [Lam+20]. This motivates the need to judiciously update the dynamical model with the goal of optimizing the planning process, and this thesis takes some preliminary steps in this direction by presenting a task-aware model learning approach that directly optimizes task success.

1.1.2 Updating the Behavior of the Planner

In contrast, the latter direction of updating the behavior of the planner using online experience has not been explored as extensively in past literature. Interestingly, this direction has been

in use by the practitioner for quite some time. As motivated earlier, in most robotic tasks we seldom have access to accurate dynamical models and the models we use for planning are often inaccurate. Robotics engineers and practitioners have been dealing with these inaccuracies by modifying how the planner uses the inaccurate model rather than updating the model to improve its accuracy. As an example, consider the task of planning footsteps of a mobile quadruped robot over partially unknown terrain as shown in Figure 1.3 taken from [Zuc+11]. The unknown part of the terrain is annotated in the figure (red oval.) To ensure that the planner does not compute footstep trajectory that goes through this region, a simple hack that the practitioner does is to inflate the cost of any state-action pair that takes the robot into this region. This results in biasing the planner away from this region thereby updating its behavior. There are several other works that deal with inaccurate modeling by simply updating the behavior of the planner [McC+20; MMB21; PB21; Lee+20a].

An observant reader will notice that these approaches require the models used for planning to not be completely inaccurate everywhere. As a simple example, if we use a model that predicts that the robot will crash for any action that it can possibly take, then simply updating the behavior of the planner will not result in any improvement as the model used is inherently bad. While these approaches have been explored in practice, there is very little prior work that has studied this direction from a theoretical point of view aiming to understand the assumptions required to guarantee task completeness, and a systematic study to analyze its empirical performance in practice. This thesis aims to fill this gap and develop a better understanding when, where, and how these methods work well in practice.

1.2 Thesis Goal and Contributions

While most existing works have presented and studied approaches that use the experience from executions to update the dynamics of the inaccurate model, one can argue that this is wasteful if we are interested in completing the task and not in modeling the dynamics accurately. Furthermore, robots operating in the real world have operating constraints that require them to quickly adapt to new scenarios and not spend hours acquiring experience to learn true dynamics. Finally in tasks where the dynamics are complicated, it could be computationally infeasible to have a representation in the model class that can capture the true dynamics. In light of these challenges and insights, the goal of this thesis is to justify the following statement:

> *By updating the behavior of the planner and not the dynamics of the model, we can leverage simplified and potentially inaccurate models and significantly reduce the amount of real-world experience needed to provably guarantee that the robot completes the task.*

We support this argument through the primary contributions of this thesis which are detailed in the following sections.

1.2.1 Sample Complexity of Exploration in Model-Free Policy Search

We analyze the sample complexity of exploration techniques in model-free RL methods. This analysis is presented by viewing model-free policy search methods through the lens of derivative-free optimization (DFO) and computing worst case upper bounds on the number of samples required to compute a ϵ-suboptimal policy. We present a DFO point of view for methods that involve either exploration in action space or exploration in policy space, and present trade-offs

Figure 1.3: A practitioner's approach to dealing with inaccuracies in dynamical models used for planning. In this example, the robot is planning a footstep trajectory along the partially unknown terrain to reach the other side. The planner has access to a model of the terrain which is inaccurate in the regions marked by red oval. To ensure that the planner does not compute any trajectory going through the red oval region, practitioners typically inflate the cost of any action executed within the region or any action that takes the robot into this region. This results in biasing the planner away from this region thereby updating its behavior. Figure taken from [Zuc+11] and the red oval region depicted is an example used for emphasis.

between both styles of exploration in terms of the dimensionality of the policy parameter space, and the horizon length of the task. This analysis is presented in Chapter 3 of the thesis and is also presented in our paper [VSB19]. In addition to contrasting exploration in policy space vs action space, this work also emphasizes the large sample complexity required by model-free methods, which cannot be realized in practice on a robot.

1.2.2 Planning and Execution using Inaccurate Models

We present the first systematic effort to understand methods that use online experience from executions to update the behavior of the planner and not update the dynamics of the model. These methods can make progress towards completing the task despite using a potentially inaccurate model. One can construct cases where if the model is highly inaccurate (e.g. a model that predicts a humanoid falling down for any action and failing to complete the task of moving forward,) then such a method cannot be expected to finish the task. Hence, we study the assumptions required on the accuracy of the model used for planning that ensures task completeness without requiring any updates to the dynamics of the model. Furthermore, we frame our problem in the purely online setting where the experience gathered by the robot is along a single trajectory without any access to resets. We believe that this setting is realistic and has challenges that these methods are uniquely positioned to tackle.

We propose Cmax, an approach that guarantees that the robot completes the task using the inaccurate model without any resets and without requiring any updates to the dynamics of the model. This is achieved by biasing the planner away from transitions whose dynamics are discovered to be inaccurately modeled during online execution. On the theoretical side, we establish provable guarantees on task completeness under assumptions on the accuracy of the model used for planning. Empirically, we show that Cmax outperforms state-of-the-art model-free and model-based RL methods in terms of the number of executions taken to complete the task. Crucially, Cmax exhibits goal-driven behavior which enables it to focus on completing the

task as quickly as possible and not waste executions learning the true dynamics. This method is explained in detail in Chapter 4 and is also presented in our paper [Vem+20].

1.2.3 Leveraging Experience in Planning and Execution using Inaccurate Models

While a robot using CMAX is provably guaranteed to complete the task, it requires strong assumptions on the accuracy of the model that are often not realized in practice and hard to verify prior to execution. Furthermore for repetitive tasks, CMAX fails to improve the quality of the solution over repetitions of the same task as it does not leverage previously discovered inaccurately modeled transitions. This is remedied by our second approach CMAX++ that leverages experience from past executions to improve the quality of solution over repetitions of the same task. Crucially unlike CMAX, CMAX++ can compute solution plans that contain previously discovered inaccurately modeled transitions. CMAX++ achieves this by integrating model-free value learning using acquired experience with model-based planning using the inaccurate model. As a consequence of this in addition to completeness, CMAX++ also guarantees asymptotic convergence to the optimal path cost as the number of repetitions increases. These guarantees of CMAX++ are established under assumptions on the accuracy of the model that are much more relaxed compared to the assumptions required by CMAX. Importantly, like CMAX, CMAX++ never updates the dynamics of the model. This method is explained in detail in Chapter 5 and is also presented in our paper [VBL20].

1.2.4 On the Effectiveness of Using Inaccurate Models

In Chapters 4 and 5, the focus is on proving task completeness guarantees for methods that update the behavior of the planner. However, there are no provable guarantees on the performance of the plan as a function of the modeling error. This is especially useful in understanding how accurate a model needs to be, in order to converge to a plan that has bounded worst case performance. To derive such a guarantee, we study the control of linearized systems with quadratic costs which is easier to analyze and provides insights into the performance we can expect from using an inaccurate model. In this setting, we analyze the performance of *iterative learning control* (ILC) approaches that use online experience from executions to update the behavior of the planner and do not update the model, similar to CMAX and CMAX++. We present upper bounds in terms of modeling error on the sub-optimality gap between the cost incurred by the controller that ILC converges to, and the cost incurred by using the optimal linear quadratic controller. Our analysis shows that the sub-optimality gap bound for ILC has a nice quadratic dependency on the modeling error. Furthermore, our analysis also highlights the pitfalls of methods that naively use inaccurate models without updating the behavior of the planner. For these methods, the sub-optimality gap bound has a dependence on quadratic and higher-order terms in modeling error that can make it significantly worse than ILC when modeling error is significant. This analysis is explained in detail in Chapter 6 and is also presented in our paper [Vem+21].

1.2.5 Task-Aware Online Model Search with Misspecified Model Classes

Our final contribution departs from the algorithms developed so far, and studies the alternative of using experience from executions to update the dynamics of the model. More precisely, we study the problem of planning in an environment with unknown transition dynamics when given access to a misspecified model class. A model class is misspecified if no model in the class can capture

the true dynamics of the environment, which is usually the case in real-world domains where we either have limited domain knowledge or we would like to use a small model for computational efficiency. In such a setting, finding a model that optimizes prediction error, as done in most of the existing work, can lead to poor task performance when the model is used for planning. We develop a model learning method TOMS that is task-aware, i.e. optimizes completing the task when used for planning, rather than prediction error. We achieve this by performing derivative-free optimization in the space of model parameters to explicitly select a model that results in a policy, upon planning, that achieves the best task performance during execution. To measure the performance of any policy in the environment without executing it, we rely on a monte-carlo evaluation procedure that uses the experience accumulated so far in state-action regions where we have good coverage, and fall back on an optimistic model everywhere else. Theoretically, we can show that given unlimited computation, TOMS is guaranteed to reach the goal in small state-action spaces as long as there is at least one good performing model in the model class. Empirically, we show that TOMS performs significantly better than traditional model learning methods that optimize prediction error in a simple mountain car domain. This work is explained in detail in Chapter 7.

1.3 Bibliographical Remarks

This thesis only contains works for which this author is the primary contributor.

Chapter 3 is based on joint work with Wen Sun and Drew Bagnell that appeared in [VSB19]. Chapter 4 is based on joint work with Yash Oza, Drew Bagnell, and Max Likhachev that appeared in [Vem+20]. Chapter 5 is based on joint work with Drew Bagnell and Max Likhachev that appeared in [VBL20]. Chapter 6 is based on joint work with Wen Sun, Max Likhachev, and Drew Bagnell that appeared in [Vem+21]. Chapter 7 is based on ongoing work with Sanjiban Choudhury, Drew Bagnell, and Max Likhachev.

1.4 Open Source Software

The author is a huge proponent of open sourcing research code. Previously open sourced code has hugely benefitted the work in this thesis, and the author would like to give back to the community by open sourcing all the code for this thesis. The links to code are listed below:

1. Chapter 3 code can be found at `https://github.com/LAIRLAB/contrasting_exploration_rl`
2. Chapter 4 code can be found at `https://github.com/vvanirudh/CMAX`
3. Chapter 5 code can be found at `https://github.com/vvanirudh/CMAXPP`
4. Chapter 6 code can be found at `https://github.com/vvanirudh/ILC.jl`
5. Chapter 7 code can be found at `https://github.com/vvanirudh/TOMS.jl`

1.5 Excluded Research

The author has excluded a significant portion of his doctoral work for the purpose of keeping this thesis succinct. The excluded works are listed below:

1. TRON: A Fast Solver for Trajectory Optimization with Non-Smooth Cost Functions, that appeared in [VB20].
2. Planning, Learning and Reasoning Framework for Robot Truck Unloading, that appeared in [Isl+20].
3. Provably Efficient Imitation Learning from Observation Alone, that appeared in [Sun+19b].
4. Improved Soft Duplicate Detection in Search-Based Motion Planning, that appeared in [MVL21].
5. Task-informed Fidelity Management for Speeding up Robotics Simulation, that appeared in [Tal+19].

Chapter 2

Background

If I have seen further it is by standing on the shoulders of Giants.

Isaac Newton (1676)

In this chapter, we provide background knowledge with the aim of introducing classical techniques that we will use throughout this thesis. Each chapter of this thesis is self-contained and has detailed definitions that are more tailored to the specific problem tackled in each chapter.

2.1 Fundamentals of Markov Decision Processes

In this thesis, we will primarily deal with finite horizon problems where the objective is for an agent to minimize cumulative cost incurred over a horizon of finite length. This is typically formulated as a finite horizon Markov Decision Process (MDP) [BEL57] that is represented as $(\mathbb{S}, \mathbb{A}, P, c, H)$ where $\mathbb{S}$ is the set of states that the agent can be in, $\mathbb{A}$ is the set of actions that the agent can execute, P is the transition dynamics such that for any $s_t \in \mathbb{S}$, $s_{t+1} \in \mathbb{S}$, $a_t \in \mathbb{A}$, $P(s_{t+1}|s_t, a_t)$ is the probability of transitioning to state s_{t+1} from state s_t by taking action a_t, c is the cost function such that for any transition (s_t, a_t), $c(s_t, a_t)$ is the cost incurred for that transition, and $H \in \mathbb{N}^+$ is the length of the horizon. Typically, this formulation also has a discounting factor γ and a initial state distribution ρ. In this thesis, we consider the non-discounted setting where $\gamma = 1$ and a fixed initial state s_0 that is known (thus, ρ is a delta distribution on s_0.)

A deterministic policy $\pi : \mathbb{S} \to \mathbb{A}$ maps from a state to an action. Given π and a time step t, we can define the cost-to-go or value estimate $V_\pi^t(s)$ as follows:

$$V_\pi^t(s) = \mathbb{E}\left[\sum_{i=t}^{H} c(s_i, a_i) | a_i = \pi(s_i), s_{i+1} \sim P_{s_i,a_i}, s_t = s\right] \tag{2.1}$$

where $P_{s_i,a_i} = P(\cdot|s_i, a_i)$ is a distribution over the next state.

Similarly, we can also define the state-action value estimate $Q_\pi^t(s, a)$ as follows:

$$Q_\pi^t(s, a) = c(s, a) + \mathbb{E}_{s' \sim P_{s,a}}[V_\pi^{t+1}(s')] \tag{2.2}$$

The objective function $J(\pi)$ is defined as

$$J(\pi) = V_\pi^0(s_0) \tag{2.3}$$

and the goal is to find a policy from a given policy set Π that minimizes the above objective function.

If the cost function and the transition dynamics is known, then one can compute the optimal policy using Dynamic Programming (DP.) Denote the optimal policy using π^*. Define $Q^{H-1}_{\pi^*}(s,a) = c(s,a)$, we perform DP as follows: Starting from $t = H-2$ until $t = 1$ we iteratively do

$$V^t_{\pi^*}(s) = \max_{a \in \mathbb{A}} Q^t_{\pi^*}(s,a) \tag{2.4}$$

$$Q^{t-1}_{\pi^*}(s,a) = c(s,a) + \mathbb{E}_{s' \sim P_{s,a}}[V^t_{\pi^*}(s')] \tag{2.5}$$

Given $Q^t_{\pi^*}$ we can compute the optimal action at time step t and state s_t as $\min_{a \in \mathbb{A}} Q^t_{\pi^*}(s_t, a)$. The above iterative process is called as Value Iteration. This iterative procedures is derived by observing that the value function of the optimal policy satisfies the following fixed point equation

$$V^t_{\pi^*}(s) = \min_{a \in \mathbb{A}} \left(c(s,a) + \mathbb{E}_{s' \sim P_{s,a}}[V^{t+1}_{\pi^*}(s')] \right) \tag{2.6}$$

The above equation is known as the Bellman optimality condition.

2.2 Deterministic Shortest Path Problem

The shortest path problem is to find among all paths that start at a given state and end at a goal state, a path has the minimum cost; this is also called a shortest path [BT95]. This can be instantiated as a markov decision process represented using the tuple $(\mathbb{S}, \mathbb{A}, \mathbb{G}, f, c)$ where $\mathbb{G} \subseteq \mathbb{S}$ is the set of goal states, and $f : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ is a deterministic dynamics function that determines the successor state s_{t+1} of a transition (s_t, a_t) as $f(s_t, a_t)$. The goal states in $\mathbb{G}$ are cost-free termination states, i.e. $f(g,a) = g$, $c(g,a) = 0$ for all $g \in \mathbb{G}$ and any action $a \in \mathbb{A}$. We also assume that the cost of any transition starting from a non-goal state is positive, i.e. $c(s,a) > 0$ for all $s \in \mathbb{S} \setminus \mathbb{G}$ and $a \in \mathbb{A}$.

We are interested in problems where reaching the termination state is inevitable, at least under an optimal policy. Thus, the essence of the problem is how to reach a goal state with minimum cost. In the shortest path problem setting, we use $V(s)$ to denote the cost-to-go (or value) estimate of any state $s \in \mathbb{S}$ and $V^*(s)$ to denote the optimal cost-to-go (or value.) From the Bellman optimality condition we know

$$V^*(s) = \min_{a \in \mathbb{A}} \left(c(s,a) + V^*(f(s,a)) \right) \tag{2.7}$$

A value estimate V is called admissible if underestimates the optimal value at all states, i.e. $V(s) \leq V^*(s)$ for all $s \in \mathbb{S}$. Furthermore, V is called consistent if it satisfies the condition that for any state-action pair (s,a), $s \notin \mathbb{G}$, $V(s) \leq c(s,a) + V(f(s,a))$ and $V(g) = 0$ for all $g \in \mathbb{G}$. A typical assumption made in all shortest path problems is that there exists at least one path from each state $s \in \mathbb{S}$ to one of the goal states in $\mathbb{G}$. This ensures that the optimal value for any state is finite.

2.3 Real-time Heuristic Search

A traditional way to solve the shortest path problem is to search the graph constructed using a mental model of the world, and then subsequently execute the resulting plan (or follow the

Algorithm 1 LRTA* with Lookahead 1 [Kor90]

1: $s \leftarrow s_0$
2: **while** $s \notin \mathbb{G}$ **do**
3: Compute action $a = \text{argmin}_{a \in \mathbb{A}} \left(c(s,a) + V(f(s,a))\right)$
4: Update $V(s) \leftarrow \min\left(V(s), c(s,a) + V(f(s,a))\right)$
5: Execute action a and update $s \leftarrow f(s,a)$

computed path.) Thus, planning and execution are completely separated. An alternative way of solving this problem is to search online by interleaving planning and execution which results in several advantages with the major advantage being drastic reductions in planning time. This is achieved by performing search locally until a fixed horizon (or until a fixed number of states are expanded,) and then execute the best action for the current state. After the execution, planning is performed once again to find the next best action. This can decrease the time used for planning as we are not planning all the way to the goal. Another significant advantage is when the mental model of the world is inaccurate, these methods enable the agent to update its model and ensure future replanning results in more optimal paths.

Since the future consequences of executed actions are unknown, interleaving planning and execution can result in slight overhead in terms of the number of actions executed but this is often a much smaller overhead compared to the reduced planning time. Real-time search methods are methods that interleave planning and execution by searching forward from the current state of the agent. Most importantly, real-time heuristic search methods can satisfy hard real-time requirements in large state spaces since the sizes of their local search spaces are independent of the sizes of the state spaces and can thus remain small.

2.3.1 LRTA*

In this thesis, we will focus on Learning real-time A* (LRTA*) real-time search methods that are real-time search methods that associate information with the states to prevent cycling. These methods are promising for interleaving planning and execution as they are efficient domain-independent search methods that allow fine-grained control over how much planning is allowed between executions, use heuristic knowledge to guide planning, and improve their performance over time as they solve similar planning tasks. LRTA* operates on deterministic domains only.

Algorithm 1 presents the LRTA* algorithm for a lookahead or search horizon of 1. At each time step, the algorithm looks one action execution ahead and always greedily chooses the action that leads to a successor state with the minimum sum of cost of transitioning into the successor state and the value estimate of the successor state. Furthermore unlike classical real-time search methods, LRTA* also updates the value estimates of the current state to reflect the updated estimate of the best path to the goal so that future replanning is more efficient. The planning time of LRTA* between executions is linear in the number of actions. If the size of action space is independent of the size of state space, then the planning time is independent of the size of state space which is a major improvement over offline planning methods whose computational complexity is at most the size of the state space.

LRTA* can be viewed as a form of asynchronous incremental dynamic programming method [BBS95]. It can be shown that LRTA* is guaranteed to reach a goal state in a finite number of executions and if we reset to the start state after reaching the goal state, then the value estimates eventually converge to the optimal value function [Kor90]. These guarantees hold under the

Algorithm 2 RTAA* with lookahead $K \geq 1$ [KL06]

```
s ← s_0
while s ∉ 𝔾 do
    Construct a search tree at s until K expansions
    Estimate s̄ as the leaf node with the least g + V estimate among all leaf nodes
    for all expanded states s' do
        Update V(s') ← g(s̄) + V(s̄) − g(s')
    Compute action a as the first action on the path from state s to state s̄ in the search tree
    Execute action a and update s ← f(s, a)
```

assumption that the initial value estimates that we start with are admissible and consistent. These assumptions are very similar to the traditional definitions of admissible and consistent heuristic values for A* search. Note that zero-initialized value estimates are both admissible and consistent.

2.3.2 RTAA*

Real-time Adaptive A* (RTAA*) proposed in [KL06] is similar to LRTA* (Algorithm 2). They only differ in the way they update the value estimates at each time step. To understand this better, let us look at how LRTA* updates value estimates. LRTA* replaces the value estimate of each expanded state with the sum of costs of from the state to a generated but unexpanded state s (leaf node in the search tree) and the value estimate of state s, minimized over all generated but unexpanded states (all leaf nodes of the search tree.) If we denote V as the value estimates after all the value updates then the LRTA* updates satisfy the following system of equations for all expanded states s:

$$V(s) = \min_{a \in \mathbb{A}} \left(c(s, a) + V(f(s, a)) \right) \tag{2.8}$$

On the other hand, RTAA* constructs a search tree very similar to LRTA* (the number of states expanded is equal to the lookahead) but updates the value estimates for all expanded states s as follows:

$$V(s) = g(\bar{s}) + V(\bar{s}) - g(s) \tag{2.9}$$

where $g(s)$ encodes the cost-to-come from the root of the search tree to the state s (i.e. sum of costs of all transitions on the path from root to state s,) and $\bar{s}$ is the state corresponding to the leaf node with the least sum of g and V among all leaf nodes in the search tree. In other words, $\bar{s}$ is the state that was about to be expanded just before the search was terminated. One can show that LRTA* and RTAA* updates are exactly the same when the lookahead is 1. But when the lookahead is greater than 1, these updates differ. More specifically, LRTA* updates tend to be more informed or reflect the optimal cost-to-go better when compared to RTAA* updates. However, it takes LRTA* more time to update the value estimates and is difficult to implement. This is because LRTA* performs one search to determine the local search space and a second search to determine how to update the value estimates since it is unable to use the results from the first search for this purpose. Thus, there is a trade-off between the total search time and cost of the resulting path. In practice, for lookaheads greater than 1, RTAA* tends to compute solution paths that have higher costs compared to LRTA* but the time taken for planning before each execution is significantly less in RTAA* compared to LRTA*. This makes RTAA* desirable

in applications where planning is slow but actions can be executed fast and there is a very strict time limit per search episode.

2.4 Local Function Approximation Methods

The goal of function approximation methods is to capture the underlying relationship between input and output data. A typical approach is to use all the training data to fit a global model that predicts the output given the input throughout the input space. The hope is that this approximation predicts output values that are close to the true output values of the original function. A major disadvantage of these global function approximation methods is that in many cases, there exists no parameter values that provide a sufficiently good approximation. Moving to a larger function approximation class with more parameters requires a significantly larger training data which might not be available. Furthermore, in cases where the model needs to be updated incrementally, the computational cost of recomputing the global function approximation is very high and potentially infeasible on real-time systems.

An alternative to global function approximation methods are local function approximation methods such as Locally Weighted Learning (LWL) [AMS97a; AMS97b]. LWL methods are non-parametric and prediction is computed using local functions which use only a subset of the training data. The basic idea of LWL is for each query point, a local model is constructed based on neighboring training data. Each data point is associated with a weighting factor that captures the influence of the data point in computing the prediction for the query point. Intuitively, the closer the data point to the query point the higher its influence. Since the training data is directly used during prediction and there is no pre-processing before prediction, LWL can be a very accurate and fast incremental function approximation method.

For ease of exposition, let us consider the following regression model

$$y = f(\mathbf{x}) + \epsilon \tag{2.10}$$

where $f(\mathbf{x})$ is the unknown function that we are seeking to approximate, $\mathbf{x} \in \mathbb{R}^d$, $y \in \mathbb{R}$ and ϵ is zero mean noise. Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ and a query point $\mathbf{x}_q$ we can define the following cost function,

$$J(\beta_q) = \frac{1}{N} \sum_{i=1}^{N} w_i(\mathbf{x}_q)(y_i - \beta_q^T \mathbf{x}_i)^2 \tag{2.11}$$

where w_i are weights that capture the influence of the i-th data point $(\mathbf{x}_i, y_i)$ on the prediction for query point $\mathbf{x}_q$, and β_q is the coefficients for our linear model that is used for prediction. The goal is to find β_q that minimizes the above cost function and predict $\hat{y}_q = \beta_q^T \mathbf{x}_q$ (Assume that $\mathbf{x}_i, \mathbf{x}_q$ vectors have a 1 added to account for the offset term.) The weights $w_i(\mathbf{x}_q)$ are computed typically using a distance metric $d(\mathbf{x}_i, \mathbf{x}_q)$ that captures relevance of training points to the query point, and a kernel function $K(d)$ which computes the weight given a distance value.

2.4.1 K-Nearest Neighbor Regression

A very simple LWL method is K-Nearest Neighbor (KNN) regression which given a query point $\mathbf{x}_q$ finds the K nearest neighbors in the training data $\mathcal{D}$ using a distance metric $d(\mathbf{x}_q, \mathbf{x}_i)$. There are several variants of this method, one of which uniformly weights all the K nearest neighbor's

Algorithm 3 Locally Weighted Regression

1: **Input:** Training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, Query point $\mathbf{x}_q$
2: Construct matrix X with rows corresponding to $\hat{\mathbf{x}}_i$ where $\hat{\mathbf{x}}_i = [\mathbf{x}_i^T 1]^T$
3: Construct vector $\mathbf{y}$ with each element corresponding to y_i
4: Compute diagonal weight matrix W where the i-th diagonal element is given by $\exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_q)^T D(\mathbf{x}_i - \mathbf{x}_q)\right)$
5: Compute $\beta_q = (X^T W X)^{-1} X^T W \mathbf{y}$
6: Compute prediction $\hat{y}_q = \beta_q^T \hat{\mathbf{x}_\mathbf{q}}$ where $\hat{\mathbf{x}_\mathbf{q}} = [\mathbf{x}_q^T 1]^T$

outputs to obtain the prediction for the query point, i.e.

$$\hat{y}_q = \frac{1}{K} \sum_{\mathbf{x}_i \in \mathcal{D}_K(\mathbf{x}_q)} y_i \tag{2.12}$$

where $\mathcal{D}_K(\mathbf{x}_q)$ represents the set of size K consisting of the K nearest neighbors of the query point $\mathbf{x}_q$ in $\mathcal{D}$. Another variant, which often works well in practice, is to weigh each neighbor by the inverse of their distance to the query point. Intuitively, closer neighbors have a greater influence than farther neighbors. Thus, we have

$$\hat{y}_q = \frac{\sum_{\mathbf{x}_i \in \mathcal{D}_K(\mathbf{x}_q)} w_i y_i}{\sum_{\mathbf{x}_i \in \mathcal{D}_K(\mathbf{x}_q)} w_i} \tag{2.13}$$

where the weight $w_i = \frac{1}{d(\mathbf{x}_q, \mathbf{x}_i)}$ is the inverse of distance to the query point. KNN regression requires storing all the training data in memory in the form of K-d trees which allow very fast computation of K nearest neighbors.

2.4.2 Locally Weighted Regression

Locally weighted regression (LWR) is a locally weighted learning method that maintains all the training data in memory and quickly computes the prediction for any given query point. LWR is presented in Algorithm 3 which is executed once for each query point x_q. There are also simple extensions to the batch setting where we want to obtain predictions for a batch of query points. The only hyperparameter is the matrix D which is usually set to a scaled identity matrix $D = h\mathbb{I}$ where h is a scalar hyperparameter that is chosen using cross validation. LWR typically has a very high approximation accuracy due to its local nature and has only few hyperparameters. The disadvantage is that Algorithm 3 has a computational complexity of $\mathcal{O}(N^2)$ where N is the size of training data, which can be very expensive for large datasets. Furthermore, LWR requires you to store all the training data in memory which might be infeasible for extremely large datasets.

Chapter 3

Sample Complexity of Exploration in Model-Free Policy Search

. . . much of the benefit of policy search is achieved by black-box methods.

Jens Kober, Drew Bagnell and Jan Peters (2013)

In this chapter, we will analyze the sample complexity of exploration techniques that are popularly used in model-free reinforcement learning methods. More specifically, we will contrast the number of samples required to obtain an ϵ-suboptimal policy between methods that explore in policy space versus those that explore in action space. Our goal in this chapter is to understand the inherent trade-off between these two exploration techniques and to get a better grasp of when one works better than the other, in terms of sample complexity. This chapter also motivates the general theme of this thesis that dynamical models, despite being potentially inaccurate, are extremely useful sources of domain knowledge, and the price to pay for using model-free approaches can be steep. This chapter is adapted from our original paper [VSB19].

3.1 Introduction

Model-free policy search is a general approach to learn parameterized policies from sampled trajectories in the environment without learning a model of the underlying dynamics. These methods update the parameters such that trajectories with higher returns (or total reward) are more likely to be obtained when following the updated policy [KBP13]. The simplicity of these approaches have made them popular in Reinforcement Learning (RL).

Policy gradient methods, such as REINFORCE [Wil92] and its extensions [Kak02; Bag+04; Sil+14; Sch+15], compute an estimate of a direction of improvement from sampled trajectories collected by executing a stochastic policy. In other words, these methods rely on randomized exploration in action space. These methods then leverage the Jacobian of the policy to update its parameters to increase the probability of good action sequences accordingly. Such a gradient estimation algorithm can be considered a combination of a *zeroth-order* approach and a *first-order* approach: (1) it never exploits the slope of the reward function or dynamics, with respect

to actions, but rather relies only on random exploration in action space to discover potentially good sequences of actions; (2) however, it *exploits* the first order information of the parameterized policy for updating the policy's parameters. Note that the chance of finding a sequence of actions resulting in high total reward decreases (as much as exponentially [KL02]) as the horizon length increases and thus policy gradient methods often exhibit high variance and a resulting large sample complexity [PS08; Zha+11].

Black-box policy search methods, on the other hand, seek to directly optimize the total reward in the space of parameters by employing , e.g., finite-difference-like methods to compute estimates of the gradient with respect to policy parameters [BS01; MRG03; HI08; TSC11; Seh+10; Sal+17; MGR18]. Intuitively, these methods rely on exploration in parameter space: by searching in the parameter space, these methods may discover an improvement direction. Note that these methods are fully zeroth-order, *i.e.*, they exploit no first-order information of the parameterized policy, the reward, or the dynamics. Although policy gradient methods leverage *more* information, notably the Jacobian of the action with respect to policy, black-box policy search methods have at times demonstrated better empirical performance (see the discussion in [KBP13; MGR18]). These perhaps surprising results motivate us to analyze: *In what situations should we expect parameter space policy search methods to outperform action space methods?*

To do so, we leverage prior work in zeroth-order optimization methods. In the convex setting, [FKM05; ADX10; NS17] showed that one can construct gradient estimates using zeroth order oracles and derived upper bounds on the number of samples needed. But for most RL tasks, the return as a function of parameters, or action sequence, is highly non-convex [SB98]. Hence we focus on the non-convex setting and analyze convergence to stationary points. [GL13; NS17] studied zeroth order non-convex optimization by providing upper bounds on the number of samples needed to close in on a stationary point. Computing lower bounds in zeroth order non-convex optimization is still an open problem [Car+17a; Car+17b].

In our work, we extend the analysis proposed in [GL13] to the policy search setting and analyze the sample complexity of parameter and action space exploration methods in policy search. We begin with a degenerate, one-step control problem of online linear regression with partial feedback, [FKM05], where the objective is to learn the parameters of the linear regressor without access to the true scalar regression targets. We show that for parameter space exploration methods, to achieve ϵ-optimality, requires $O(b^2/\epsilon^4)$ samples, where b is the input feature dimensionality. By contrast, an action space exploration method requires $O(1/\epsilon^4)$ many samples with a sample complexity *independent* of input feature dimensionality b. This is tested empirically on two simple tasks: Bandit Multi-class learning on MNIST with policies parameterized by convolutional neural networks which can be seen as a Contextual Bandit problem with rich observations, and Online Linear Regression with partial information. The results demonstrate action space exploration methods outperform parameter space methods when the parameter dimensionality is substantially larger than action dimensionality.

We present similar analysis for the multi-step control problem of model-free policy search in reinforcement learning, [KBP13], by considering the objective of reaching ϵ-close to a stationary point in the sense that $\|\nabla J(\theta)\|_2^2 \leq \epsilon$ for the non-convex objective $J(\theta)$. Our results show that, under certain assumptions, parameter space exploration methods need $\mathcal{O}(\frac{d^2}{\epsilon^3})$ samples to reach ϵ close to a stationary point, where d is the policy parameter dimensionality. On the other hand, action space exploration methods need $\mathcal{O}(\frac{p^2H^4}{\epsilon^4})$ samples to achieve the same objective, where p is the action dimensionality and H is the horizon length of the task. This shows that action space exploration methods have a dependence on the horizon length H while parameter space

exploration methods depend only on parameter space dimensionality d. Ongoing work by [TR18] demonstrated through asymptotic lower bounds that the dependence of sample complexity of action space exploration methods on horizon H is unavoidable in the LQR setting. This is tested empirically on popular RL benchmarks from OpenAI gym [Bro+16a], and the results show that as horizon length increases, parameter space methods outperform action space exploration methods. This matches the intuition and results presented in recent works like [BS01; SL06; TSC11; Sal+17; MGR18] that show parameter space black-box policy search methods outperforming state-of-the-art action space methods for tasks with long horizon lengths.

In summary, our analysis and experimental results suggests that the complexity of exploration in action space depends on both the dimensionality of action space and *horizon*, while the complexity of exploration in parameter space solely depends on dimensionality of parameter space, providing a natural way to trade-off between these approaches.

3.2 Problem Setup

3.2.1 Multi-step Control: Reinforcement Learning

We consider the problem setting of model-free policy search with the goal of minimizing sum of costs (or maximizing sum of rewards) over a fixed, finite horizon H. In reinforcement learning (RL), this is typically formulated using Markov Decision Processes (MDP) [SB98]. Denote the state space of the MDP as $\mathcal{S} \subset \mathbb{R}^b$, action space as $\mathcal{A} \subset \mathbb{R}^p$, transition probabilities as $\mathbb{P}_{sa} = \mathbb{P}(\cdot|s,a)$ (which is the distribution of next state after executing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$), an initial state distribution μ, and a cost function $c(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Note that the cost can be interpreted as negative of the reward. In addition to this, we assume a restricted class of deterministic, stationary policies Π parameterized by $\theta \in \mathbb{R}^d$ where each $\pi(\theta, \cdot) \in \Pi$ is differentiable at all θ and is a mapping from $\mathcal{S}$ to $\mathcal{A}$, i.e. $\pi(\theta, \cdot) : \mathcal{S} \to \mathcal{A}$. The distribution of states at timestep t induced by running the policy $\pi(\theta, \cdot)$ until and including t, is defined $\forall s_t : d^t_{\pi_\theta}(s_t) = \sum_{\{s_i\}_{i \le t-1}} \mu(s_0) \prod_{i=0}^{t-1} \mathbb{P}(s_{i+1}|s_i, a_i = \pi(\theta, s_i))$, where by definition $d^0_{\pi_\theta}(s) = \mu(s)$ for any π. We define the value function $V^t_{\pi_\theta}(s)$ for $t \le H-1$ as

$$V^t_{\pi_\theta}(s) = \mathbb{E}[\sum_{i=t}^{H} c(s_i, \pi(\theta, s_i))|s_t = s]$$

and state-action value function $Q^t_{\pi_\theta}(s,a)$ as

$$Q^t_{\pi_\theta}(s,a) = c(s,a) + \mathbb{E}_{s' \sim \mathbb{P}_{sa}}[V^{t+1}_{\pi_\theta}(s')]$$

Throughout this work, we assume the total cost is upper bounded by a constant, i.e., $\sup_{c_1,\ldots,c_T} \sum_t c_t \le \mathcal{Q} \in \mathbb{R}^+$, to prevent confounding due to just a change in the scale of total costs. We have then that $Q^t_{\pi_\theta}$ is upper bounded by a constant $\mathcal{Q}$ for all t and θ.

We seek to minimize the performance objective given by $J(\theta) = \mathbb{E}_{s \sim \mu}[V^0_{\pi_\theta}(s)]$. Given this objective, the optimization problem can be formulated as:

$$\min_\theta J(\theta) \tag{3.1}$$

The goal is to find parameters θ^* that minimize the expected sum of costs $J(\theta)$, given no access to the underlying dynamics of the environment other than samples from the distribution $\mathbb{P}_{sa}$ by

executing the policy $\pi(\theta, \cdot)$. However, the objective $J(\theta)$ can be highly non-convex and finding a global minima could be intractable. Thus, in this work, we hope to find a stationary point θ^* of the objective $J(\theta)$, *i.e.* a point where $\nabla_\theta J(\theta) \approx 0$.

3.2.2 One-Step Control: Online Linear Regression with Partial Information

The online linear regression problem is defined as follows: We denote $\mathcal{S} \subset \mathbb{R}^b$ as the feature space, and $\Theta \subset \mathbb{R}^d = \mathbb{R}^b$ as the linear policy parameter space where each $\theta \in \Theta$ represents a policy $\pi(\theta, s) = \theta^\top s$. Online linear regression operates in an adversarial online learning fashion: every round i, nature presents a feature vector $s_i \in \mathcal{S}$, the learner makes a decision by choosing a policy $\theta_i \in \Theta$ and predicts the scalar action $\hat{a}_i = \theta_i^\top s_i$; nature then reveals the loss $(\hat{a}_i - a_i)^2 \in \mathbb{R}^+$, which is just a scalar, to the learner, where a_i is ground truth selected by nature and is never revealed to the learner. We do not place any statistical assumption on the nature's process of generating feature vector s_i and ground truth a_i, which could be completely adversarial. Other than the adversarial aspect of the problem, note that the above setup is a special setting of RL with horizon $H = 1$, linear policy $\theta^\top s_i$, one-dimension action space, and a cost function $c_i(\theta) = (\theta^\top s_i - a_i)^2$. In this setting, we consider the *regret* with respect to the optimal solution in hindsight,

$$\text{Regret} = \sum_{i=1}^{T} c_i(\theta_i) - \min_{\theta^\star \in \Theta} \sum_{i=1}^{T} c_i(\theta^\star) \tag{3.2}$$

3.3 Online Linear Regression with Partial Information

3.3.1 Exploration in Parameter Space

We can apply a zeroth-order online gradient descent algorithm for the sequence of loss functions $\{c_i\}_{i=1}^T$, which is summarized in Algorithm 4. The main idea is to add random noise u, sampled from a unit sphere in b-dim space $\mathbb{S}_b$, to the parameter θ, and querying loss at $\theta + \delta u$ for some $\delta > 0$. Using the received loss $c_i(\theta + \delta u)$, one can form an estimation of $\nabla_\theta c_i(\theta)$ as $\frac{c_i b}{\delta} u$ [FKM05].

Algorithm 4 Random Search in Parameter Space (BGD [FKM05])

1: **Input:** $\alpha \in \mathbb{R}^+$, $\delta \in \mathbb{R}^+$.
2: Learner initializes $\theta_1 \in \Theta$.
3: **for** $i = 1$ to T **do**
4: Learner samples $u \sim \mathbb{S}_b$.
5: Learner chooses predictor $\theta'_i = \theta_i + \delta u$.
6: Learner only receives loss signal $c_i(\theta'_i)$.
7: Learner update: $\theta'_{i+1} = \theta_i - \alpha \frac{c_i b}{\delta} u$.
8: Projection $\theta_{i+1} = \arg\min_{\theta \in \Theta} \|\theta'_{i+1} - \theta\|_2^2$.

3.3.2 Exploration in Action Space

The *key difference* between exploration in action space and exploration in parameter space is that we are going to leverage our knowledge of the policy $\pi(\theta, s) = \theta^\top s$. Since we design the policy class, we can compute its *Jacobian* with respect to its parameters θ without interaction with the environment. The Jacobian of the policy gives us a locally linear relationship between

a small change in parameter and the resulting change in policy's action space. The main idea then in this approach is to explore with randomization in action space, and then leverage the Jacobian of the policy to update the parameters θ accordingly so that the policy's output moves towards better actions. Intuitively, we expect that random exploration in action space will result in smaller regret, as in our setting the action space is just 1-dimensional, while the parameter space is b-dimensional. The approach is summarized in Algorithm 5. Denote $\ell_i = (\hat{a}_i - a_i)^2$ and $\hat{a}_i = \pi(\theta_i, s_i) = \theta_i^\top s_i$. The main idea is that we can compute $\nabla_\theta c_i(\theta_i)$ via a chain rule as $\nabla_\theta c_i(\theta_i) = \frac{\partial \ell_i}{\partial \hat{a}_i} \nabla_\theta \pi(\theta_i, s_i)$. Note that $\nabla_\theta \pi(s_i, \theta_i) = \nabla_\theta \theta_i^\top s_i = s_i$ is the Jacobian of the policy to which we have full access. We then use zeroth order approximation method to approximate $\partial \ell_i / \partial \hat{a}_i$ at $\hat{a}_i = \pi(\theta_i, s_i)$.

Algorithm 5 Random Search in Action Space

1: **Input:** $\alpha \in \mathbb{R}^+$, $\delta \in \mathbb{R}^+$.
2: Learner initializes $\theta_1 \in \Theta$.
3: **for** $i = 1$ to T **do**
4: Learner receives feature s_i.
5: Learner samples e uniformly from $\{-1, 1\}$.
6: Learner makes a prediction $\hat{a}_i = \theta_i^\top s_i + \delta e$
7: Learner only receives loss signal $c_i = (\hat{a}_i - a_i)^2$.
8: Learner update: $\theta'_{i+1} = \theta_i - \alpha \frac{c_i e}{\delta} s_i$.
9: Projection $\theta_{i+1} = \arg\min_{\theta \in \Theta} \|\theta'_{i+1} - \theta\|_2^2$.

3.3.3 Analysis

We analyze the regret of the exploration in parameter space algorithm (Alg. 4) and the exploration in action space algorithm (Alg. 5) in this section. For analysis, we assume that Θ is bounded, i.e., $\sup_{\theta \in \Theta} \|\theta\|_2 \leq C_\theta \in \mathbb{R}^+$, $\mathcal{S}$ is bounded, i.e., $\sup_{s \in \mathcal{S}} \|s\|_2 \leq C_s \in \mathbb{R}^+$, and the ground truth a_i is bounded, i.e., $|a_i| \leq C_a$ for any i. Under the above assumptions, we can make sure that the loss is bounded as well, $(\theta^\top s - a)^2 \leq C \in \mathbb{R}^+$. The loss function is also Lipschitz continuous with Lipschitz constant $L \leq (C_\theta C_s + C_a) C_s$. We call these constants C_s, C_θ, and C_a as problem dependent constants, which are independent of feature dimension b and number of rounds T. In regret bounds, we absorb problem dependent constants into $\mathcal{O}$ notations, but the bounds will be explicit in b and T. The theorem below presents the average regret analysis for these methods,

Theorem 3.3.1. *After T rounds, with $\alpha = \frac{C_\theta \delta}{b(C^2 + C_s^2)\sqrt{T}}$ and $\delta = T^{-0.25} \sqrt{\frac{C_\theta b (C^2 + C_s^2)}{2L}}$, Alg. 4 incurs average regret:*

$$\frac{1}{T}(\mathbb{E}[\sum_{i=1}^{T} c_i(\theta_i)] - \min_{\theta^\star \in \Theta} \sum_{i=1}^{T} c_i(\theta^\star)) \leq \mathcal{O}(\sqrt{b} T^{-\frac{1}{4}}), \tag{3.3}$$

and with $\alpha = \frac{C_\theta \delta}{(C^2+1) C_s \sqrt{T}}$ and $\delta = T^{-0.25} \sqrt{\frac{C_\theta (C^2+1) C_s}{2C}}$, Alg. 5 incurs average regret:

$$\frac{1}{T}(\mathbb{E}[\sum_{i=1}^{T} c_i(\theta_i)] - \min_{\theta^\star \in \Theta} \sum_{i=1}^{T} c_i(\theta^\star)) \leq \mathcal{O}(T^{-\frac{1}{4}}), \tag{3.4}$$

for any $\theta \in \Theta$.

The proof for the above theorem can be found in Appendix 9.1.1.

The above regret analysis essentially shows that exploration in action space delivers a regret bound that is independent of parameter space dimension b, while the regret of the exploration in parameter space algorithm will have explicit polynomial dependency on feature dimension b. Converting the regret bounds to sample complexity bounds, we have that for any $\epsilon \in (0, 1)$, to achieve ϵ-average regret, Alg. 4 needs $\mathcal{O}(\frac{b^2}{\epsilon^4})$ many rounds, while Alg. 5 requires $\mathcal{O}(1/\epsilon^4)$ many rounds.

Note that in general if we have a multivariate regression problem, i.e., $a \in \mathbb{R}^p$, regret of Algorithm 5 will depend on $\sqrt{p}$ as well. But from our extreme case with $p = 1$, we clearly demonstrate the sharp advantage of exploration in action space: *when the action space's dimension is smaller than the dimension of parameter space*, we should prefer the strategy of exploration in action space.

3.4 Reinforcement Learning

In this section, we study exploration in parameter space versus exploration in action space for multi-step control problem of model-free policy search in RL. As explained in Section 3.2, we are interested in rates of convergence to a stationary point of $J(\theta)$.

3.4.1 Exploration in Parameter Space

The objective defined in Section 3.2.1 can be optimized directly over the space of parameters $\mathbb{R}^d$. Since we do not use first-order (or gradient) information about the objective, this is equivalent to derivative-free (or zeroth-order) optimization with noisy function evaluations. More specifically, for a parameter vector θ, we can execute the corresponding policy $\pi(\theta, \cdot)$ in the environment, to obtain a noisy estimate of $J(\theta)$. This noisy function evaluation can be used to construct a gradient estimate and an iterative stochastic gradient descent approach can be used to optimize the objective. An algorithm that closely follows the ones proposed in [ADX10; MGR18] and optimizes over the space of parameters is shown in Algorithm 6. Since we are working in episodic RL setting, we can use a two-point estimate to form a gradient estimation (Line 7 & 8 in Alg. 6), which in general will reduce the variance of gradient estimation [ADX10], compared to one-point estimates. We will analyze the finite rate of convergence of Algorithm 6 to a stationary point

Algorithm 6 Policy Search in Parameter Space

1: **Input:** Learning rate $\alpha \in \mathbb{R}^+$, standard deviation of exploration noise $\delta \in \mathbb{R}$
2: Initialize parameters $\theta_1 \in \mathbb{R}^d$
3: **for** $i = 1$ to T **do**
4: Sample $u \sim \mathbb{S}_d$, a d-dimensional unit sphere
5: Construct parameters $\theta_i + \delta u$, $\theta_i - \delta u$
6: Execute policies $\pi(\theta_i + \delta u, \cdot), \pi(\theta_i - \delta u, \cdot)$
7: Obtain noisy estimates of the objective $J_i^+ = J(\theta_i + \delta u) + \eta_i^+$ and $J_i^- = J(\theta_i - \delta u) + \eta_i^-$ where η_i^+, η_i^- are zero mean random i.i.d noise
8: Compute gradient estimate $g_i = \frac{d(J_i^+ - J_i^-)}{2\delta} u$
9: Update $\theta_{i+1} = \theta_i - \alpha g_i$

of the non-convex objective $J(\theta)$. First, we will lay out the assumptions and then present the

convergence analysis.

Assumptions and Analysis To analyze convergence to stationary point of a nonconvex objective, we make several assumptions about the objective. Firstly, we assume that $J(\theta)$ is differentiable with respect to θ over the entire domain. We also assume that $J(\theta)$ is G-lipschitz and L-smooth, i.e. for all $\theta_1, \theta_2 \in \mathbb{R}^d$, we have $|J(\theta_1) - J(\theta_2)| \leq G\|\theta_1 - \theta_2\|$ and $\|\nabla_\theta J(\theta_1) - \nabla_\theta J(\theta_2)\| \leq L\|\theta_1 - \theta_2\|$. Note that these assumptions are similar to the assumptions made in other zeroth-order analysis works, [FKM05; ADX10; Duc+15; Sha13; GL13; NS17].

Our analysis is along the lines of works like [GL13; NS17] that also analyze the convergence to stationary points in zeroth order non-convex optimization. The general strategy is to first construct a smoothed version of the objective $J(\theta)$, denoted as $\hat{J}(\theta) = \mathbb{E}_{v\sim\mathbb{B}_d}[J(\theta + \delta v)]$, where $\mathbb{B}_d$ is the d-dimensional unit ball. We can then show that Algorithm 6 is essentially running SGD on the objective function $\hat{J}(\theta)$, which allows us to apply standard SGD analysis on $\hat{J}(\theta)$. Lastly we link the stationary point of the smoothed objective $\hat{J}(\theta)$ to that of the objective $J(\theta)$ using the assumptions on $J(\theta)$.

Theorem 3.4.1. *Consider running Algorithm 6 for T steps where the true objective $J(\theta)$ satisfies the assumptions stated above. Then we have,*

$$\frac{1}{T}\sum_{i=1}^{T}\mathbb{E}\|\nabla_\theta J(\theta_i)\|_2^2 \leq \mathcal{O}(\mathcal{Q}^{\frac{1}{2}} d T^{\frac{-1}{2}} + \mathcal{Q}^{\frac{1}{3}} d^{\frac{2}{3}} T^{\frac{-1}{3}} \sigma) \tag{3.5}$$

where $J(\theta) \leq \mathcal{Q}$ for all $\theta \in \Theta$ and σ^2 is the variance of the random noise η in Algorithm 6.

The proof for the above theorem can be found in Appendix 9.1.2.

The above theorem gives us a convergence rate to a stationary point of policy search in parameter space. The role of variance of i.i.d noise in the noisy evaluations of the true objective is very important. Consider the case where there is little stochasticity in the environment dynamics, i.e. $\sigma \to 0$, then the first term in Equation 3.5 becomes dominant and we only need at most $\mathcal{O}(\frac{d^2\mathcal{Q}}{\epsilon^2})$ samples to reach a point θ where $\mathbb{E}\|\nabla_\theta J(\theta)\|_2^2 \leq \epsilon$. However, if there is a lot of stochasticity in the environment dynamics then the second term is dominant and we need at most $\mathcal{O}(\frac{d^2\mathcal{Q}\sigma^3}{\epsilon^3})$ samples. It is interesting to observe the direct impact that the stochasticity of environment dynamics has on convergence rate of policy search, which is also experimentally demonstrated in Sec. 3.5.2. Note that the convergence rate has no dependency on horizon length H because of the regularity assumption we used on total reward: J is always bounded by a constant $\mathcal{Q}$ that is independent of H. However, as we will see later, even under the regularity assumption convergence rate of action space exploration methods have an explicit dependence on H which will prove to be the primary reason why black-box parameter space policy search methods in [MGR18] have been so effective when compared to action space methods.

3.4.2 Exploration in Action Space

Another way to optimize the objective defined in Section 3.2.1 is to optimize over the space of actions $\mathcal{A}$. From [Sil+14], we know that for $J(\theta) = \mathbb{E}_{s\sim\mu}[V^0_{\pi_\theta}(s)]$ we can express the gradient as

$$\nabla_\theta J(\theta) = \sum_{t=0}^{H-1}\mathbb{E}_{s_t\sim d^t_{\pi_\theta}}\left[\nabla_\theta \pi(\theta, s_t)\nabla_a Q^t_{\pi_\theta}(s_t, \pi(\theta, s_t))\right] \tag{3.6}$$

Observe that the first term in the above gradient $\nabla_\theta \pi(\theta, s)$ is the Jacobian of the policy, the local linear relationship between a small change in policy parameters θ and a small change in its output, i.e., actions. The second term $\nabla_a Q(s, a)$ is actually the improvement direction at state action pair (s, a), i.e., conditioned on state s, if we move action a an infinitesimally small step along the negative gradient $-\nabla_a Q(s, a)$, we decrease the cost-to-go $Q(s, a)$. Eqn 3.6 then leverages policy's Jacobian to transfer the improvement direction in action space to an improvement direction in parameter space.

We can compute Jacobian $\nabla_\theta \pi(\theta, s)$ exactly as we have knowledge of the policy function, i.e, we can leverage the first-order information of the parameterized policy. The second term $\nabla_a Q^t_{\pi_\theta}(s, \pi(\theta, s_t))$, however, is unknown as it depends on the dynamics and cost functions and needs to be estimated by interacting with the environment. We could employ a similar algorithm as Algorithm 6, shown in Algorithm 7, to obtain an estimate of the gradient $\nabla_a Q^t_{\pi_\theta}(s, \pi(\theta, s_t))$, i.e., a zeroth order estimation of $\nabla_a Q^t_{\pi_\theta}$, computed as $\frac{p\tilde{Q}_i}{\delta} u$, where $\tilde{Q}_i$ is an unbiased estimate of $Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t) + \delta u)$, with $u \sim \mathbb{S}_p$ (Line 7 & 9 in Alg. 7).

Another important difference from Algorithm 6 is the fact that we use a one-point estimate for the gradient g_i in Algorithm 7. We cannot employ the idea of two-point estimate in random exploration in action space to reduce the variance of the estimate of $\nabla_a Q^t_{\pi_\theta}(s_t, a)$. This is due to the fact that environment is stochastic, and we cannot guarantee that we will reach the same state s_t at any two independent roll-ins with π_θ at time step t. Similar to Section 3.4.1, we will

Algorithm 7 Policy Search in Action Space

1: **Input:** Learning rate $\alpha \in \mathbb{R}^+$, standard deviation of exploration noise $\delta \in \mathbb{R}$, Horizon length H, Initial state distribution μ
2: Initialize parameters $\theta_1 \in \mathbb{R}^d$
3: **for** $i = 1$ to T **do**
4: Sample $u \sim \mathbb{S}_p$, a p-dimensional unit sphere
5: Sample uniformly $t \in \{0, \cdots, H-1\}$
6: Execute policy $\pi(\theta_i, \cdot)$ until $t-1$ steps
7: Execute perturbed action $a_t = \pi(\theta_i, s_t) + \delta u$ at timestep t and continue with policy $\pi(\theta_i, \cdot)$ until timestep $H-1$ to obtain an estimate $\tilde{Q}_i = Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t) + \delta u) + \tilde{\eta}_i$ where $\tilde{\eta}_i$ is zero mean random noise
8: Compute policy Jacobian $\Psi_i = \nabla_\theta \pi(\theta_i, s_t)$
9: Compute gradient estimate $g_i = H \Psi_i \frac{p\tilde{Q}_i}{\delta} u$
10: Update $\theta_{i+1} = \theta_i - \alpha g_i$

analyze the rate of convergence of Algorithm 7 to a stationary point of the objective $J(\theta)$. The following section will lay out the assumptions and present the convergence analysis.

Assumptions and Analysis The assumptions for policy search in action space are similar to the assumptions in Section 3.4.1. We assume that $J(\theta)$ is differentiable with respect to θ over the entire domain. We also assume that $J(\theta)$ is G-lipschitz and L-smooth. In addition to these assumptions, we will assume that the policy function $\pi(\theta, s)$ is K-lipschitz in θ and the state-action value function $Q^t_{\pi_\theta}(s, a)$ is W-lipschitz and U-smooth in a. Finally, we assume that the state-action value function $Q(s, a)$ is differentiable with respect to a over the entire domain. Note that the Lipschitz assumptions above on $J(\theta)$, $Q^t_{\pi_\theta}(s, a)$, and $\pi(\theta, s)$ are also used in the analysis

of Deterministic policy gradient [Sil+14]. We need extra smoothness assumption to study the convergence of our algorithms.

Note that the gradient estimate g_i used in Algorithm 7 is a biased estimate of $\nabla_\theta J(\theta)$. We can show this by considering

$$\mathbb{E}_i[g_i] = \mathbb{E}_t \mathbb{E}_{s_t \sim d^t_{\pi_{\theta_i}}} \left[H \nabla_\theta \pi(\theta_i, s_t) \mathbb{E}_{u \sim \mathbb{S}_p} \left[\frac{p\tilde{Q}_i}{\delta} u \right] \right]$$

where $\mathbb{E}_i$ denotes expectation with respect to the randomness at iteration i. From [FKM05], we have that $\mathbb{E}[\frac{p\tilde{Q}_i}{\delta} u] = \nabla_a \mathbb{E}_{v \sim \mathbb{B}_p}[Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t) + \delta v)]$ so we can rewrite the above equation as

$$\mathbb{E}[g_i] = \sum_{t=0}^{H-1} \mathbb{E}_{s_t \sim d^t_{\pi_{\theta_i}}} \mathbb{E}_{v \sim \mathbb{B}_p} \left[\nabla_\theta \pi(\theta_i, s_t) \nabla_a Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t) + \delta v) \right]$$

Comparing the above expression with equation 3.6, we can see that g_i is not an unbiased estimate of the gradient $\nabla_\theta J(\theta)$. We can also explicitly upper bound the variance of g_i by $\mathbb{E}_i \|g_i\|_2^2$. Note that in the limit when $\delta \to 0$, g_i becomes an unbiased estimate of $\nabla_\theta J(\theta)$, but the variance will approach to infinity. In our analysis, we explicitly tune δ to balance the bias and variance.

Theorem 3.4.2. *Consider running Algorithm 7 for T steps where the objective $J(\theta)$ satisfies the assumptions stated above. Then, we have*

$$\frac{1}{T} \sum_{i=1}^{T} \mathbb{E} \|\nabla_\theta J(\theta_i)\|_2^2 \leq \mathcal{O}(T^{-\frac{1}{4}} H p^{\frac{1}{2}} (\mathcal{Q}^3 + \sigma^2 \mathcal{Q})^{\frac{1}{4}}) \tag{3.7}$$

where $J(\theta) \leq \mathcal{Q}$ for all $\theta \in \Theta$ and σ^2 is the variance of the random noise $\tilde{\eta}$ in Algorithm 7.

The proof for the above theorem can be found in Appendix 9.1.3.

The above theorem gives us a convergence rate to a stationary point of $J(\theta)$ for policy search in action space. This means that to reach a point θ where $\mathbb{E}\|\nabla_\theta J(\theta)\|_2^2 \leq \epsilon$, policy search in action space needs at most $\mathcal{O}\left(\frac{p^2 H^4}{\epsilon^4}(\mathcal{Q}^3 + \sigma^2 \mathcal{Q})\right)$ samples. Interestingly, the convergence rate has a dependence on the horizon length H, unlike policy search in parameter space. Also, observe that the convergence rate has no dependence on the parameter dimensionality d as we have complete knowledge of the Jacobian of policy, and we have a dependence on stochasticity of the environment σ that slows down the convergence as the stochasticity increases, similar to policy search in parameter space.

3.5 Experiments

Given the analysis presented in the previous sections, we test the convergence properties of parameter and action space policy search approaches across several experiments: Contextual Bandit with rich observations, Linear Regression, RL benchmark tasks and Linear Quadratic Regulator (LQR). We use Augmented Random Search (ARS), from [MGR18], as the policy search in parameter space method in our experiments as it has been empirically shown to be effective in RL tasks. For policy search in action space, we use either REINFORCE [Wil92], or ExAct (Exploration in Action Space), the method described by Algorithm 7. In all the plots shown,

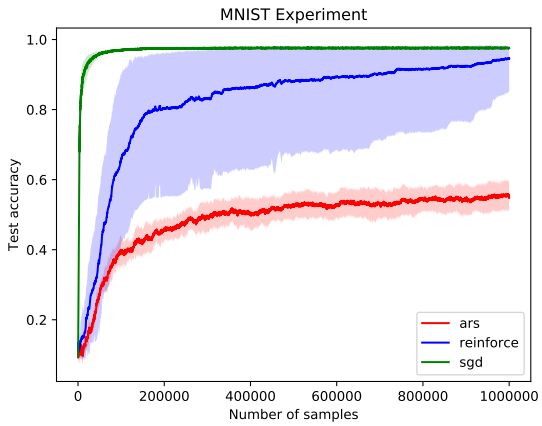

Figure 3.1: Mean test accuracy with standard error for different approaches against number of samples

solid lines represent the mean estimate over 10 random seeds and shaded regions correspond to ± 1 standard error. The code for all our experiments can be found here[1][2].

3.5.1 One-Step Control

In these sets of experiments, we test the convergence rate of policy search methods for one time-step prediction. The objective is to minimize the instantaneous cost incurred. The motivation behind such experiments is that we want to understand the dependence of different policy search methods on parametric dimensionality d without the effect of horizon length H.

MNIST as a Contextual Bandit Our first set of experiments is the MNIST digit recognition task [LeC+98]. To formulate the task in an RL framework, we consider a sequential decision making problem where at each time-step the agent is given the features of the image and needs to predict one of ten actions (corresponding to digits). A reward of $+1$ is given for predicting the correct digit, and a reward of -1 for an incorrect prediction. With this reduction, the problem is essentially a Contextual Bandit Problem [Aga+14]. We use a standard LeNet-style convolutional architecture, [LeC+98], with $d = 21840$ trainable parameters. Figure 3.1 shows the learning curves for SGD under standard full-information supervised learning setting with cross entropy loss, REINFORCE and ARS. We can observe that in this setting where the parameter space dimensionality d significantly exceeds the action space dimensionality $p = 1$, policy search in action space outperforms parameter space methods.

Linear Regression with Partial Information These set of experiments are designed to understand how the sample complexity of different policy search methods vary as the parametric complexity is varied. More specifically, from our analysis in Section 3.3, we know that sample complexity of parameter space methods have a dependence on d, the parametric complexity, whereas action space methods have no dependence on d. We test this hypothesis in this experiment using artificial data with varying input dimensionality and output scalar values. Figure 9.3

[1] https://github.com/LAIRLAB/contrasting_exploration_rl
[2] https://github.com/LAIRLAB/ARS-experiments

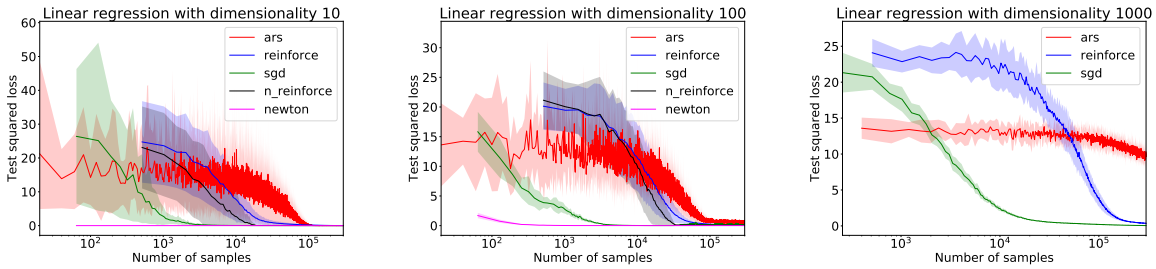

Figure 3.2: Linear Regression Experiments with varying input dimensionality

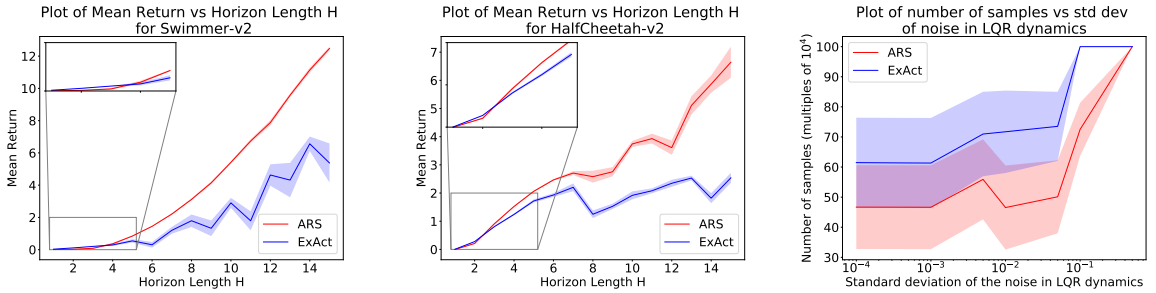

Figure 3.3: Multi-step Control. The left and middle figures show performance of different methods as horizon length varies. The right figure shows number of samples needed to reach close to a stationary point as noise in dynamics varies

shows the learning curves for standard full-information supervised learning approaches with full access to the square loss (SGD & Newton), REINFORCE, natural REINFORCE [Kak02], and ARS as we increase the input dimensionality, and hence parametric dimensionality d. Note that we have not included natural REINFORCE and Newton method in Figure 9.3 (right) as extensive hyperparameter search for these methods is computationally expensive in such high dimensionality settings. The learning curves in Figure 9.3 match our expectations, and show that action space policy search methods do not degrade as parametric dimensionality increases whereas parameter space methods do. Moreover, action space methods lie between the curves of supervised learning and parameter space methods as they take advantage of the Jacobian of the policy and learn more quickly than parameter space methods.

3.5.2 Multi-Step Control

The above experiments provide insights on the dependence of policy search methods on parametric dimensionality d. We now shift our focus on to the dependence on horizon length H. In this set of experiments, we extend the time horizon and test the convergence rate of policy search methods for multi-step control. The objective is to minimize the sum of costs incurred over a horizon H, i.e. $J(\theta) = \mathbb{E}[\sum_{t=1}^{T} c(s_t, a_t)]$. According to our analysis, we expect action space policy search methods to have a dependence on the horizon length H.

We test ARS and ExAct on two popular continuous control simulated benchmark tasks in OpenAI gym [Bro+16a]: Swimmer and HalfCheetah. We chose these two environments as they allow you to vary the horizon length H without terminating the task early. For both tasks, we use linear policies as they have been shown to be very effective in [MGR18; Raj+17]. Swimmer

has an observation space dimensionality of $d = 8$ and a continuous action space of dimensionality $p = 2$. Similarly, for HalfCheetah $d = 17$ and $p = 6$. Figure 3.3 (left and middle) show the performance of both approaches in terms of the mean return $J(\theta)$ (expected sum of rewards) they obtain as the horizon length H varies. Note that both approaches are given access to the same number of samples $10^4 \times H$ from the environments for each horizon length H. In the regime of short horizon lengths, action space methods are better than parameter space methods as they do not have a dependence on parametric complexity d. However, as the horizon length increases, parameter space methods start outperforming action space methods handily as they do not have an explicit dependence on the horizon length, as pointed out by our analysis. We have observed the same trend of parameter space methods handily outperforming action space methods as far as $H = 200$ and expect this trend to continue beyond. This empirical insight combined with our analysis presented in Sections 3.4.1, 3.4.2 explains why ARS, a simple parameter space search method, outperformed state-of-the-art actor critic action space search methods in [MGR18] on OpenAI gym benchmarks where the horizon length H is typically as high as 1000.

Effect of environment stochasticity In this final set of experiments, we set out to understand the effect of stochasticity in environment dynamics on the performance of policy search methods. As our analysis in Sections 3.4.1 and 3.4.2 points out, the stochasticity of the environment plays an important role in controlling the variance of our gradient estimates in zeroth order optimization procedures. To empirically observe this, we use a stochastic LQR environment where we have access to the true cost function c and hence, can compute the gradient $\nabla_\theta J(\theta)$ exactly. Given access to such information, we vary the standard deviation σ of the noise in LQR dynamics and observe the number of samples needed for ARS to reach θ such that $\|\nabla_\theta J(\theta)\|_2^2 \leq 0.05$. Figure 3.3 (right) presents the number of samples needed to reach close to a stationary point of $J(\theta)$ as the standard deviation of noise in LQR dynamics varies. Note that we limit the maximum number of samples to 10^6 for each run. The results match our expectations from the analysis, where we observed that as the stochasticity of the environment increases, convergence rate of policy search methods slows down.

3.6 Conclusion

Parameter space exploration via black-box optimization methods have often been shown to outperform sophisticated action space exploration approaches for the reinforcement learning problem. Our work highlights the major difference between parameter and action space exploration methods: the latter leverages Jacobian of the parameterized policy. This allows sample complexity of action space exploration methods to be independent of parameter space dimensionality and only dependent on the dimensionality of action space and horizon length. For domains where the action space dimensionality and horizon length are small and the dimensionality of parameter space is large, we conclude that exploration in action space should be preferred. On the other hand, for long horizon control problems with low dimensional policy parameterization, exploration in parameter space will outperform exploration in action space.

Chapter 4

Planning and Execution using Inaccurate Models

For example, if in a car our current policy drives a maneuver too far to the left, ... even a very poor model of the car can then be used to tell us that the change we should make is to turn the steering wheel clockwise (rather than anti-clockwise) to correct for this error.

Pieter Abbeel, Morgan Quigley, Andrew Ng (2006)

This chapter presents our first algorithmic contribution in this thesis. The algorithm presented is in the spirit of this thesis which advocates using online experience to adapt the behavior of the planner rather than updating the dynamics of the model. The algorithm perfectly captures the intuition that figuring out something is wrong is much easier than knowing how to fix it. While the idea behind the proposed algorithm is quite simple, the guarantees provided are non-trivial and it works very well in practice. This algorithm is the first approach in literature to interleave planning and execution using inaccurate dynamical model, with provable guarantees on completing the task without requiring any updates to the dynamics of the model. This chapter is adapted from our original paper [Vem+20].

4.1 Introduction

Modern robotic planning approaches involve use of models that tend to be sophisticated and complex. These models are used to simulate the dynamics of the real world and foresee the outcomes of actions executed. From using fast analytical solvers to generate motion primitives on-the-fly [Coh+11] to simulators that do reasoning based on physics, and optimization to resolve contacts [TET12], these models are getting better at modeling the dynamics of the real world. However, real world robotic tasks are rife with situations that cannot be predicted and therefore, modeled before execution. Thus, we need a planning approach that can use potentially inaccurate models and still complete the task.

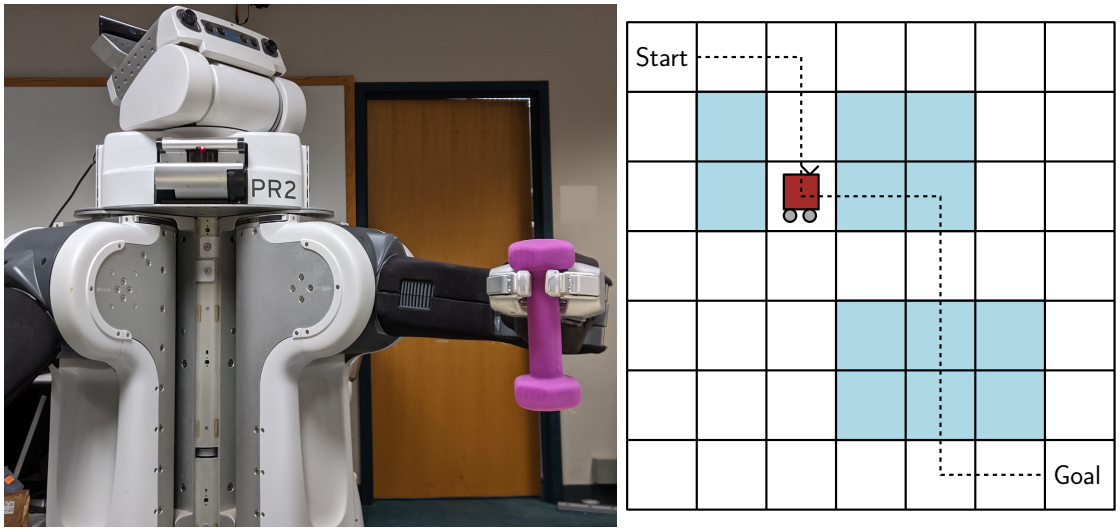

Figure 4.1: (left) PR2 executing a pick-and-place task with a heavy object that is modeled as light, resulting in hitting joint torque limits during execution. (right) Mobile robot navigating a gridworld with icy states, where the robot slips, that are not modeled as icy resulting in discrepancy in dynamics.

For example, consider the task depicted in Figure 5.1 (left) where a robotic arm needs to pick an object and place it at a goal location. Without knowledge of the mass of the object, the model can be inaccurate in simulating the dynamics. If the object is modeled as light, the planned path would pick it to a certain height before placing it at the goal location. However, if the object is heavy in the real world, like in Figure 5.1 (left), this plan cannot be executed as the joint torque limits are reached and the arm cannot move higher. Thus, by using the inaccurate model for planning, the arm is stuck and cannot reach the goal. Figure 5.1 (right) presents another simple scenario where a mobile robot is navigating a gridworld containing icy states, where the robot slips, i.e. if the robot tries to go right or left in an icy state, it will move two cells rather than one cell in that direction. However, the model used for planning does not model the icy states and hence, cannot simulate the real world dynamics correctly. This can lead to highly suboptimal paths or sometimes even inability to reach the goal, when using such a model for planning.

A typical solution to this problem is to update the dynamics of the model and replan [Sut91]. However, this is often impossible in real world planning problems where we use models that are complex and in some cases obtained from expensive computation that is done offline before execution [Hau+06]. The dynamics of these models cannot be changed online arbitrarily without deteriorating their simulation capabilities in other scenarios and sacrificing real-time execution. In addition, this solution might require us to have the knowledge of what part of the model dynamics is inaccurate and how to correct it. Going back to the pick-and-place example in Figure 5.1, to update the model we need to first identify that the modeled mass is incorrect and then estimate the true mass to correct the dynamics of the model. Both of these steps require specialized non-trivial implementations. Finally, in the case of models that *can* be updated online efficiently, it might still not be possible to model the true dynamics without an unreasonably large number of online executions because the true dynamics are often very complex, e.g. modeling cooperative navigation dynamics in human crowds [VMO17]. The above aspects make the solution of updating model dynamics online undesirable in real world robotic tasks, where we are interested

in completing the task and *not* in modeling the dynamics accurately.

In this work, we present an alternative approach CMAX for interleaving planning and execution that does not require updating the dynamics of the model. Instead during execution, whenever we discover an action where the dynamics differ between the real world and the model, we update the cost function to penalize executing such state-action pairs in the future. This biases the planner to replan paths that do not consist of such state-action pairs, and thereby avoid regions of state-action space where the dynamics are known to differ. Based on this idea, we present algorithms for both small state spaces, where we can do exact planning, and large state spaces, including continuous state spaces, where we resort to function approximation to update the cost function and to maintain cost-to-go estimates. Our framework CMAX comes with provable guarantees on reaching the goal, without any resets, under specific assumptions on the model. The proposed algorithms are tested on a range of tasks including simulated 4D planar pushing as well as physical robot 3D pick-and-place task where the mass of the object is incorrectly modeled, and 7D arm planning tasks when one of the joints is not operational, leading to discrepancy in dynamics.

4.2 Preliminaries

We are interested in the deterministic shortest path problem represented by the tuple $M = (\mathbb{S}, \mathbb{A}, \mathbb{G}, f, c)$ where $\mathbb{S}$ denotes the state space, $\mathbb{A}$ denotes the action space, $\mathbb{G} \subseteq \mathbb{S}$ is the non-empty set of goal states we are interested in reaching, $f : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ denotes the deterministic dynamics governing the transition to next state given current state and action, and $c : \mathbb{S} \times \mathbb{A} \to [0, 1]$ is the cost function. For the purposes of this work, we will focus on small discrete action spaces, bounded costs lying between 0 and 1[1], and a cost-free termination goal state i.e. for all $g \in \mathbb{G}$, we have $c(g, a) = 0$ and $f(g, a) = g$ for all actions $a \in \mathbb{A}$. The objective of the shortest path problem is to find the least-cost path from any given start state $s_0 \in \mathbb{S}$ to any goal state $g \in \mathbb{G}$ in M. We assume that there exists at least one path from each state $s \in \mathbb{S}$ to one of the goal states $g \in \mathbb{G}$ in M, and that the cost of any transition starting from a non-goal state is positive i.e. $c(s, a) > 0$ for all $s \in \mathbb{S} \setminus \mathbb{G}, a \in \mathbb{A}$. These assumptions are typical for analysis in deterministic shortest path problems [Ber05]. We use $V(s)$ to denote the cost-to-go estimate of any state $s \in \mathbb{S}$ and $V^*(s)$ to denote the optimal cost-to-go. From dynamic programming literature [Ber05], we know that the optimal cost-to-go satisfies the Bellman optimality condition $V^*(s) = \min_{a \in \mathbb{A}}[c(s, a) + V^*(f(s, a))]$. A cost-to-go estimate V is called admissible if it underestimates the optimal cost-to-go $V(s) \leq V^*(s)$ for all $s \in \mathbb{S}$, and is called consistent if it satisfies the condition that for any state-action pair $(s, a), s \notin \mathbb{G}$, $V(s) \leq c(s, a) + V(f(s, a))$, and $V(g) = 0$ for all $g \in \mathbb{G}$.

In this work, we assume that the exact dynamics are initially unknown to the robot, and can only be discovered through executions. Thus, instead of offline planning methods, we need online methods that interleave planning with action execution. Specifically, we focus on the online real-time planning setting where the robot does not have access to resets, and the robot has to interleave planning and execution to ensure real-time operation. This is similar to the classical real-time search setting considered by works like LRTA* [Kor90], RTAA* [KL06], RTDP [BBS95] and several others. An important aspect of these approaches is that the robot can only perform a fixed amount of computation for planning, independent of the size of state space, before it has to execute an action.

[1] Any bounded non-negative cost can be scaled to fit this assumption

4.3 Problem Setup

Consider the problem of a robot acting to find a least-cost path to a goal in an environment represented by the tuple $M = (\mathbb{S}, \mathbb{A}, \mathbb{G}, f, c)$ with unknown deterministic dynamics f and known cost function c. The robot gathers knowledge of the dynamics over a single trajectory in the environment, and does not have access to any resets, ruling out any episodic approach. This is an extremely challenging setting as the robot has to reason about whether to exploit its current knowledge of the dynamics to act near-optimally or to explore to gain more knowledge of the dynamics, possibly at the expense of suboptimality.

We assume that the agent has access to an approximate model, $\hat{M} = (\mathbb{S}, \mathbb{A}, \mathbb{G}, \hat{f}, c)$, that it can use to simulate the outcome of its actions and use for planning. In our motivating gridworld example (Figure 5.1 right), this model represents a grid with no icy states, so the dynamics $\hat{f}$ moves the robot to the next cell based on the executed action without any slip. However, the real environment contains icy states resulting in dynamics f that differ on state-action pairs where the state is icy. For the remainder of this paper, we will refer to such state-action pairs where f and $\hat{f}$ differ as "incorrect" state-action pairs, and use the notation $\mathcal{X} \subseteq \mathbb{S} \times \mathbb{A}$ to denote the set of "incorrect" state-action pairs, i.e. $f(s, a) \neq \hat{f}(s, a)$ for all $(s, a) \in \mathcal{X}$. The objective is for the robot to reach a goal state from a given start state, despite using an inaccurate model for planning, while minimizing the cost incurred and ensuring real-time execution.

4.4 Approach

Existing planning and learning approaches try to learn a very good approximation of M from scratch through online executions [KS02; BT02; JS07; DFR15], or update the dynamics of model $\hat{M}$ so that it approximates M well [AQN06a; Jia18; RKK18]. In this work, we propose an approach CMAX that uses the inaccurate model $\hat{M}$ online *without* updating its dynamics, and is provably guaranteed to complete the task. In a nutshell, instead of learning a new dynamics model from scratch or updating the dynamics of existing model, CMAX maintains a running estimate of the set $\mathcal{X}_t$ consisting of all state-action pairs that have been executed and have been discovered to be incorrect until timestep t. Using the set $\mathcal{X}_t$, we update the cost function to bias the planner to plan future paths that avoid state-action pairs that are known to be incorrect. It is important to note that the challenge of dealing with exploration-exploitation dilemma online still exists, as we do not know the set of state-action pairs $\mathcal{X}$ where the dynamics differ ahead of online execution. A similar approach was proposed in [Jia18] for the episodic setting where the robot had access to resets, and for small state spaces where we could perform full state space planning. CMAX extends it to the significantly more challenging online real-time setting and we present a practical algorithm for large state spaces.

4.4.1 Penalized Model

We formalize our approach as follows: Given a model $\hat{M}$ and a set $\mathcal{X} \subseteq \mathbb{S} \times \mathbb{A}$ consisting of state-action pairs that have been discovered to be incorrect so far, define the penalized model $\tilde{M}_{\mathcal{X}}$ as:

Definition 4.4.1 (Penalized Model)**.** *The penalized model $\tilde{M}_{\mathcal{X}} = (\mathbb{S}, \mathbb{A}, \mathbb{G}, \hat{f}, \tilde{c}_{\mathcal{X}})$ has the same state space, action space, set of goals, and dynamics as $\hat{M}$. The cost function $\tilde{c}_{\mathcal{X}}$ though is defined*

Algorithm 8 Limited-Expansion Search based on RTAA*[KL06]

1: **function** SEARCH($s, \tilde{M}_{\mathcal{X}}, V, K$)
2: Initialize $g(s) \leftarrow 0$
3: Initialize min-priority open list O, and closed list C
4: Add s to open list O with priority $g(s) + V(s)$
5: **for** $i = 1, 2, \cdots, K$ **do**
6: Pop s_i from open list O
7: If $s_i \in \mathbb{G}$, then $s_{\mathsf{best}} \leftarrow s_i$ and move to Line 19
8: **for** $a \in \mathbb{A}$ **do** ▷ *Expanding state* s_i
9: Get successor $s' = \hat{f}(s_i, a)$
10: If $s' \in C$, continue to next action
11: **if** $s' \in O$ and $g(s') > g(s_i) + \tilde{c}_{\mathcal{X}}(s_i, a)$ **then**
12: Update $g(s') \leftarrow g(s_i) + \tilde{c}_{\mathcal{X}}(s_i, a)$
13: Reorder open list O
14: **else if** $s' \notin O$ **then**
15: Set $g(s') \leftarrow g(s_i) + \tilde{c}_{\mathcal{X}}(s_i, a)$
16: Add s' to O with priority $g(s') + V(s')$
17: Add s_i to the closed list C
18: Pop s_{best} from open list O
19: **for** $s' \in C$ **do**
20: Update $V(s') \leftarrow g(s_{\mathsf{best}}) + V(s_{\mathsf{best}}) - g(s')$
21: Backtrack from s_{best} to s, and set a_{best} as the first action on path from s to s_{best}
return a_{best}

as $\tilde{c}_{\mathcal{X}}(s, a) = |\mathbb{S}|$ *if* $(s, a) \in \mathcal{X}$, *else* $\tilde{c}_{\mathcal{X}}(s, a) = c(s, a)$.[2]

Intuitively, the penalized model $\tilde{M}_{\mathcal{X}}$ has a very high cost for any transition where the dynamics differ, i.e. $(s, a) \in \mathcal{X}$, and the same cost as the model $\hat{M}$ otherwise. More specifically, the cost is inflated to the size of the statespace, which is the maximum cost of a path that visits all states[3] (remember, that our cost is normalized to lie within 0 and 1.) This biases the planner to "explore" all other state-action pairs that are not yet known to be incorrect before it plans a path through an incorrect state-action pair. In the next section, we will describe how we use the penalized model $\tilde{M}_{\mathcal{X}}$ for real-time planning.

4.4.2 Limited-Expansion Search for Planning

During online execution, the robot has to constantly plan the next action to execute from its current state in real-time. This forces the robot to use a fixed amount of computation for planning before it has to execute the best action found so far. In this work, we use a real-time search method that is adapted from RTAA* proposed by [KL06].

The planner is summarized in Algorithm 8. At any timestep t, given the current penalized model $\tilde{M}_{\mathcal{X}_t}$ and the current state s_t, the planner constructs a lookahead search tree using at most K state expansions. We obtain the successors of any expanded state and the cost of any state-action pair using the penalized model $\tilde{M}_{\mathcal{X}_t}$. After expanding K states, it finds the best state s_{best}

[2]This is similar to the notion of penalized MDP, introduced in [Jia18]
[3]Hence, the name CMAX for our approach

Algorithm 9 CMAX – Small State Spaces

Initialize $\hat{M}_1 \leftarrow \hat{M}$, $\mathcal{X}_1 \leftarrow \{\}$, start state $s_1 \in \mathbb{S}$, cost-to-go estimates V, number of expansions K, $t \leftarrow 1$
while $s_t \notin \mathbb{G}$ **do**
  Get $a_t = \texttt{SEARCH}(s_t, \hat{M}_t, V, K)$
  Execute a_t in environment M to get $s_{t+1} = f(s_t, a_t)$
  if $s_{t+1} \neq \hat{f}(s_t, a_t)$ **then**
    Add (s_t, a_t) to the set : $\mathcal{X}_{t+1} \leftarrow \mathcal{X}_t \cup \{(s_t, a_t)\}$
    Update the penalized model : $\hat{M}_{t+1} \leftarrow \tilde{M}_{\mathcal{X}_{t+1}}$
  else
    $\mathcal{X}_{t+1} \leftarrow \mathcal{X}_t$, $\hat{M}_{t+1} \leftarrow \hat{M}_t$
  $t \leftarrow t + 1$

among the leaves of the search tree that has the least sum of cost-to-come from s_t and cost-to-go to a goal state (line 18 in Algorithm 8). The best action to execute in the current state s_t is chosen to be the first action on the path from s_t to s_{best} in the search tree and the cost-to-go estimates of all expanded states are updated as: $V(s_{\mathsf{expanded}}) = g(s_{\mathsf{best}}) + V(s_{\mathsf{best}}) - g(s_{\mathsf{expanded}})$, where $g(s)$ is the cost-to-come from s_t for any state s in the search tree. The amount of computation used to compute the best action for the current state is bounded as a factor of the number of expansions K in the search tree. Thus, we can bound the planning time and ensure real-time operation for our robot.

4.4.3 Warm Up: Small State Spaces

In this section, we will present an algorithm that is applicable for small discrete state spaces where it is feasible to maintain cost-to-go estimates for all states $s \in \mathbb{S}$ using a tabular representation, and we can maintain a running set $\mathcal{X}_t$ containing all the discovered incorrect state-action pairs so far, without resorting to function approximation. The algorithm[4] is shown in Algorithm 12. Intuitively, Algorithm 12 maintains a running set of incorrect state-action pairs $\mathcal{X}_t$, updates the set whenever it encounters an incorrect state-action pair, and recomputes the penalized model $\tilde{M}_{\mathcal{X}_t}$. Crucially, the algorithm never updates the dynamics of the model $\hat{M}$, and only updates the cost function according to Definition 4.4.1. In order to prove completeness, we assume the following:

Assumption 4.4.1. *Given a penalized model $\tilde{M}_{\mathcal{X}_t}$ and the current state s_t at any timestep t, there always exists at least one path from s_t to a goal state that does not contain any state-action pairs (s, a) that are known to be incorrect, i.e. $(s, a) \in \mathcal{X}_t$.* [5]

A Closer Look at the Assumption Observe that this assumption is on both the quality of the initial model $\hat{M}$ and the operation of our algorithm. We would like to emphasize that this assumption is less restrictive than the following assumption:

Assumption 4.4.2. *For any state $s \in \mathbb{S}$, there always exists at least one path from s to a goal state that does not contain any incorrect state-action pairs, i.e. $(s, a) \in \mathcal{X}^*$.*

[4] A similar algorithm in the episodic setting with full state space planning is presented in [Jia18]

[5] This assumption is less restrictive than the assumption that there exists at least one path from the current state to a goal that does not contain any state-action pairs (s, a) that are incorrect i.e. $(s, a) \in \mathcal{X}$

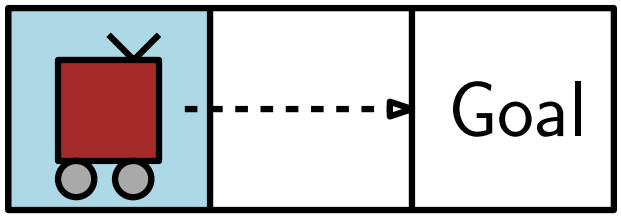

Figure 4.2: A small icy gridworld where Assumption 4.4.1 is satisfied but Assumption 4.4.2 is not satisfied.

It is easy to see that Assumption 4.4.2 implies Assumption 4.4.1 since for any timestep t, $\mathcal{X}_t \subseteq \mathcal{X}^*$. We will show that Assumption 4.4.1 does not imply Assumption 4.4.2 through an example: Recall the motivating icy gridworld from Figure 5.1 and consider the small icy gridworld shown in Figure 4.2 (right). Assumption 4.4.2 is clearly violated since there exists no path from robot's current state to the goal that does not contain any incorrect state-action pair. However, Assumption 4.4.1 is not violated since the action of moving right in the robot's current state is not yet discovered to be incorrect. In fact, once it executes the move right action it immediately reaches the goal state since the robot slips on ice and moves two cells to the right. Thus, Assumption 4.4.1 does not imply Assumption 4.4.2.

Assumption 4.4.1 requires that there exists at least one path from the *current state* to a goal state that does not contain any transition that is known to be incorrect. In other words, it should hold for every time step t before the robot reaches the goal. Hence, it is an assumption on both the quality of the approximate model and the execution trace (the states visted during execution) of CMAX. This makes the assumption hard to verify prior to execution as it is dependent on the operation of the algorithm under true dynamics.

We can also interpret this assumption as a variant of the *optimistic model* assumption that is presented in Section 5.4.4 and is introduced in [Jia18]. Specifically, an optimistic model is an approximate model $\hat{M}$ whose optimal cost-to-go under its dynamics $\hat{f}$ at every state $s \in \mathbb{S}$ underestimates teh optimal cost-to-go under true dynamics f. A popular example of the optimistic model assumption is the free space assumption [Zel92] that is typically used in tasks involving navigation in unknown environments, where we assume that any unknown region in the map is free of obstacles. Assumption 4.4.1 can be interpreted as requiring that the model $\hat{M}_t$ remains optimistic in Algorithm 12 until the robot reaches the goal. Unfortunately, this interpretation does not make the assumption easier to verify prior to execution as it still depends on the execution trace of CMAX.

Under this assumption, we can show the following guarantee for Algorithm 12:

Theorem 4.4.1. *Assume Assumption 4.4.1 holds then, if $\mathcal{X}$ denotes the set consisting of all incorrect state-action pairs, and the initial cost-to-go estimates used are admissible and consistent, then using Algorithm 12 the robot is guaranteed to reach a goal state in at most $|\mathbb{S}|^2$ timesteps. Furthermore, if we allow for $K = |\mathbb{S}|$ expansions, then we can guarantee that the robot will reach a goal state in at most $|\mathbb{S}|(|\mathcal{X}| + 1)$ timesteps.*

Proof Sketch. From [KL06] Theorem 3 and Assumption 4.4.1, we have that using RTAA*, the robot is guaranteed to reach a goal state. Combining this result with the $|\mathbb{S}|^2$ upper bound on the number of timesteps it takes for LRTA* (which is equivalent to RTAA* with $K = 1$ expansion) to reach the goal from [KS93], we have that using Algorithm 12 a robot is guaranteed to reach a goal state in at most $|\mathbb{S}|^2$ timesteps.

To prove the second part, observe that when we do $K = |\mathbb{S}|$ expansions at any timestep t in RTAA* and update the cost-to-go, we obtain the optimal cost-to-go V^* for the penalized model $\tilde{M}_{\mathcal{X}_t}$. Once we obtain the optimal cost-to-go, there will be no further cost-to-go updates in subsequent timesteps until we either discover an incorrect state-action pair or reach the goal. Since the number of incorrect (s, a) pairs is $|\mathcal{X}|$ and the length of the longest path is bounded above by $|\mathbb{S}|$, using pigeon hole principle we have that the robot is guaranteed to reach the goal in at most $|\mathbb{S}|(|\mathcal{X}| + 1)$ timesteps. □

The above theorem establishes that using Algorithm 12, the robot is guaranteed to reach a goal state under Assumption 4.4.1. In practice, we observe that the number of timesteps to reach a goal has a smaller dependence on the size of state space than the worst-case bound, especially if Algorithm 12 starts with cost-to-go estimates that are reasonably accurate for the initial model $\hat{M}$.

4.4.4 Large State Spaces

In large state spaces, it is infeasible to maintain cost-to-go estimates for all states $s \in \mathbb{S}$ using a tabular representation and maintain a running estimate of the set $\mathcal{X}_t$, as both could be very large in size. Thus, we will need to resort to function approximations for both cost-to-go estimates and the set $\mathcal{X}_t$.

We will assume existence of a fixed distance metric $d : \mathbb{S} \times \mathbb{S} \to \mathbb{R}^+ \cup \{0\}$, and that $\mathbb{S}$ is bounded under this metric. We relax the definition of $\mathcal{X}$ using the distance metric d as follows: Define any state-action pair $(s, a) \in \mathcal{X}^\xi$ to be ξ-incorrect if $d(f(s, a), \hat{f}(s, a)) > \xi$ where $\xi \geq 0$. We assume that there is an underlying path following controller that is used to execute our plan and can deal with discrepancies smaller than ξ. Thus, we allow for small discrepancies in our approximate model $\hat{M}$ that can be resolved using a low-level controller.

Our algorithm for large state spaces is presented in Algorithm 13. The main idea of the algorithm is to "cover" the set $\mathcal{X}^\xi$ using hyperspheres in $\mathbb{S} \times \mathbb{A}$. Since the action space $\mathbb{A}$ is a discrete set, we maintain separate sets of hyperspheres for each action $a \in \mathbb{A}$. Whenever the agent encounters an incorrect state-action pair $(s, a) \in \mathcal{X}^\xi$, it places a hypersphere at s corresponding to action a whose radius (as measured by the metric d) is given by $\delta > 0$, a domain-dependent constant. We inflate the cost of a state-action pair (s, a), according to Definition 4.4.1, if s lies inside any hypersphere corresponding to action a. In practice, this is implemented by constructing separate KD-Trees in state space $\mathbb{S}$ for each action $a \in \mathbb{A}$ to enable efficient lookup.

After executing the action and placing a hypersphere if a discrepancy in dynamics was observed, the function approximation for cost-to-go is updated iteratively as follows (Line 15 to Line 17): Sample a batch of states from the buffer of previously visited states with replacement, construct a lookahead tree for each state in the batch (through parallel jobs) to obtain all states on the closed list and their corresponding cost-to-go updates using Algorithm 8, and finally update the parameters of the cost-to-go function approximator to minimize the mean squared loss $\mathcal{L}(V_\theta, \mathbb{X}) = \frac{1}{2|\mathbb{X}|} \sum_{(s,V(s))\in\mathbb{X}} (V(s) - V_\theta(s))^2$ for all the expanded states through a gradient descent step (Line 17).

Observe that, similar to Algorithm 12, we do not update the dynamics $\hat{f}$ of the model, and only update the cost function according to Definition 4.4.1. However, unlike Algorithm 12, we do not explicitly maintain a set of incorrect state-action pairs but maintain it implictly through hyperspheres. By using hyperspheres, we obtain local generalization and increase the cost of all the state-action pairs inside a hypersphere. In addition, unlike Algorithm 12, we update cost-to-

Algorithm 10 CMAX – Large State Spaces

Initialize $\hat{M}_1 \leftarrow \hat{M}$, Cost-to-go function approximation V_{θ_1}, Set of hyperspheres $\mathcal{X}_1^\xi \leftarrow \{\}$, Start state s_1, Number of planning updates N, Batch size B, Buffer $\mathcal{D}$, Number of expansions K, Learning rate η, $t \leftarrow 1$, Radius of hypersphere δ, Discrepancy threshold ξ
while $s_t \notin \mathbb{G}$ **do**
  Get $a_t \leftarrow \texttt{SEARCH}(s_t, \hat{M}_t, V_{\theta_t}, K)$
  Execute a_t in environment M to get $s_{t+1} \leftarrow f(s_t, a_t)$
  if $d(s_{t+1}, \hat{f}(s_t, a_t)) > \xi$ **then**
    Add $\mathcal{X}_{t+1}^\xi \leftarrow \mathcal{X}_t^\xi \cup \{\mathsf{sphere}(s_t, a_t, \delta)\}$
  else
    $\mathcal{X}_{t+1}^\xi \leftarrow \mathcal{X}_t^\xi$
  Update $\hat{M}_{t+1} \leftarrow \tilde{M}_{\mathcal{X}_{t+1}^\xi}$
  Add s_t to buffer $\mathcal{D}$
  Update $V_{\theta_{t+1}} \leftarrow \texttt{UPDATE}(s_t, \hat{M}_{t+1}, V_{\theta_t}, \mathcal{D})$
  $t \leftarrow t + 1$
function $\texttt{UPDATE}(s, \hat{M}, V_\theta, \mathcal{D})$
  for $n = 1, \cdots, N$ **do**
    Sample batch of B states S_n from buffer $\mathcal{D}$ with replacement
    Call $\texttt{SEARCH}(s_i, \hat{M}, V_\theta, K)$ for each $s_i \in S_n$ to get all states on closed list s_i' and their corresponding cost-to-go updates $V(s_i')$ and construct the training set $\mathbb{X}_n = \{(s_i', V(s_i'))\}$
    Update: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(V_\theta, \mathbb{X}_n)$
  return V_θ

go estimates of not only the expanded states in the lookahead tree obtained from current state s_t, but also from previously visited states. This ensures that the function approximation used for maintaining cost-to-go estimates does not deteriorate for states that were previously visited, and potentially help in generalization.

We can provide a guarantee on the completeness of Algorithm 13 by assuming the following:

Assumption 4.4.3. *Given a penalized model $\tilde{M}_{\mathcal{X}_t^\xi}$ and the current state s_t at any timestep t during execution, there always exists at least one path from s_t to a goal state that is at least δ distance away from any state-action pair (s, a) that is known to be ξ-incorrect, i.e. $(s, a) \in \mathcal{X}_t^\xi$.*

The above assumption has two components: the first one relaxes Assumption 4.4.1 to accommodate the notion of ξ-incorrectness, and the second one states that, unlike Assumption 4.4.1, there exists a path that not only does not contain any state-action pairs that are known to be ξ-incorrect, but also that any state-action pair on the path is at least δ distance, as measured by the metric d, away from any state-action pair that is known to be ξ-incorrect. The second component makes this assumption stronger. However, it can lead to substantial speedups in the time taken to reach a goal as we can place hyperspheres of radius δ to quickly "cover" the ξ-incorrect set.

Algorithm 13 employs approximate planning by using a function approximator for cost-to-go estimates and performing batch updates to fit the approximator. This is necessary as the state space is large, and maintaining tabular cost-to-go estimates for each state is expensive in memory and would take a large number of timesteps to update them in practice. However, for ease of analysis, we will assume that we do exact updates and maintain tabular cost-to-go estimates like Algorithm 12. Then, we can show the following guarantee:

Theorem 4.4.2. *Assume Assumption 4.4.3 holds then, if $\mathcal{X}^\xi$ denotes the set of all ξ-incorrect state-action pairs, and the initial cost-to-go estimates are admissible and consistent, then using Algorithm 13 with exact updates and tabular representation for cost-to-go estimates, the robot is guaranteed to reach a goal state in at most $|\mathbb{S}|^2$ timesteps. Furthermore, if we allow for $K = |\mathbb{S}|$ expansions, then we can guarantee that the robot will reach a goal state in at most $|\mathbb{S}|(\mathcal{C}(\delta) + 1)$ timesteps, where $\mathcal{C}(\delta)$ is the covering number of the set $\mathcal{X}^\xi$.*

Proof Sketch. The proof of the first part of the theorem is very similar to the proof of Theorem 4.4.1. It is crucial to notice that under Assumption 4.4.3, we will always have a path from the current state to a goal that has no transition within a hypersphere. Thus, using RTAA* guarantees we have that using Algorithm 13 a robot is guaranteed to reach a goal state in at most $|\mathbb{S}|^2$ timesteps.

To prove the second part, we use a similar pigeon hole principle proof as Theorem 4.4.1. However, since we "cover" the incorrect set $\mathcal{X}^\xi$ with hyperspheres, the number of times we update our heuristic to the optimal cost-to-go of the corresponding penalized model is equal to the covering number $\mathcal{C}(\delta)$ of the $\mathcal{X}^\xi$, i.e. the number of radius δ spheres whose union is a superset of $\mathcal{X}^\xi$. Thus, with $K = |\mathbb{S}|$ expansions the robot is guaranteed to reach the goal in at most $|\mathbb{S}|(\mathcal{C}(\delta) + 1)$ timesteps. □

The above theorem states that, using Algorithm 13, the robot is guaranteed to reach a goal state, if the initial cost-to-go estimates are admissible and consistent. The theorem also provides a stronger guarantee that the number of timesteps to the goal has a dependence on the covering number, if we do $|\mathbb{S}|$ number of expansions at each timestep. Covering number $\mathcal{C}(\delta)$ of a set A is formally defined as the size of the set B of state-action pairs (s, a) such that $A \subseteq \bigcup_{(s,a)\in B} \mathsf{sphere}(s, a, \delta)$. Note that the covering number $\mathcal{C}(\delta)$ is typically much smaller than the size of the set $\mathcal{X}^\xi$. Although performing $|\mathbb{S}|$ expansions at each timestep is infeasible in large state spaces with real-time constraints, it is useful to note that we achieve speedup from adding hyperspheres of radius δ. Importantly, the efficiency of the Algorithm 13 degrades gracefully with decreasing δ and reduces to the bound presented in Theorem 4.4.1, if only Assumption 4.4.1 holds. Similar to the worst-case bounds presented in Theorem 4.4.1, the number of timesteps it takes for the robot to reach a goal state, in practice as shown in our experiments, has a much smaller dependence on size of state space if we start with cost-to-go estimates that are reasonably accurate for the initial model $\hat{M}$, and use cost-to-go function approximation as we do in Algorithm 13.

4.5 Experiments

We test the applicability and efficiency of our approach CMAX on a range of robotic tasks across simulation and real-world experiments.[6] In simulated experiments, we record the mean and standard error for the number of timesteps taken by the robot to reach the goal emphasizing the performance of CMAX. For physical robot experiments, we present real-time execution statistics of CMAX. The video of our physical robot experiments can be found at https://youtu.be/eQmAeWIhjO8.

[6]Code to reproduce simulated experiments can be found at https://github.com/vvanirudh/CMAX

	Accurate Model		Inaccurate Model	
	Steps	**% Success**	**Steps**	**% Success**
CMAX	63 ± 22	90%	192 ± 40	80%
Q-Learning	34 ± 5	90%	441 ± 100	45%
Model NN	62 ± 26	90%	348 ± 82	15%
Model KNN	106 ± 34	95%	533 ± 118	50%
Plan with Accurate Model	63 ± 22	90%	364 ± 53	85%

Table 4.1: Results for the simulated 4D planar pushing task. First column corresponds to the case when the environment has no obstacles, and the model is accurate. Second column corresponds to when the environment has static obstacles. and model (with no obstacles) is inaccurate. Each entry in the Steps subcolumn is obtained using 20 random start and goal locations, and we present mean and standard error of number of timesteps it takes the robot to reach the goal *among successful trials*. The % success subcolumn indicates percentage of successful trials where the robot reached the goal in less than 1000 timesteps. The last row corresponds to using the planner with an accurate model (the same as the environment.)

4.5.1 Simulated 4D Planar Pushing in the Presence of Obstacles

In this experiment, the task is for a robotic gripper to push a cube from a start location to a goal location in the presence of static obstacles without any resets, as shown in Figure 4.3 (right). This can be represented as a planning problem in 4D continuous state space $\mathbb{S}$ with any state represented as the tuple $s = (g_x, g_y, o_x, o_y)$ where (g_x, g_y) are the xy-coordinates of the gripper and (o_x, o_y) are the xy-coordinates of the object. The model $\hat{M}$ used for planning *does not* have the static obstacles and the robot can only discover the state-action pairs that are affected due to the obstacles through real world executions. The action space $\mathbb{A}$ is a discrete set of 4 actions that move the gripper end-effector in the 4 cardinal directions by a fixed offset using an IK-based controller. The cost of each transition is 1 when the object is not at the goal location, and 0 otherwise.

We compare CMAX with the following baselines: a model-free Q-learning approach [Mni+15] that learns from online executions in environment and does not use the model $\hat{M}$, and a model learning approach that uses limited-expansion search for planning but updates a learned residual that compensates for the discrepancy in dynamics between the model and environment. The model learning approach is very similar to previous works that learn residual dynamics models and have been shown to work well in episodic settings [RKK18; HY15; Sav+17]. We chose two function approximators for the learned residual dynamics to account for model learning approaches that use global function approximators such as neural networks (NN) [Jan+19], and local function approximators such as K-nearest neighbor regression (KNN) [NL08; JS07]. Finally, we compare against a limited-expansion search planner that uses an accurate model with the full knowledge about obstacles to understand the difficulty of the task. Specific details on the architecture and baseline parameters can be found in Appendix 9.2.1.

For our implementation, we follow Algorithm 13 with euclidean distance metric, $\xi = 0.01$, and $\delta = 0.02$. These values are chosen to capture the discrepancies observed in the object and gripper position when pushed into an obstacle, and the size of the obstacles. We use the same values for the model learning KNN baseline to ensure a fair comparison. The results of our experiments are presented in Table 4.1. We notice that all the approaches have almost the same performance

	Steps	% Success
Cmax	47 ± 6	100%
RTAA*	138 ± 65	30%

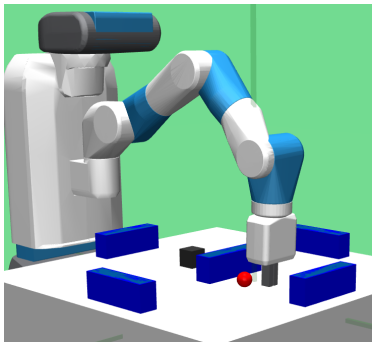

Figure 4.3: (left) Results for simulated 7D arm planning experiment comparing RTAA* and CMAX. Each entry in the Steps column is obtained using 10 trials with random start configurations and goal locations, and we present mean and standard error of number of timesteps it takes the arm to reach the goal *among successful trials*. The % success column indicates percentage of successful trials where the arm reached the goal in less than 300 timesteps.(right) 4D Planar Pushing in the presence of obstacles. The task is to push the black box to the red goal using the end-effector.

when both model and environment have no obstacles (first column). This validates that all the baselines do well when the model is accurate. However, when the model is inaccurate (second column), the performance varies across baselines. Q-learning performs decently well since it relies on the model only for the initialized Q-values and not during online executions, but as the task is now more difficult, it solves much fewer trials and is highly suboptimal. It is interesting to see that model learning baselines do not do as well as one would expect. This can be attributed to the number of online executions required to learn the correct residual, which can be prohibitively large. Among the two model learning baselines, KNN works better since it requires fewer samples to learn the residual, while NN requires large amounts of data. In contrast, CMAX does not seek to learn the true dynamics and instead is more focused on reaching the goal quickly. When compared with a planner that uses the accurate model with obstacles and solves 17 trials (last row in Table 4.1), our approach solves 16 trials and achieves the lowest mean number of timesteps to reach the goal among all baselines. We would like to note that the planner with accurate model takes a larger number of timesteps because we used the same initial cost-to-go estimates as other approaches. The initial cost-to-go estimates are more accurate for the model with no obstacles than for the model with obstacles. Hence, it spends a larger number of timesteps updating cost-to-go estimates. This experiment shows that by focusing on reaching the goal and not trying to correct the model dynamics, CMAX performs the best and solves the most number of trials among baselines.

4.5.2 3D Pick-and-Place with a Heavy Object

The task of this physical robot experiment (Figure 4.4) is to pick and place a heavy object using a PR2 arm from a start pick location to a goal place location while avoiding an obstacle. This can be represented as a planning problem in 3D discrete state space $\mathbb{S}$ where each state corresponds to the 3D location of the end-effector. Since it is a relatively small state space, we use exact planning updates without any function approximation following Algorithm 12 with $K = 3$ expansions. The action space is a discrete set of 6 actions corresponding to a fixed offset movement in positive

Figure 4.4: Physical robot 3D pick-and-place experiment. The task is to pick the object (light - wooden block, heavy - black dumbbell) and place it at the goal location (green) while avoiding the obstacle (box). For the light object (first 3 images), the model dynamics are accurate and the robot takes it on the optimal path that goes above the obstacle. For the heavy object (next 3 images), the model dynamics are inaccurate but using CMAX the robot discovers that there is a discrepancy in dynamics when the object is lifted beyond a certain height (due to joint torque limits), adds hyperspheres at that height to account for these transitions (red spheres in the last image), and quickly finds an alternate path going behind the obstacle.

or negative direction along each dimension. The model $\hat{M}$ used by planning *does not* model the object as heavy and hence, does not capture the dynamics of the arm correctly when it holds the heavy object. Specific details regarding the experiment can be found in Appendix 9.2.2.

We observe that if the object was not heavy, then the arm takes the object from the start pick location to the goal place location on the optimal path which goes above the obstacle (first 3 images of Figure 4.4). However, when executed with a heavy object, the arm cannot lift the object beyond a certain height as its joint torque limits are reached. At this point, the robot notes the discrepancy in dynamics between the model $\hat{M}$ and the real world, and inflates the cost of any executed transition that tried to move the object higher. Subsequently, the robot figures out an alternate path that does not require it to lift the object higher by taking the object behind the obstacle to the goal place location (last 4 images of Figure 4.4). The robot takes 36 timesteps (25.8 seconds) to reach the goal with the heavy object, in comparison to 26 timesteps (22.8 seconds) for the light object (see video). Thus, the robot using CMAX successfully completes the task despite having a model with inaccurate dynamics.

4.5.3 7D Arm Planning with a Non-Operational Joint

The task of this physical robot experiment (Figure 4.5) is to move the PR2 arm with a non-operational joint from a start configuration so that the end-effector reaches a goal location, specified as a 3D region. We represent this as a planning problem in 7D discrete statespace $\mathbb{S}$ where each dimension corresponds to a joint of the arm bounded by its joint limits. The action space $\mathbb{A}$ is a discrete set of size 14 corresponding to moving each joint by a fixed offset in the positive or negative direction. The model $\hat{M}$ used for planning *does not* know that a joint is non-operational and assumes that the arm can attain any configuration within the joint limits. In the real world, if the robot tries to move the non-operational joint, the arm does not move. Specific details regarding the experiment can be found in Appendix 9.2.3.

For the purpose of this experiment since the state space is very large, we follow Algorithm 13 with $\delta = 1$, $\xi = 1$, and make the shoulder lift joint (marked by red cross and arrows in last image of Figure 4.5) of PR2 non-operational. We use a kernel regressor with RBF kernel of length scale $\gamma = 10$ for the cost-to-go function approximation. Figure 4.5 shows CMAX operating in the real world to place an object at a desired location with a goal tolerance of 10 cm. When the shoulder lift joint is operational, the robot finds a path quickly to the place location by using the joint

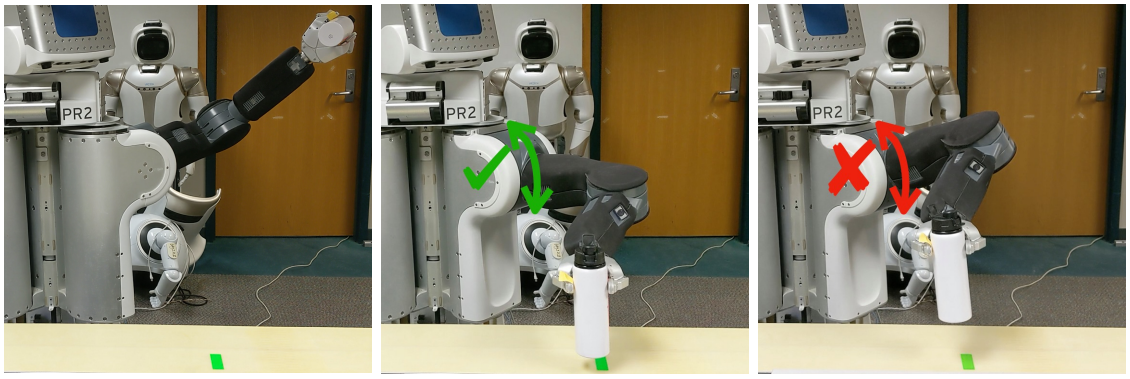

Figure 4.5: Physical robot 7D arm planning experiment. The task is to start from a fixed configuration (shown in the first image) and move the arm so that the end-effector reaches the object place location (green). When the shoulder lift joint is operational, the robot uses the joint to quickly find a path to the goal (middle image). However, when the joint is non-operational, it encounters discrepancies in its model and compensates by finding a path that uses other joints to reach the goal (last image.)

(middle image of Figure 4.5). However, when the shoulder lift joint is non-operational, the robot notes discrepancy in dynamics whenever it tries to move the joint, places hyperspheres in 7D to inflate the cost, and comes up with an alternate path (last image of Figure 4.5) to reach the place location. The robot takes 13 timesteps (32.4 seconds) to reach the goal location with the non-operational joint, in comparison to 10 timesteps (25.8 seconds) for the case where the joint is working (see video). Thus, the robot successfully finds a path to the place location despite using a model with inaccurate dynamics.

To emphasize the need for cost-to-go function approximation and local generalization from hyperspheres in large state spaces, we compared CMAX against RTAA*, an exact planning method that uses a tabular representation for cost-to-go estimates and updates model dynamics online. Results are presented in Figure 4.3 (left) and show that RTAA* fails to solve 7 of the 10 trials whereas CMAX solves all of them, and in smaller mean number of timesteps.

4.5.4 Effect of Function Approximation and Size of Hyperspheres

While previous experiments have tested CMAX against other baselines and on a physical robot, this experiment is designed to evaluate the effect of cost-to-go function approximation and the size of hyperspheres on the performance of CMAX in large state spaces (Algorithm 13.) For the first set of experiments (Figure 4.6 left), we use the setup of Section 4.5.3 and focus on varying the smoothness of the kernel regressor cost-to-go function approximation by varying the length scale γ of the RBF kernel. Intuitively, small length scales result in approximation with high variance, and for large scales we obtain highly smooth approximation. We notice that for small γ, the performance is poor and as γ increases, the performance of CMAX becomes better as it can generalize the cost-to-go estimates in the state space. However, for large γ the performance deteriorates as it fails to capture the difference in cost-to-go values among nearby states due to excessive smoothing. This showcases the need for generalization in cost-to-go estimates for efficient updates in large state spaces.

For the second set of experiments (Figure 4.6 right), we vary the radius of the hyperspheres

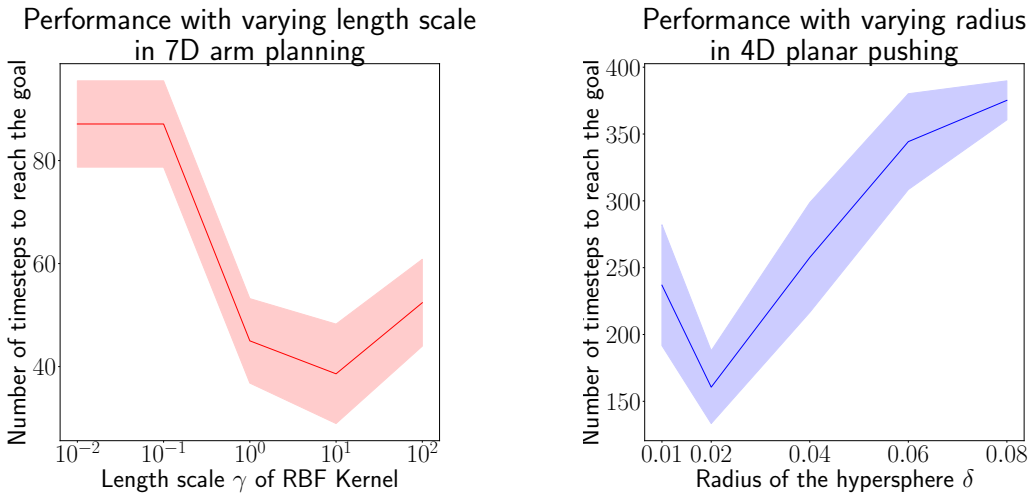

Figure 4.6: (left) Performance of CMAX for 7D arm planning as the smoothness of the cost-to-go function approximator varies. The plot is generated for each value of length scale γ by generating 10 random start configurations and goal locations, and running our approach for a maximum of 100 timesteps. (right) Performance of our approach for 4D planar pushing as the radius of the hypersphere δ varies. The plot is generated for each value of radius δ by generating 10 random start and goal locations, and running CMAX for a maximum of 400 timesteps.

% Ice →	**0%**	**40%**	**80%**
CMAX	78 ± 4	231 ± 18	2869 ± 331
RTAA*	78 ± 4	219 ± 18	2185 ± 249
Q-Learning	3914 ± 303	1220 ± 103	996 ± 108

Table 4.2: Results for gridworld navigation in presence of icy states for a grid of size 100×100. Each entry is obtained using 50 random seeds, and we present the mean and standard error of the number of timesteps it takes the robot to reach the goal. The columns represent the percentage of icy states in the gridworld.

δ introduced whenever an incorrect state-action pair is discovered in Algorithm 13. We use the setup of Section 4.5.1, vary δ and observe the number of timesteps it takes the robot to push the object to the goal. We observe that when δ is large, the performance is poor as we potentially penalize state-action pairs that are not incorrect and could result in a very suboptimal path. However, a very small δ can also lead to a poor performance, as we need more online executions to discover the set of incorrect state-action pairs. Hence, the radius δ needs to be chosen carefully to quickly "cover" the incorrect set, while not penalizing any correct state-action pairs.

4.5.5 Simulated 2D Gridworld Navigation with Icy States

In our final experiment, we want to understand the performance of CMAX compared to other baselines in small domains where model dynamics can be represented using a table, and can be updated efficiently. We consider the 2D gridworld such as the one shown in Figure 5.1(right) with icy states where the robot slips (moving left or right on ice moves the robot by two cells.) The model used for planning *does not* contain ice, and is an empty gridworld. The results are

presented in Table 4.2. We can observe that model-free approaches like Q-learning perform well compared to model-based approaches in cases where the model available is highly inaccurate (see Table 4.2 last column.) However, when the model is reasonably accurate RTAA* performs the best. But the results show that even in domains where model dynamics are simple and can be updated efficiently, CMAX competes closely with RTAA*. Thus, our approach is still applicable in such domains and is relatively easier to implement.

4.6 Related Work

The proposed approach has components concerning real-time heuristic search, local function approximation methods, and dealing with inaccuracy in models. There is a wide array of existing work at the intersection of planning and learning that deal with these topics. Notably, we leverage prior work on real-time heuristic search [Kor90; KL06] for the limited-expansion search-based planner presented in Algorithm 8. Using local function approximation methods in robotics has been heavily explored in seminal works [VS00; AS97] due to their smaller sample complexity requirements and local generalization properties that do not cause interference [AMS97a; CAN08]. More recently, [JS07], [NL08] and [BS10] have also proposed approaches that learn local models from online executions. However unlike CMAX, they use these models to approximate the dynamics of the real world. Our work is also closely related to the field of real-time reinforcement learning that tackles the problem of acting near-optimally in unknown environments, without any resets [Sut91; BBS95; KS93]. The analysis presented in Theorem 4.4.1 and 4.4.2 borrows several useful results from [KS93]. Prior works in model-based reinforcement learning with provable guarantees, such as [KS02; BS10; BT02; KKL03], are also related. However, these works learn the true dynamics by updating the model and give sample complexity results in the finite-horizon setting or discounted infinite-horizon setting, unlike our shortest path setting. Among these works, [KKL03], which proposes a method for exploration in metric state spaces, serves as an inspiration for the covering number bounds given in Theorem 4.4.2. The work that is most closely related to ours is [Jia18] which proposed an approach that uses a similar idea of updating the cost function, in cases where updating the model dynamics is infeasible. However, their approach is suitable only for episodic settings and small state spaces. Concurrent work by [McC+20] employs a binary classifier trained on offline data to predict whether a transition is incorrect or correct, that is then queried during online motion planning to construct the search tree consisting of only transitions that are classified as correct.

4.7 Discussion and Conclusion

CMAX is the first approach for interleaving planning and execution that does not require updating dynamics, and is guaranteed to reach the goal despite using an inaccurate dynamical model. The biggest advantage of CMAX is that it does not rely on any knowledge of how the model is inaccurate, and whether it can be updated in real-time. Hence, it is broadly applicable in real world robotic tasks with complex inaccurate models. In domains where modeling the true dynamics is intractable, such as deformable manipulation, CMAX can still be employed to ensure successful execution. In comparison, approaches that update the model dynamics online rely on the flexibility of the model to be updated, knowledge of what is lacking in the model, and a large number of online executions to correct it. For example, to learn accurate dynamics for a

transition in N-D statespace we need at least N samples in the worst case, whereas our approach needs only 1 sample to observe a discrepancy and inflate the cost.

The most important shortcomings of CMAX are Assumptions 4.4.1 and 4.4.3, which are hard to verify, and are not satisfied in several real world robotic tasks. For example, consider the task of opening a spring-loaded door which is not modeled as loaded. All transitions would have discrepancy in dynamics, and CMAX as is would fail at completing the task in a reasonable amount of time. In addition, the hyperparameter δ describing the radius of hypersphere needs to be tuned carefully for each domain which is a limitation of CMAX.

To summarize, we present CMAX for interleaving planning and execution using inaccurate models that does not require updating the dynamics of the model, and still provably completes the task. We propose practical algorithms for both small and large state spaces, and deploy them successfully in real world robot tasks showing its broad applicability. In simulation, we analyze CMAX and show that it outperforms baselines that update dynamics online.

Chapter 5

Leveraging Experience in Planning and Execution using Inaccurate Models

. . . many of mundane manipulation tasks such as picking and placing various objects in a kitchen are highly repetitive. It is therefore expected that robots should be capable of learning and improving their performance with every execution of these repetitive tasks.

Mike Phillips, Ben Cohen, Sachin Chitta, Max Likhachev (2012)

While CMAX, presented in Chapter 4, guaranteed task completeness using inaccurate models, it required strong assumptions on the accuracy of the model. Furthermore, in a repetitive task where the robot is required to complete the same task over several repetitions, CMAX fails to improve the solution quality and in some cases, fails to complete the task in later repetitions. This chapter presents an algorithm that avoids this behavior by leveraging the online experience to improve quality of solution across repetitions while retaining task completeness guarantee in each repetition. These guarantees require a milder assumption on the accuracy of the model which is easier to satisfy when compared to the assumption required by CMAX. However, like CMAX, the proposed algorithm does not require any updates to the dynamics of the model and only adapts the behavior of the planner in the spirit of this thesis. This chapter is adapted from our original paper [VBL20].

5.1 Introduction

We often require robots to perform tasks that are highly repetitive, such as picking and placing objects in assembly tasks and navigating between locations in a warehouse. For such tasks, robotic planning algorithms have been highly effective in cases where system dynamics is easily specified by an efficient forward model [BAG12]. However, for tasks involving interactions with objects, dynamics are very difficult to model without complete knowledge of the parameters of the objects such as mass and friction [JX01]. Using inaccurate models for planning can result in plans

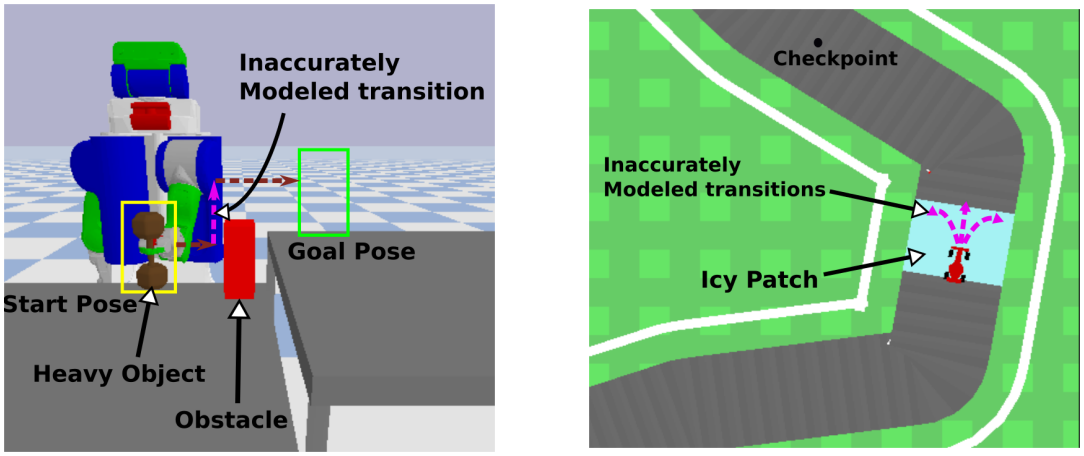

Figure 5.1: (left) PR2 lifting a heavy dumbbell, that is modeled as light, to a goal location that is higher than the start location resulting in dynamics that are inaccurately modeled (right) Mobile robot navigating around a track with icy patches with unknown friction parameters leading to the robot skidding. In both cases, any path to the goal needs to contain a transition (pink) whose dynamics are not modeled accurately.

that are ineffective and fail to complete the task [McC+20]. In addition for such repetitive tasks, we expect the robot's task performance to improve, leading to efficient plans in later repetitions. Thus, we need a planning approach that can use potentially inaccurate models while leveraging experience from past executions to complete the task in each repetition, and improve performance across repetitions.

A recent planning approach, CMAX, introduced in [Vem+20] adapts its planning strategy online to account for any inaccuracies in the forward model without requiring any updates to the dynamics of the model. CMAX achieves this online by inflating the cost of any transition that is found to be incorrectly modeled and replanning, thus biasing the resulting plans away from regions where the model is inaccurate. It does so while maintaining guarantees on completing the task, without any resets, in a finite number of executions. However, CMAX requires that there always exists a path from the current state of the robot to the goal containing only transitions that have not yet been found to be incorrectly modeled. This is a strong assumption on the accuracy of the model and can often be violated, especially in the context of repetitive tasks.

For example, consider the task shown in Figure 5.1(left) where a robotic arm needs to repeatedly pick a heavy object, that is incorrectly modeled as light, and place it on top of a taller table while avoiding an obstacle. As the object is heavy, transitions that involve lifting the object will have discrepancy between true and modeled dynamics. However, any path from the start pose to the goal pose requires lifting the object and thus, the resulting plan needs to contain a transition that is incorrectly modeled. This violates the aforementioned assumption of CMAX and it ends up inflating the cost of any transition that lifts the object, resulting in plans that avoid lifting the object in future repetitions. Thus, the quality of CMAX solution deteriorates across repetitions and, in some cases, it even fails to complete the task. Figure 5.1(right) presents another example task where a mobile robot is navigating around a track with icy patches that have unknown friction parameters. Once the robot enters a patch, any action executed results in the robot skidding, thus violating the assumption of CMAX because any path to the goal from current state will have inaccurately modeled transitions. CMAX ends up inflating the cost of all

actions executed inside the icy patch, leading to the robot being unable to find a path in future laps and failing to complete the task. Thus, in both examples, we need a planning approach that allows solutions to contain incorrectly modeled transitions while ensuring that the robot reaches the goal.

In this paper we present CMAX++, an approach for interleaving planning and execution that uses inaccurate models and leverages experience from past executions to provably complete the task in each repetition without any resets. Furthermore, it improves the quality of solution across repetitions. In contrast to CMAX, CMAX++ requires weaker conditions to ensure task completeness, and is provably guaranteed to converge to a plan with optimal cost as the number of repetitions increases. The key idea behind CMAX++ is to combine the conservative behavior of CMAX that tries to avoid incorrectly modeled regions with model-free Q-learning that tries to estimate and follow the optimal cost-to-goal value function with no regard for any discrepancies between modeled and true dynamics. This enables CMAX++ to compute plans that utilize inaccurately modeled transitions, unlike CMAX. Based on this idea, we present an algorithm for small state spaces, where we can do exact planning, and a practical algorithm for large state spaces using function approximation techniques. We also propose an adaptive version of CMAX++ that intelligently switches between CMAX and CMAX++ to combine the advantages of both approaches, and exhibits goal-driven behavior in earlier repetitions and optimality in later repetitions. The proposed algorithms are tested on simulated robotic tasks: 3D mobile robot navigation where the track friction is incorrectly modeled (Figure 5.1 right) and a 7D pick-and-place task where the mass of the object is unknown (Figure 5.1 left).

5.2 Related Work

A typical approach to planning in tasks with unknown parameters is to use acquired experience from executions to update the dynamics of the model and replan [Sut91]. This works well in practice for tasks where the forward model is flexible and can be updated efficiently. However for real world tasks, the models used for planning cannot be updated efficiently online [TET12] and are often precomputed offline using expensive procedures [Hau+06]. Another line of works [Sav+17; AQN06a] seek to learn a residual dynamical model to account for the inaccuracies in the initial model. However, it can take a prohibitively large number of executions to learn the true dynamics, especially in domains like deformable manipulation [EBG12]. This precludes these approaches from demonstrating a goal-driven behavior as we show in our experimental analysis.

Recent works such as CMAX [Vem+20] and [McC+20] pursue an alternative approach which does not require updating the dynamics of the model or learning a residual component. These approaches exhibit goal-driven behavior by focusing on completing the task and not on modeling the true dynamics accurately. While CMAX achieves this by inflating the cost of any transition whose dynamics are inaccurately modeled, [McC+20] present an approach that learns a binary classifier offline that is used online to predict whether a transition is accurately modeled or not. Although these methods work well in practice for goal-oriented tasks, they do not leverage experience acquired online to improve the quality of solution when used for repetitive tasks.

Our work is closely related to approaches that integrate model-based planning with model-free learning. [Lee+20b] use model-based planning in regions where the dynamics are accurately modeled and switch to a model-free policy in regions with high uncertainty. However, they mostly focus on perception uncertainty and require a coarse estimate of the uncertain region prior to execution, which is often not available for tasks with other modalities of uncertainty like unknown

inertial parameters. A very recent work by [LLK20] uses a model-based planner until a model inaccuracy is detected and switches to a model-free policy to complete the task. Similar to our approach, they deal with general modeling errors but rely on expert demonstrations to learn the model-free policy. In contrast, our approach does not require any expert demonstrations and only uses the experience acquired online to obtain model-free value estimates that are used within planning.

Finally, our approach is also related to the field of real-time heuristic search which tackles the problem of efficient planning in large state spaces with bounded planning time. In this work, we introduce a novel planner that is inspired by LRTA* [Kor90] which limits the number of expansions in the search procedure and interleaves execution with planning. Crucially, our planner also interleaves planning and execution but unlike these approaches, employs model-free value estimates obtained from past experience within the search.

5.3 Problem Setup

Following the notation of [Vem+20], we consider the deterministic shortest path problem that can be represented using the tuple $M = (\mathbb{S}, \mathbb{A}, \mathbb{G}, f, c)$ where $\mathbb{S}$ is the state space, $\mathbb{A}$ is the action space, $\mathbb{G} \subseteq \mathbb{S}$ is the non-empty set of goals, $f : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ is a deterministic dynamics function, and $c : \mathbb{S} \times \mathbb{A} \to [0, 1]$ is the cost function. Note that we assume that the costs lie between 0 and 1 but any bounded cost function can be scaled to satisfy this assumption. Crucially, our approach assumes that the action space $\mathbb{A}$ is discrete, and any goal state $g \in \mathbb{G}$ is a cost-free termination state. The objective of the shortest path problem is to find the least-cost path from a given start state $s_1 \in \mathbb{S}$ to any goal state $g \in \mathbb{G}$ in M. As is typical in shortest path problems, we assume that there exists at least one path from each state $s \in \mathbb{S}$ to one of the goal states, and that the cost of any transition from a non-goal state is positive [Ber05]. We will use $V(s)$ to denote the state value function (a running estimate of cost-to-goal from state s,) and $Q(s, a)$ to denote the state-action value function (a running estimate of the sum of transition cost and cost-to-goal from successor state,) for any state s and action a. Similarly, we will use the notation $V^*(s)$ and $Q^*(s, a)$ to denote the corresponding optimal value functions. A value estimate is called admissible if it underestimates the optimal value function at all states and actions, and is called consistent if it satisfies the triangle inequality, i.e. $V(s) \leq c(s, a) + V(f(s, a))$ and $Q(s, a) \leq c(s, a) + V(f(s, a))$ for all s, a, and $V(g) = 0$ for all $g \in \mathbb{G}$.

In this work, we focus on repetitive robotic tasks where the true deterministic dynamics f are unknown but we have access to an approximate model described using $\hat{M} = (\mathbb{S}, \mathbb{A}, \mathbb{G}, \hat{f}, c)$ where $\hat{f}$ approximates the true dynamics. In each repetition of the task, the robot acts in the environment M to acquire experience over a single trajectory and reach the goal, without access to any resets. This rules out any episodic approach. Since the true dynamics are unknown and can only be discovered through executions, we consider the online real-time planning setting where the robot has to interleave planning and execution. In our motivating navigation example (Figure 5.1 right,) the approximate model $\hat{M}$ represents a track with no icy patches whereas the environment M contains icy patches. Thus, there is a discrepancy between the modeled dynamics $\hat{f}$ and true dynamics f. Following [Vem+20], we will refer to state-action pairs that have inaccurately modeled dynamics as "incorrect" transitions, and use the notation $\mathcal{X} \subseteq \mathbb{S} \times \mathbb{A}$ to denote the set of discovered incorrect transitions. The objective in our work is for the robot to reach a goal in each repetition, despite using an inaccurate model for planning while improving performance, measured using the cost of executions, across repetitions.

Algorithm 11 Hybrid Limited-Expansion Search

1: **procedure** SEARCH($s, \hat{M}, V, Q, \mathcal{X}, K$)
2: Initialize $g(s) = 0$, min-priority open list O, and closed list C
3: Add s to open list O with priority $p(s) = g(s) + V(s)$
4: **for** $i = 1, 2, \cdots, K$ **do**
5: Pop s_i from O
6: **if** s_i is a dummy state or $s_i \in \mathbb{G}$ **then**
7: Set $s_{\mathsf{best}} \leftarrow s_i$ and go to Line 22
8: **for** $a \in \mathbb{A}$ **do** ▷ *Expanding state* s_i
9: **if** $(s_i, a) \in \mathcal{X}$ **then** ▷ *Incorrect transition*
10: Add a dummy state s' to O with priority $p(s') = g(s_i) + Q(s_i, a)$
11: **continue**
12: Get successor $s' = \hat{f}(s_i, a)$
13: If $s' \in C$, **continue**
14: **if** $s' \in O$ and $g(s') > g(s_i) + c(s_i, a)$ **then**
15: Set $g(s') = g(s_i) + c(s_i, a)$ and recompute $p(s')$
16: Reorder open list O
17: **else if** $s' \notin O$ **then**
18: Set $g(s') = g(s_i) + c(s_i, a)$
19: Add s' to O with priority $p(s') = g(s') + V(s')$
20: Add s_i to closed list C
21: Pop s_{best} from open list O
22: **for** $s' \in C$ **do**
23: Update $V(s') \leftarrow p(s_{\mathsf{best}}) - g(s')$
24: Backtrack from s_{best} to s, and set a_{best} as the first action on path from s to s_{best} in the search tree
return a_{best}

5.4 Approach

In this section, we will describe the proposed approach CMAX++. First, we will present a novel planner used in CMAX++ that can exploit incorrect transitions using their model-free Q-value estimates. Second, we present CMAX++ and its adaptive version for small state spaces, and establish their guarantees. Finally, we describe a practical instantiation of CMAX++ for large state spaces leveraging function approximation techniques.

5.4.1 Hybrid Limited-Expansion Search Planner

During online execution, we want the robot to acquire experience and leverage it to compute better plans. This requires a hybrid planner that is able to incorporate value estimates obtained using past experience in addition to model-based planning, and quickly compute the next action to execute. To achieve this, we propose a real-time heuristic search-based planner that performs a bounded number of expansions and is able to utilize Q-value estimates for incorrect transitions.

The planner is presented in Algorithm 11. Given the current state s, the planner constructs a lookahead search tree using at most K state expansions. For each expanded state s_i, if any

Algorithm 12 CMAX++ and A-CMAX++ in small state spaces

Require: Model $\hat{M}$, start state s, initial value estimates V, Q, number of expansions K, $t \leftarrow 1$, incorrect set $\mathcal{X} \leftarrow \{\}$, Number of repetitions N, Sequence $\{\alpha_i \geq 1\}_{i=1}^{N}$, initial penalized value estimates $\tilde{V} = V$, penalized model $\tilde{M} \leftarrow \hat{M}$

1: **for** each repetition $i = 1, \cdots, N$ **do**
2: $\quad t \leftarrow 1$, $s_1 \leftarrow s$
3: $\quad$ **while** $s_t \notin \mathbb{G}$ **do**
4: $\quad\quad$ Compute $a_t = \texttt{SEARCH}(s_t, \hat{M}, V, Q, \mathcal{X}, K)$
5: $\quad\quad$ Compute $\tilde{a}_t = \texttt{SEARCH}(s_t, \tilde{M}, \tilde{V}, Q, \{\}, K)$
6: $\quad\quad$ If $\tilde{V}(s_t) \leq \alpha_i V(s_t)$, assign $a_t = \tilde{a}_t$
7: $\quad\quad$ Execute a_t in environment to get $s_{t+1} = f(s_t, a_t)$
8: $\quad\quad$ **if** $s_{t+1} \neq \hat{f}(s_t, a_t)$ **then**
9: $\quad\quad\quad$ Add (s_t, a_t) to the set: $\mathcal{X} \leftarrow \mathcal{X} \cup \{(s_t, a_t)\}$
10: $\quad\quad\quad$ Update: $Q(s_t, a_t) = c(s_t, a_t) + V(s_{t+1})$
11: $\quad\quad\quad$ Update penalized model $\tilde{M} \leftarrow \tilde{M}_{\mathcal{X}}$
12: $\quad\quad$ $t \leftarrow t + 1$

outgoing transition has been flagged as incorrect based on experience, i.e. $(s_i, a) \in \mathcal{X}$, then the planner creates a dummy state with priority computed using the model-free Q-value estimate of that transition (Line 10). Note that we create a dummy state because the model $\hat{M}$ does not know the true successor of an incorrect transition. For the transitions that are correct, we obtain successor states using the approximate model $\hat{M}$. This ensures that we rely on the inaccurate model only for transitions that are not known to be incorrect. At any stage, if a dummy state is expanded then we need to terminate the search as the model $\hat{M}$ does not know any of its successors, in which case we set the best state s_{best} as the dummy state (Line 7). Otherwise, we choose s_{best} as the best state (lowest priority) among the leaves of the search tree after K expansions (Line 21). Finally, the best action to execute at the current state s is computed as the first action along the path from s to s_{best} in the search tree (Line 24). The planner also updates state value estimates V of all expanded states using the priority of the best state $p(s_{\mathsf{best}})$ to make the estimates more accurate (Lines 22 and 23) similar to RTAA* [KL06].

The ability of our planner to exploit incorrect transitions using their model-free Q-value estimates, obtained from past experience, distinguishes it from real-time search-based planners such as LRTA* [Kor90] which cannot utilize model-free value estimates during planning. This enables CMAX++ to result in plans that utilize incorrect transitions if they enable the robot to get to the goal with lower cost.

5.4.2 CMAX++ in Small State Spaces

CMAX++ in small state spaces is simple and easy-to-implement as it is feasible to maintain value estimates in a table for all states and actions and to explicitly maintain a running set of incorrect transitions with fast lookup without resorting to function approximation techniques.

The algorithm is presented in Algorithm 12 (only the text in black.) CMAX++ maintains a running estimate of the set of incorrect transitions $\mathcal{X}$, and updates the set whenever it encounters an incorrect state-action pair during execution. Crucially, unlike CMAX, it maintains a Q-value estimate for the incorrect transition that is used during planning in Algorithm 11, thereby enabling the planner to compute paths that contain incorrect transitions. It is also important to note that,

like CMAX, CMAX++ never updates the dynamics of the model. However, instead of using the penalized model for planning as CMAX does, CMAX++ uses the initial model $\hat{M}$, and utilizes both model-based planning and model-free Q-value estimates to replan a path from the current state to a goal.

The downside of CMAX++ is that estimating Q-values from online executions can be inefficient as it might take many executions before we obtain an accurate Q-value estimate for an incorrect transition. This has been extensively studied in the past and is a major disadvantage of model-free methods [Sun+19a]. As a result of this inefficiency, CMAX++ lacks the goal-driven behavior of CMAX in early repetitions of the task, despite achieving optimal behavior in later repetitions. In the next section, we present an adaptive version of CMAX++ (A-CMAX++) that combines the goal-driven behavior of CMAX with the optimality of CMAX++.

5.4.3 Adaptive Version of CMAX++

Background on CMAX

Before we describe A-CMAX++, we will start by summarizing CMAX. For more details, refer to [Vem+20]. At each time step t during execution, CMAX maintains a running estimate of the incorrect set $\mathcal{X}$, and constructs a penalized model specified by the tuple $\tilde{M}_{\mathcal{X}} = (\mathbb{S}, \mathbb{A}, \mathbb{G}, \hat{f}, \tilde{c}_{\mathcal{X}})$ where the cost function $\tilde{c}_{\mathcal{X}}(s, a) = |\mathbb{S}|$ if $(s, a) \in \mathcal{X}$, else $\tilde{c}_{\mathcal{X}}(s, a) = c(s, a)$. In other words, the cost of any transition found to be incorrect is set high (or inflated) while the cost of other transitions are the same as in $\hat{M}$. CMAX uses the penalized model $\tilde{M}_{\mathcal{X}}$ to plan a path from the current state s_t to a goal state. Subsequently, CMAX executes the first action a_t along the path and observes if the true dynamics and model dynamics differ on the executed action. If so, the state-action pair (s_t, a_t) is appended to the incorrect set $\mathcal{X}$ and the penalized model $\tilde{M}_{\mathcal{X}}$ is updated. CMAX continues to do this at every timestep until the robot reaches a goal state.

Observe that the inflation of cost for any incorrect state-action pair biases the planner to "explore" all other state-action pairs that are not yet known to be incorrect before it plans a path using an incorrect transition. This induces a goal-driven behavior in the computed plan that enables CMAX to quickly find an alternative path and not waste executions learning the true dynamics

A-CMAX++

A-CMAX++ is presented in Algorithm 12 (black and blue text.) A-CMAX++ maintains a running estimate of incorrect set $\mathcal{X}$ and constructs the penalized model $\tilde{M}$ at each time step t, similar to CMAX. For any state at time step t, we first compute the best action a_t based on the approximate model $\hat{M}$ and the model-free Q-value estimates (Line 4.) In addition, we also compute the best action $\tilde{a}_t$ using the penalized model $\tilde{M}$, similar to CMAX, that inflates the cost of any incorrect transition (Line 5.) The crucial step in A-CMAX++ is Line 6 where we compare the penalized value $\tilde{V}(s_t)$ (obtained using penalized model $\tilde{M}$) and the non-penalized value $V(s_t)$ (obtained using approximate model $\hat{M}$ and Q-value estimates.) Given a sequence $\{\alpha_i \geq 1\}$ for repetitions $i = 1, \cdots, N$ of the task, if $\tilde{V}(s_t) \leq \alpha_i V(s_t)$, then we execute action $\tilde{a}_t$, else we execute a_t. This implies that if the cost incurred by following CMAX actions in the future is within α_i times the cost incurred by following CMAX++ actions, then we prefer to execute CMAX.

If the sequence $\{\alpha_i\}$ is chosen to be non-increasing such that $\alpha_1 \geq \alpha_2 \cdots \geq \alpha_N \geq 1$, then we can observe that A-CMAX++ has the desired anytime-like behavior. It remains goal-driven in

early repetitions, by choosing CMAX actions, and converges to optimal behavior in later repetitions, by choosing CMAX++ actions. Further, the executions needed to obtain accurate Q-value estimates is distributed across repetitions ensuring that A-CMAX++ does not have poor performance in any single repetition. Thus, A-CMAX++ combines the advantages of both CMAX and CMAX++.

5.4.4 Theoretical Guarantees

We will start with formally stating the assumption needed by CMAX to ensure completeness:

Assumption 5.4.1 ([Vem+20]). *Given a penalized model $\tilde{M}_{\mathcal{X}_t}$ and the current state s_t at any time step t, there always exists at least one path from s_t to a goal that does not contain any state-action pairs (s, a) that are known to be incorrect, i.e. $(s, a) \in \mathcal{X}_t$.*

Observe that the above assumption needs to be valid at every time step t before the robot reaches a goal and thus, can be hard to satisfy. Before we state the theoretical guarantees for CMAX++, we need the following assumption on the approximate model $\hat{M}$ that is used for planning:

Assumption 5.4.2. *The optimal value function $\hat{V}^*$ using the dynamics of approximate model $\hat{M}$ underestimates the optimal value function V^* using the true dynamics of M at all states, i.e. $\hat{V}^*(s) \leq V^*(s)$ for all $s \in \mathbb{S}$.*

In other words, if there exists a path from any state s to a goal state in the environment M, then there exists a path with the same or lower cost from s to a goal in the approximate model $\hat{M}$. In our motivating example of pick-and-place (Figure 5.1 left,) this assumption is satisfied if the object is modeled as light in $\hat{M}$, as the object being heavy in reality can only increase the cost. This assumption was also considered in previous works such as [Jia18] and is known as the *Optimistic Model Assumption.*

Significance of Optimistic Model Assumption

To understand the significance of the optimistic model assumption, it is important to note that completeness guarantees usually require the use of admissible and consistent value estimates, i.e. estimates that always underestimate the true cost-to-goal values. This requirement needs to hold every time we plan (or replan) to ensure that we never discard a path as being expensive in terms of cost, when it is cheap in reality.

All our guarantees assume that the initial value estimates are consistent and admissible, but to ensure that they always remain consistent and admissible throughout execution, we need the optimistic model assumption. This assumption ensures that updating value estimates by planning in the model $\hat{M}$ always results in estimates that are admissible and consistent. In other words, the optimal value function (which we obtain by doing full state space planning in $\hat{M}$) of the model $\hat{M}$ always underestimates the optimal value function of the environment M at all states $s \in \mathbb{S}$.

A very intuitive way to understand the assumption is to imagine a navigation task where the robot is navigating from a start to goal in the presence of obstacles. In this example, the optimistic model assumption requires that the model should never place an obstacle in a location when there isn't an obstacle in the environment at the same location. However, if there truly is an obstacle at some location, then the model can either have an obstacle or not have one at the same location. Put simply, an agent that is planning using the model should never be "pleasantly surprised" by what it sees in the environment. This example is popularly known as the free space assumption [Zel92] in robot navigation literature. Several other toy examples of the optimistic

model assumption are presented in [Jia18] and we recommend the reader to look at them for more intuition.

We can now state the following guarantees:

Theorem 5.4.1 (Completeness)**.** *Assume the initial value estimates V, Q are admissible and consistent. Then we have,*

1. *If Assumption 5.4.2 holds then using either* CMAX++ *or* A-CMAX++*, the robot is guaranteed to reach a goal state in at most $|\mathbb{S}|^3$ time steps in each repetition.*
2. *If Assumption 5.4.1 holds then (a) using* A-CMAX++ *with a large enough α_i in any repetition i (typically true for early repetitions,) the robot is guaranteed to reach a goal state in at most $|\mathbb{S}|^2$ time steps, and (b) using* CMAX++*, it is guaranteed to reach a goal state in at most $|\mathbb{S}|^3$ time steps in each repetition*

Proof Sketch. To prove completeness, first we need to note that the Q-update in CMAX++ always ensures that the Q-value estimates remain consistent and admissible as long as the state value estimates remain consistent and admissible. We have already seen why the optimistic model assumption ensures that the state value estimates always remain consistent and admissible. Thus, we can use Theorem 3 from RTAA* [KL06] in conjunction with the optimistic model assumption to ensure completeness. Note that if the model is inaccurate everywhere, then our planner reduces to doing $K = 1$ expansions at every time step and acts similar to Q-learning, which is also guaranteed to be complete with admissible and consistent estimates [KS93]. The worst case bound of $|\mathbb{S}|^3$ steps is taken directly from the upper bound on Q-learning from [KS93]. The above arguments are true for both CMAX++ and A-CMAX++. Note that for A-CMAX++ if all paths to the goal contains an incorrect transition then the penalized value estimate $\tilde{V}(s) > \alpha V(s)$ for any finite α and thus, will fall back on CMAX++.

For the second part of the theorem, the assumption of CMAX (we will refer this as *optimistic penalized model assumption*) in conjunction with RTAA* guarantee again ensures completeness for CMAX++ and A-CMAX++. To see this, observe that the optimistic penalized model assumption ensures that the value estimates are always admissible and consistent w.r.t the true penalized model ($\tilde{M}_{\mathcal{X}}$ where $\mathcal{X}$ contains all the incorrect transitions) and from the assumption, we know that there exists a path to the goal in the true penalized model. Hence, CMAX++ and A-CMAX++ are bound to find this path.

CMAX++ again utilizes the worst case bounds of Q-learning under the optimistic penalized assumption as well and attains an upper bound of $|\mathbb{S}|^3$ steps. However, A-CMAX++ with a sufficiently large α_i for any repetition i acts similar to CMAX, and thereby can utilize the worst case bounds of LRTA* (which is simply RTAA* with $K = 1$ expansions) from [KS93] giving an upper bound of $|\mathbb{S}|^2$ time steps. This shows the advantage of A-CMAX++ over CMAX++, especially in earlier repetitions when the incorrect set $\mathcal{X}$ is small (thus, making the optimistic penalized model assumption hold,) and α_i is large. □

Theorem 5.4.2 (Asymptotic Convergence)**.** *Assume Assumption 5.4.2 holds, and that the initial value estimates V, Q are admissible and consistent. For sufficiently large number of repetitions N, there exists an integer $j \leq N$ such that the robot follows a path with the optimal cost to the goal using* CMAX++ *in Algorithm 12 in repetitions $i \geq j$.*

Proof Sketch. The asymptotic convergence proof completely relies on the asymptotic convergence of Q-learning [KS93] and asymptotic convergence of LRTA* [Kor90] to optimal value estimates. The proof again crucially relies on the fact that the value estimates always remain admissible

Algorithm 13 CMAX++ in large state spaces

Require: Model $\hat{M}$, start state s, value function approximators V_θ, Q_ζ, number of expansions K, $t \leftarrow 1$, Discrepancy threshold ξ, Radius of hypersphere δ, Set of hyperspheres $\mathcal{X}^\xi \leftarrow \{\}$, Number of repetitions N, Batch size B, State buffer $\mathcal{D}_S$, Transition buffer $\mathcal{D}_{SA}$, Learning rate η, Number of updates U

for each repetition $i = 1, \cdots, N$ **do**
  $t \leftarrow 1$, $s_1 \leftarrow s$
  while $s_t \notin \mathbb{G}$ **do**
    Compute $a_t = \texttt{SEARCH}(s_t, \hat{M}, V_\theta, Q_\zeta, \mathcal{X}^\xi, K)$
    Execute a_t in environment to get $s_{t+1} = f(s_t, a_t)$
    if $d(s_{t+1}, \hat{f}(s_t, a_t)) > \xi$ **then**
      Add hypersphere: $\mathcal{X}^\xi \leftarrow \mathcal{X}^\xi \cup \{\mathsf{sphere}(s_t, a_t, \delta)\}$
    Add s_t to $\mathcal{D}_S$, and (s_t, a_t, s_{t+1}) to $\mathcal{D}_{SA}$
    for $u = 1, \cdots, U$ **do** ▷ *Approximator updates*
      $\texttt{Q\_UPDATE}(Q_\zeta, V_\theta, \mathcal{D}_{SA})$
      $\texttt{V\_UPDATE}(V_\theta, Q_\zeta, \mathcal{D}_S, \mathcal{X}^\xi)$
    $t \leftarrow t + 1$
procedure $\texttt{Q\_UPDATE}(Q_\zeta, V_\theta, \mathcal{D}_{SA})$
  Sample B transitions from $\mathcal{D}_{SA}$ with replacement
  Construct training set $\mathbb{X}_Q = \{((s_i, a_i), Q(s_i, a_i))\}$ for each sampled transition (s_i, a_i, s_i') and compute $Q(s_i, a_i) = c(s_i, a_i) + V_\theta(s_i')$
  Update: $\zeta \leftarrow \zeta - \eta \nabla_\zeta \mathcal{L}_Q(Q_\zeta, \mathbb{X}_Q)$
procedure $\texttt{V\_UPDATE}(V_\theta, Q_\zeta, \mathcal{D}_S, \mathcal{X}^\xi)$
  Sample B states from $\mathcal{D}_S$ with replacement
  Call $\texttt{SEARCH}(s_i, \hat{M}, V_\theta, Q_\zeta, \mathcal{X}^\xi, K)$ for each sampled s_i to get all states on closed list s_i' and their corresponding value updates $V(s_i')$ to construct training set $\mathbb{X}_V = \{(s_i', V(s_i')\}$
  Update: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_V(V_\theta, \mathbb{X}_V)$

and consistent, which is ensured by the optimistic model assumption. Note that the optimistic penalized model assumption is not enough to guarantee asymptotic convergence to the optimal cost in M as we penalize incorrect transitions. However, it is possible to show that under the optimistic penalized model assumption both CMAX++ and A-CMAX++ converge to the optimal cost in the true penalized model $\tilde{M}_{\mathcal{X}}$ where $\mathcal{X}$ contains all incorrect transitions. □

It is important to note that the conditions required for Theorem 5.4.1 are weaker than the conditions required for completeness of CMAX. Firstly, if either Assumption 5.4.1 or Assumption 5.4.2 holds then CMAX++ can be shown to be complete, but CMAX is guaranteed to be complete only under Assumption 5.4.1. Furthermore, Assumption 5.4.2 only needs to hold for the approximate model $\hat{M}$ we start with, whereas Assumption 5.4.1 needs to be satisfied for every penalized model $\tilde{M}$ constructed at any time step t during execution.

5.4.5 Large State Spaces

In this section, we present a practical instantiation of CMAX++ for large state spaces where it is infeasible to maintain tabular value estimates and the incorrect set $\mathcal{X}$ explicitly. Thus, we

leverage function approximation techniques to maintain these estimates. Assume that there exists a metric d under which $\mathbb{S}$ is bounded. We relax the definition of incorrect set using this metric to define $\mathcal{X}^{\xi}$ as the set of all (s, a) pairs such that $d(f(s, a), \hat{f}(s, a)) > \xi$ where $\xi \geq 0$. Typically, we chose ξ to allow for small modeling discrepancies that can be compensated by a low-level path following controller.

CMAX++ in large state spaces is presented in Algorithm 13. The algorithm closely follows CMAX for large state spaces presented in [Vem+20]. The incorrect set $\mathcal{X}^{\xi}$ is maintained using sets of hyperspheres with each set corresponding to a discrete action. Whenever the agent executes an incorrect state-action (s, a), CMAX++ adds a hypersphere centered at s with radius δ, as measured using metric d, to the incorrect set corresponding to action a. In future planning, any state-action pair (s', a') is declared incorrect if s' lies inside any of the hyperspheres in the incorrect set corresponding to action a'. After each execution, CMAX++ proceeds to update the value function approximators (Line 9) by sampling previously executed transitions and visited states from buffers and performing gradient descent steps (Procedures 13 and 17) using mean squared loss functions given by $\mathcal{L}_Q(Q_\zeta, \mathbb{X}_Q) = \frac{1}{2|\mathbb{X}_Q|} \sum_{(s_i,a_i)\in\mathbb{X}_Q} (Q(s_i, a_i) - Q_\zeta(s_i, a_i))^2$ and $\mathcal{L}_V(V_\theta, \mathbb{X}_V) = \frac{1}{2|\mathbb{X}_V|} \sum_{s_i\in\mathbb{X}_V} (V(s_i) - V_\theta(s_i))^2$.

By using hyperspheres, CMAX++ "covers" the set of incorrect transitions, and enables fast lookup using KD-Trees in the state space. Like Algorithm 12, we never update the approximate model $\hat{M}$ used for planning. However, unlike Algorithm 12, we update the value estimates for sampled previous transitions and states (Lines 14 and 18). This ensures that the global function approximations used to maintain value estimates V_θ, Q_ζ have good generalization beyond the current state and action. Algorithm 13 can also be extended in a similar fashion as Algorithm 12 to include A-CMAX++ by maintaining a penalized value function approximation and updating it using gradient descent.

5.5 Experiments

We test the efficiency of CMAX++ and A-CMAX++ on simulated robotic tasks emphasizing their performance in each repetition of the task, and improvement across repetitions[1]. In each task, we start the next repetition only if the robot reached a goal in previous repetition.

5.5.1 3D Mobile Robot Navigation with Icy Patches

In this experiment, the task is for a mobile robot with Reed-Shepp dynamics [RS90] to navigate around a track M with icy patches (Figure 5.1 right.) This can be represented as a planning problem in 3D discrete state space $\mathbb{S}$ with any state represented using the tuple (x, y, θ) where (x, y) is the 2D position of the robot and θ describes its heading. The XY-space is discretized into 100×100 grid and the θ dimension is discretized into 16 cells. We construct a lattice graph [PKK09] using 66 motion primitives that are pre-computed offline respecting the differential constraints on the motion of the robot. The model $\hat{M}$ used for planning contains the same track as M but without any icy patches, thus the robot discovers transitions affected by icy patches only through executions.

Since the state space is small, we use Algorithm 12 for CMAX++ and A-CMAX++. For A-CMAX++, we use a non-increasing sequence with $\alpha_i = 1 + \beta_i$ where $\beta_1 = 100$ and β_i is decreased by 2.5 after every 5 repetitions (See Appendix 9.3.1 for more details on choosing the

[1]The code to reproduce our experiments can be found at https://github.com/vvanirudh/CMAXPP.

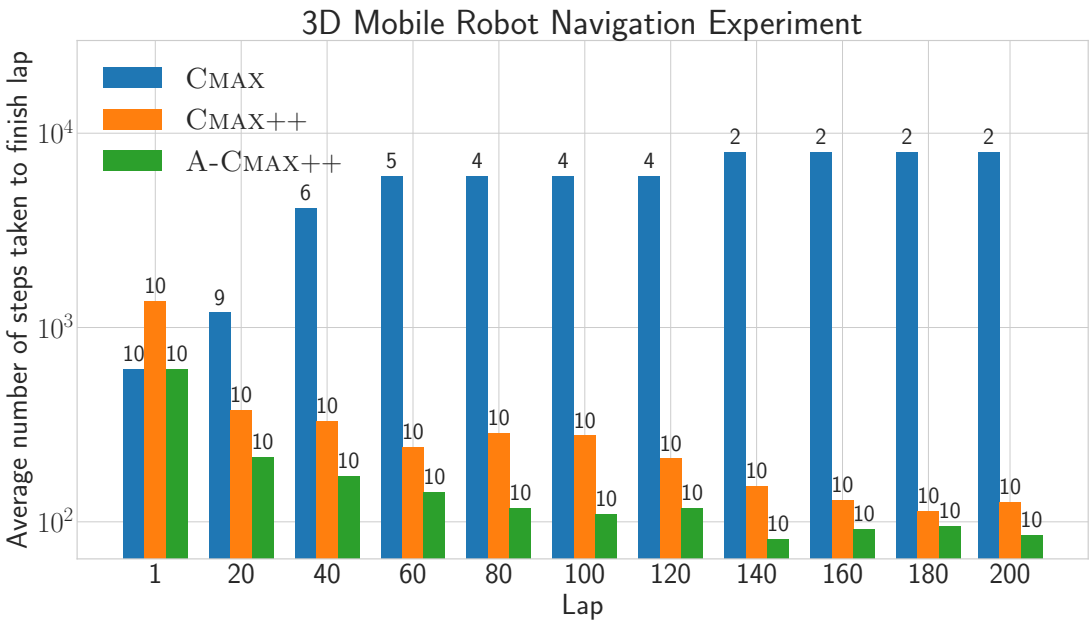

Figure 5.2: Number of steps taken to finish a lap averaged across 10 instances each with 5 icy patches placed randomly around the track. The number above each bar reports the number of instances in which the robot was successful in finishing the respective lap within 10000 time steps.

sequence.) We compare both algorithms with CMAX. For all the approaches, we perform $K = 100$ expansions. Since the motion primitives are computed offline using an expensive procedure, it is not feasible to update the dynamics of model $\hat{M}$ online and hence, we do not compare with any model learning baselines. We also conducted several experiments with model-free Q-learning, and found that it performed poorly requiring a very large number of executions and finishing only 10 laps in the best case. Hence, we do not include it in our results shown in Figure 5.2.

CMAX performs well in the early laps computing paths with lower costs compared to CMAX++. However, after a few laps the robot using CMAX gets stuck within an icy patch and does not make any more progress. Observe that when the robot is inside the icy patch, Assumption 5.4.1 is violated and CMAX ends up inflating all transitions that take the robot out of the patch leading to the robot finishing 200 laps in 2 out of 10 instances. CMAX++, on the other hand, is suboptimal in the initial laps, but converges to paths with lower costs in later laps. More importantly, the robot using CMAX++ manages to finish 200 laps in all 10 instances. A-CMAX++ also successfully finishes 200 laps in all 10 instances. However, it outperforms both CMAX and CMAX++ in all laps by intelligently switching between them achieving goal-driven behavior in early laps and optimal behavior in later laps. Thus, A-CMAX++ combines the advantages of CMAX and CMAX++.

5.5.2 7D Pick-and-Place with a Heavy Object

The task in this experiment is to pick and place a heavy object from a shorter table, using a 7 degree-of-freedom (DOF) robotic arm (Figure 5.1 left) to a goal pose on a taller table, while avoiding an obstacle. As the object is heavy, the arm cannot generate the required force in certain configurations and can only lift the object to small heights. The problem is represented

Repetition→	1		5		10		15		20	
	Steps	*Success*	*Steps*	*Success*	*Steps*	*Success*	*Steps*	*Success*	*Steps*	*Success*
CMAX	$\mathbf{17.8 \pm 3.4}$	100%	13.6 ± 0.5	60%	18 ± 0	20%	15 ± 0	20%	15 ± 0	20%
CMAX++	$\mathbf{17 \pm 4.9}$	100%	14.2 ± 3.3	100%	$\mathbf{10.6 \pm 0.3}$	100%	$\mathbf{11 \pm 0}$	100%	$\mathbf{10.8 \pm 0.1}$	100%
A-CMAX++	$\mathbf{17.8 \pm 3.4}$	100%	$\mathbf{11.6 \pm 0.7}$	100%	17 ± 6	100%	$\mathbf{10.4 \pm 0.3}$	100%	$\mathbf{10.6 \pm 0.4}$	100%
Model KNN	40.6 ± 7.3	100%	12.8 ± 1.3	100%	29.6 ± 16.1	100%	15.8 ± 2.9	100%	12.4 ± 1.4	100%
Model NN	56 ± 16.2	100%	208.2 ± 92.1	80%	124.5 ± 81.6	40%	28 ± 7.7	40%	37.5 ± 20.1	40%
Q-learning	172.4 ± 75	100%	23.2 ± 10.3	80%	26.5 ± 6.7	80%	18 ± 2.8	80%	10.2 ± 0.6	80%

Table 5.1: Number of steps taken to reach the goal in 7D pick-and-place experiment for 5 instances, each with random start and obstacle locations. We report mean and standard error *only among* successful instances in which the robot reached the goal within 500 timesteps. The success subcolumn indicates percentage of successful instances.

as planning in 7D discrete statespace where the first 6 dimensions describe the 6 DOF pose of the arm end-effector, and the last dimension corresponds to the redundant DOF in the arm. The action space $\mathbb{A}$ is a discrete set of 14 actions corresponding to moving in each dimension by a fixed offset in the positive or negative direction. The model $\hat{M}$ used for planning models the object as light, and hence does not capture the dynamics of the arm correctly when it tries to lift the heavy object. The state space is discretized into 10 cells in each dimension resulting in a total of 10^7 states. Thus, we need to use Algorithm 13 for CMAX++ and A-CMAX++. The goal is to pick and place the object for 20 repetitions where at the start of each repetition the object is in the start pose and needs to reach the goal pose by the end of repetition.

We compare with CMAX for large state spaces, model-free Q-learning [HGS16], and residual model learning baselines [Sav+17]. We chose two kinds of function approximators for the learned residual dynamics: global function approximators such as Neural Networks (NN) and local memory-based function approximators such as K-Nearest Neighbors regression (KNN.) Q-learning baseline uses Q-values that are cleverly initialized using the model $\hat{M}$ making it a strong model-free baseline. We use the same neural network function approximators for maintaining value estimates for all approaches and perform $K = 5$ expansions. We chose the metric d as the manhattan metric and use $\xi = 0$ for this experiment. We use a radius of $\delta = 3$ for the hyperspheres introduced in the 7D discrete state space, and to ensure fair comparison use the same radius for KNN regression. These values are chosen to reflect the discrepancies observed when the arm tries to lift the object. All approaches use the same initial value estimates obtained through planning in $\hat{M}$. A-CMAX++ uses a non-increasing sequence $\alpha_i = 1 + \beta_i$ where $\beta_1 = 4$ and $\beta_{i+1} = 0.5\beta_i$.

The results are presented in Table 5.1. Model-free Q-learning takes a large number of executions in the initial repetitions to estimate accurate Q-value estimates but in later repetitions computes paths with lower costs managing to finish all repetitions in 4 out of 5 instances. Among the residual model learning baselines, the KNN approximator is successful in all instances but takes a large number of executions to learn the true dynamics, while the NN approximator finishes all repetitions in only 2 instances. CMAX performs well in the initial repetitions but quickly gets stuck due to inflated costs and manages to complete the task for 20 repetitions in only 1 instance. CMAX++ is successful in finishing the task in all instances and repetitions, while improving performance across repetitions. Finally as expected, A-CMAX++ also finishes all repetitions, sometimes even having better performance than CMAX and CMAX++.

5.6 Discussion

A major advantage of CMAX++ is that, unlike previous approaches that deal with inaccurate models, it can exploit inaccurately modeled transitions without wasting online executions to learn the true dynamics. It estimates the Q-value of incorrect transitions leveraging past experience and enables the planner to compute solutions containing such transitions. Thus, CMAX++ is especially useful in robotic domains with repetitive tasks where the true dynamics are intractable to model, such as deformable manipulation, or vary over time due to reasons such as wear and tear. Furthermore, the optimistic model assumption is easier to satisfy, when compared to assumptions used by previous approaches like CMAX, and performance of CMAX++ degrades gracefully with the accuracy of the model reducing to Q-learning in the case where the model is inaccurate everywhere. Limitations of CMAX++ and A-CMAX++ include hyperparameters such as the radius δ and the sequence $\{\alpha_i\}$, which might need to be tuned for the task. However, from our sensitivity experiments (see Appendix 9.3.1) we observe that A-CMAX++ performance is robust to the choice of sequence $\{\alpha_i\}$ as long as it is non-increasing.

Note that Assumption 5.4.2 can be restrictive for tasks where designing a initial optimistic model requires extensive domain knowledge. We have seen examples of domains where this assumption is satisfied in the experiments, where we performed pick and place of an object with the model assuming that it is light weight, and navigation assuming that there are no icy patches. One can construct other such examples where optimism is easy to build in to the model. For example, for a planar pushing task the model can assume that there is no friction between the object and the surface, when in reality there is friction. Once again, this results in an optimistic model which allows the use of CMAX++. One can always construct a naive optimistic model M_{naive} for any robotic task which simply predicts that any action in any state results in the robot reaching the goal. But using M_{naive}, CMAX++ simply degrades to Q-learning and loses its goal-driven behavior from the lack of information in the model. Thus, there is always a tradeoff in ensuring that the model is optimistic and ensuring that the model is accurate, which can be a burden on the practitioner. However, it is infeasible to avoid this tradeoff without resorting to global undirected exploration techniques [Thr92], which are highly sample inefficient, to ensure completeness.

Chapter 6

On the Effectiveness of using Inaccurate Models

Internal models play an important role in generating command corrections from performance errors. As an internal model is made more accurate, learning efficiency and initial performance are improved.

Chae H. An, Christopher G. Atkeson, John M. Hollerbach (1988)

The approaches, presented in Chapters 4 and 5, are in the spirit of this thesis by using online experience to update the behavior of the planner rather than updating the accuracy of the model. To understand the usefulness of such approaches from a theoretical perspective, we will take a closer look at a similar approach that is popularly known as *iterative learning control* in the literature. To derive concrete bounds on the performance of the controller (or plan/policy) as a function of the modeling errors, we consider the simplified setting of continuous linearized systems with quadratic costs (LQR.) Using LQR as our testbed allows easier analysis and provides insights into the performance one can expect from feedforward adjustment to controls using an approximate model, rather than updating the model itself. This chapter is adapted from our original paper [Vem+21].

6.1 Introduction

Iterative learning control (ILC) has seen widespread adoption in a range of control applications where the dynamics of the system are subject to unknown disturbances or in instances where model parameters are misspecified [MDB92]. While traditional feedback-based control methods have been successful at tackling non-repetitive noise, ILC has shown itself to be effective at adjusting to repetitive disturbance through feedforward control adjustment [AKM84]. This was shown empirically in several robotic applications such as manipulation [KNL91], and quadcopter trajectory tracking [SMD12; MSD12] among others. Prior work [AAH88] uses fixed point theory to analyze the conditions for convergence of ILC but does not present performance bounds at convergence. Very recent work [Aga+21] presented a ILC algorithm that is robust to model

mismatch and uncertainty. However, they analyze the algorithm using planning regret, which measures regret with respect to the best open loop plan in hindsight, and do not study how the performance depends on modeling error. Our work contributes to understanding the effectiveness of ILC by studying its worst case performance, as a function of modeling error, in the linear quadratic regulator (LQR) setting with unknown transition dynamics and access to an approximate model of the dynamics.

A simple approach to the LQR problem with an approximate model of the dynamics is to do optimal control using the misspecified model (MM.) The resulting controller is similar to the certainty equivalent controller obtained by performing optimal control on estimated parameters of the regulator and ignoring the uncertainty of the estimates in adaptive control [ÅW13]. Despite the simplicity of MM, it is challenging to quantify its suboptimality, with respect to the optimal LQR controller, as a result of the modeling errors in the approximate model.

Our first contribution is proving worst case cost suboptimality bounds for MM in the finite horizon LQR setting in terms of the modeling error. This requires us to depart from the fixed point analysis used in prior work [MTR19; KPC93], as the solution to the discrete Ricatti equation in the finite horizon is not a fixed point. A key part of our analysis is establishing perturbation bounds by carefully tracking the effect of modeling error through the horizon of the control task. This allows us to quantify the worst case suboptimality gap of MM in the finite horizon LQR setting.

The second contribution is to utilize the same proof techniques as we used for MM to analyze the suboptimality gap of ILC. This allows us to explicitly compare the worst case performance of ILC and MM for LQR problems, and understand why ILC works well in the regime of large modeling errors when MM often performs poorly. Our analysis highlights that the suboptimality gap for ILC is lower than that for MM by higher order terms that can become significant when modeling errors are high. We also show that ILC is capable of keeping the system stable and cost from blowing up even in the presence of large modeling errors, which MM is incapable of. By interpreting the worst case bounds, we identify several linear systems with key characteristics that enable ILC to be robust to large model misspecifications, whereas MM is unable to deal with model errors and results in poor solutions.

The final contribution of this work is to present simple empirical experiments involving optimal control tasks with linear and nonlinear dynamical systems that back the theoretical findings from our analysis. The experiment results reinforce our finding that in the regime of large modeling errors, ILC performs better than MM and synthesizes control inputs that result in smaller suboptimality gaps.

6.2 Problem Setup

We consider the finite horizon linear quadratic regulator (LQR) setting with a horizon H and a fixed initial state $x_0 \in \mathbb{R}^n$. The dynamics of the system are described by unknown matrices $A_t \in \mathbb{R}^{n\times n}$ and $B_t \in \mathbb{R}^{n\times d}$ for $t = 0, \cdots, H-1$ as follows: $x_{t+1} = A_t x_t + B_t u_t$ where $u_t \in \mathbb{R}^d$ is the control input at time step t. Any sequence of control inputs $(u_0, \cdots, u_{H-1})$ results in a state trajectory $(x_0, \cdots, x_H)$. The cost function is defined using matrices $Q \in \mathbb{R}^{n\times n}$, $Q_f \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{d\times d}$ as follows:

$$V_0(x_0) = \sum_{t=0}^{H-1} x_t^T Q x_t + u_t^T R u_t + x_H^T Q_f x_H \tag{6.1}$$

From optimal control literature [AM07], we know that the above cost is minimized by a linear

time-varying state-feedback controller $K^\star = (K_0^\star, \cdots, K_{H-1}^\star)$ with control inputs $u_t = K_t^\star x_t$ satisfying:

$$K_t^\star = -(R + B_t^T P_{t+1}^\star B_t)^{-1} B_t^T P_{t+1}^\star A_t$$
$$P_t^\star = Q + A_t^T P_{t+1}^\star (I + B_t R^{-1} B_t^T P_{t+1}^\star)^{-1} A_t$$

where we initialize $P_H^\star = Q_f$ and the matrices $P_t^\star$ define the optimal cost-to-go incurred using the optimal controller $K^\star$ from time step t as $V_t^\star(x_t) = x_t^T P_t^\star x_t$. For any controller K, we will use the notation $M_t(K)$ to denote the matrix $A_t + B_t K_t$, and the notation $L_t(K)$ to denote the product $\prod_{i=0}^t M_i(K)$. This is useful for conciseness as we can observe that the state trajectory obtained using K can be expressed as $x_t = M_{t-1}(K)x_{t-1} = L_{t-1}(K)x_0$.

We are given access to an approximate model of the dynamics of the system specified by matrices $\hat{A}_t \in \mathbb{R}^{n\times n}$ and $\hat{B}_t \in \mathbb{R}^{n\times d}$ for $t = 0, \cdots, H-1$ such that there exists some $\epsilon_A, \epsilon_B \geq 0$ (also referred to as the modeling error) satisfying $||A_t - \hat{A}_t|| \leq \epsilon_A$ and $||B_t - \hat{B}_t|| \leq \epsilon_B$. For the purposes of this paper, we use the notation $||\cdot||$ to refer to the matrix norm induced by the L2 vector norm. In this paper, we consider two control strategies: optimal control using the misspecified model (MM) and iterative learning control (ILC.)

6.2.1 Optimal Control using Misspecified Model

Optimal control using misspecified model uses the approximate model to synthesize a time-varying linear controller $K^{\mathsf{MM}} = (K_0^{\mathsf{MM}}, \cdots, K_{H-1}^{\mathsf{MM}})$ satisfying:

$$K_t^{\mathsf{MM}} = -(R + \hat{B}_t^T P_{t+1}^{\mathsf{MM}} \hat{B}_t)^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{MM}} \hat{A}_t$$
$$P_t^{\mathsf{MM}} = Q + \hat{A}_t^T P_{t+1}^{\mathsf{MM}} (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{MM}})^{-1} \hat{A}_t$$

where we initialize $P_H^{\mathsf{MM}} = Q_f$ and the control inputs are defined as $u_t^{\mathsf{CE}} = K_t^{\mathsf{MM}} x_t$. One can observe that the controller K^{MM} results in suboptimal cost when executed in the system as it is optimizing the cost under approximate dynamics rather than the true dynamics of the system. Thus, the suboptimality gap $V_0^{\mathsf{MM}}(x_0) - V_0^\star(x_0)$ depends on the approximate dynamics $\hat{A}_t, \hat{B}_t$, and how well they approximate the true dynamics.

6.2.2 Iterative Learning Control

Iterative learning control [AKM84; MDB92] is a framework that is used to efficiently calculate the feedforward input signal adjustment by using information from previous trials to improve the performance in a small number of iterations. An example of an ILC algorithm is shown in Algorithm 14. ILC assumes a rollout access to the system, i.e. we are allowed to conduct full rollouts of horizon H in the system to evaluate the cost and obtain the trajectory under true dynamics (Line 4). Note that this access is only restricted to rollouts, and the true dynamics A_t, B_t are unknown. ILC can be understood as an iterative shooting method where we synthesize control inputs by always evaluating in the true system while computing updates to the controls using the approximate model [AQN06b; Aga+21]. In Algorithm 14, this is achieved by linearizing the dynamics and quadraticizing the cost around the observed trajectory (Line 5) resulting in an LQR problem with the objective:

$$J(\Delta x, \Delta u) = \sum_{t=0}^{H-1} (2x_t + \Delta x_t)^T Q \Delta x_t + (2u_t + \Delta u_t)^T R \Delta u_t + (2x_H + \Delta x_H)^T Q_f \Delta x_H \tag{6.2}$$

Algorithm 14 ILC Algorithm for Linear Dynamical System with Approximate Model

1: **Input:** Approximate model $\hat{A}_t, \hat{B}_t$, Initial state x_0, Step size α, cost matrix Q, R, Q_f
2: Initialize a control sequence $u_{0:H-1}$ using approximate model
3: **while** not converged **do**
4: Rollout $u_{0:H-1}$ on the true system to get trajectory $x_{0:H}$
5: Compute LQR solution $\arg\min_{\Delta x, \Delta u} J(\Delta x, \Delta u)$ subject to $\hat{A}_t \Delta x_t + \hat{B}_t \Delta u_t = \Delta x_{t+1}$
6: Update $u_{0:H-1} = u_{0:H-1} + \alpha \Delta u_{0:H-1}$

where $x_{0:H}$ is the observed trajectory on the true system when executing controls $u_{0:H-1}$, and for any $t = 0, \cdots, H-1$ we have $\hat{A}_t \Delta x_t + \hat{B}_t \Delta u_t = \Delta x_{t+1}$.

At convergence in Algorithm 14, we have $\Delta u = 0$, i.e. the LQR problem in line 5 returns the solution where $\Delta u = 0$. The solution to the LQR problem can be derived in closed form using dynamic programming, and for any $t \in \{0, \cdots, H-1\}$ is given by

$$\Delta u_t = -(R + \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} \hat{B}_t)^{-1} (R u_t + \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} x_{t+1})$$

where P_{t+1}^{ILC} captures the cost-to-go from time step $t+1$ with $P_H^{\mathsf{ILC}} = Q_f$. To obtain $\Delta u_t = 0$ for any $t \in \{0, \cdots, H-1\}$, it is necessary for the following condition to hold,

$$\begin{aligned}
&R u_t + \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} x_{t+1} = 0 \\
&R u_t + \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} (A_t x_t + B_t u_t) = 0 \\
&u_t = -(R + \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} B_t)^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} A_t x_t
\end{aligned}$$

where we use the rollout trajectory to obtain $x_{t+1} = A_t x_t + B_t u_t$. It is important to note that converging to these control inputs require carefully chosing appropriate step sizes α at each iteration in the ILC Algorithm 14. Thus, we can see that the control inputs $u_{0:H-1}$ ILC converges to can be described using a time-varying state-feedback linear controller K^{ILC} defined as:

$$\begin{aligned}
K_t^{\mathsf{ILC}} &= -(R + \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} B_t)^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} A_t \\
P_t^{\mathsf{ILC}} &= Q + \hat{A}_t^T P_{t+1}^{\mathsf{ILC}} (I + B_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}})^{-1} A_t
\end{aligned}$$

where we initialize $P_H^{\mathsf{ILC}} = Q_f$ and the control inputs are defined as $u_t^{\mathsf{ILC}} = K_t^{\mathsf{ILC}} x_t$. We can observe that the ILC converges to control inputs that are different from the ones computed by the optimal controller $K^\star$, and hence achieves suboptimal cost. In the next few sections, we will analyze the suboptimality bounds for both MM and ILC, and show how ILC converges to control sequence that achieves lower costs and is more robust to high modeling errors when compared to MM.

6.2.3 Assumptions

In this section, we will present all the assumptions used in our analysis. Our first assumption is on the cost matrices Q, Q_f and R, also used in [MTR19]:

Assumption 6.2.1. *We assume that Q, Q_f, and R are positive-definite matrices. Note that simply scaling all of Q, Q_f, and R does not change the optimal controller $K^\star$, so we can assume that the smallest singular value of R, $\underline{\sigma}(R) \geq 1$.*

The above assumption allows us to ignore terms relating to singular values of R in the analysis, keeping it concise. The next assumption states that the true system is stable under the optimal controller $K^\star$. Similar notions of stability have been considered in [Coh+18]:

Assumption 6.2.2. *We assume that the optimal controller $K^\star$ satisfies $||A_t + B_t K_t^\star|| \leq 1 - \delta$ for some $0 < \delta \leq 1$ and all $t = 0, \cdots, H-1$.*

Observe that the above assumption implies that $||M_t(K^\star)|| \leq 1 - \delta$ and $||L_t(K^\star)|| \leq (1 - \delta)^{t+1} \leq e^{-\delta(t+1)}$. Finally, we make a crucial assumption about the model that is required for our ILC analysis:

Assumption 6.2.3. *We assume that the matrix $B_t R^{-1} \hat{B}_t^T$ has eigenvalues that have non-negative real parts for all $t = 0, \cdots, H-1$. A sufficient condition for this to hold is that the modeling error satisfy $\epsilon_B \leq \frac{\sigma(B_t^T R B_t)}{||B_t^T R||}$ for all $t = 0, \cdots, H-1$.*

The above assumption ensures that $x^T B_t R^{-1} \hat{B}_t^T x \geq 0$ for any vector $x \in \mathbb{R}^n$ and all time steps t. Intuitively, if this is not true then ILC is not guaranteed to converge to a local minima. A more detailed explanation is given in Appendix 9.4.4.

6.3 Main Results

In this section, we will present the main results concerning the worst case performance bounds of MM and ILC in the LQR setting with an approximate model as described in Section 6.2. Our first theorem bounds the cost suboptimality of any time-varying linear controller $\hat{K}$ in terms of the norm differences $\|K_t^\star - \hat{K}_t\|$:

Theorem 6.3.1. *Suppose $d \leq n$. Denote $\Gamma = 1 + \max_t\{||A_t||, ||B_t||, ||P_t^\star||, ||K_t^\star||\}$. Then under Assumption 6.2.2 and if $||K_t^\star - \hat{K}_t|| \leq \frac{\delta}{2||B_i||}$ for all $t = 0, \cdots, H-1$, we have*

$$\hat{V}_0(x_0) - V_0^\star(x_0) \leq d\Gamma^3 \|x_0\|^2 \sum_{t=0}^{H-1} e^{-\delta t} \|K_t^\star - \hat{K}_t\|^2 \tag{6.3}$$

The proof for the above theorem is given in Appendix 9.4.1.

This theorem is central to our analysis as it states that as long as we can keep the norm differences $||\hat{K}_t - K_t^\star||$ small, then the cost suboptimality scales with the norm difference squared at each time step and goes exponentially down with time step. We will now present results on how we can bound these norm differences for both MM and ILC.

Results for Optimal Control with Misspecified Model Our next lemma bounds the difference $\|K_t^{\mathsf{MM}} - K_t^\star\|$ in terms of $\|P_{t+1}^{\mathsf{MM}} - P_{t+1}^\star\|$ and modeling errors ϵ_A, ϵ_B:

Lemma 6.3.1. *If $||A_t - \hat{A}_t|| \leq \epsilon_A$ and $||B_t - \hat{B}_t|| \leq \epsilon_B$ for $t = 0, \cdots, H-1$, and we have $||P_{t+1}^\star - P_{t+1}^{\mathsf{MM}}|| \leq f_{t+1}^{\mathsf{MM}}(\epsilon_A, \epsilon_B)$ for some function f_{t+1}^{MM}. Then we have under Assumption 6.2.1 for all $t = 0, \cdots, H-1$,*

$$||K_t^\star - K_t^{\mathsf{MM}}|| \leq 14\Gamma^3 \epsilon_t \tag{6.4}$$

where $\Gamma = 1 + \max_t\{||A_t||, ||B_t||, ||P_t^\star||, ||K_t^\star||\}$ and $\epsilon_t = \max\{\epsilon_A, \epsilon_B, f_{t+1}^{\mathsf{MM}}(\epsilon_A, \epsilon_B)\}$.

The proof for the above lemma is given in Appendix 9.4.3.

This result is very promising but there is a big piece still missing: how do we bound $f_{t+1}^{\mathsf{MM}}(\epsilon_A, \epsilon_B)$. To do this, we need to establish perturbation bounds for the discrete ricatti equation in the finite horizon setting. Prior work [KPC93; MTR19] has only established such bounds in the infinite horizon setting using fixed point analysis. Our treatment is significantly different as the finite horizon solution is not a fixed point. Our final perturbation bounds are presented in the theorem below:

Theorem 6.3.2. *If the cost-to-go matrices for the optimal controller and MM controller are specified by $\{P_t^\star\}$ and $\{P_t^{\mathsf{MM}}\}$ such that $P_H^\star = P_H^{\mathsf{MM}} = Q_f$ then,*

$$\begin{aligned}||P_t^\star - P_t^{\mathsf{MM}}|| \leq \|A_t\|^2\|P_{t+1}^\star\|^2(2\|B_t\|\|R^{-1}\|\epsilon_B + \|R^{-1}\|\epsilon_B^2) \\ + 2\|A_t\|\|P_{t+1}^\star\|\epsilon_A + \|P_{t+1}^\star\|\epsilon_A^2 \\ + c_{P_{t+1}^\star}(\|A_t\| + \epsilon_A)^2||P_{t+1}^\star - P_{t+1}^{\mathsf{MM}}||\end{aligned} \tag{6.5}$$

for $t = 0, \cdots, H-1$ where $c_{P_{t+1}^\star} \in \mathbb{R}^+$ is a constant that is dependent only on $P_{t+1}^\star$ if ϵ_A, ϵ_B are small enough such that $\|P_{t+1}^\star - P_{t+1}^{\mathsf{MM}}\| \leq \|P_{t+1}^\star\|^{-1}$. Furthermore, the upper bound (6.5) is tight up to constants that only depend on the true dynamics A_t, B_t, cost matrix R, and $P_{t+1}^\star$.

The proof for the above theorem is given in Appendix 9.4.3.

The above theorem gives us an upper bound for f_t^{MM} for $t = 0, \cdots, H-1$ in Lemma 6.3.1 with $f_H^{\mathsf{MM}} = 0$. The resulting upper bound on $\|K_t^\star - K_t^{\mathsf{MM}}\|$ from Lemma 6.3.1 combined with Theorem 6.3.1 gives us the cost suboptimality bound for MM. Notice that the bound on f_t^{MM} grows quickly as t decreases making f_{t+1}^{MM} in Lemma 6.3.1 the dominant error term that affects the cost suboptimality of MM.

Results for Iterative Learning Control Our final set of results establish similar worst case cost suboptimality bounds for ILC by first establishing a bound on the difference $\|K_t^{\mathsf{ILC}} - K_t^\star\|$ in terms of $\|P_{t+1}^{\mathsf{ILC}} - P_{t+1}^\star\|$ and modeling error ϵ_A, ϵ_B:

Lemma 6.3.2. *If $||A_t - \hat{A}_t|| \leq \epsilon_A$ and $||B_t - \hat{B}_t|| \leq \epsilon_B$ for $t = 0, \cdots, H-1$, and we have $||P_{t+1} - P_{t+1}^{\mathsf{ILC}}|| \leq f_{t+1}^{\mathsf{ILC}}(\epsilon_A, \epsilon_B)$ for some function f_{t+1}^{ILC}. Then we have under Assumption 6.2.1 for all $t = 0, \cdots, H-1$,*

$$||K_t^\star - K_t^{\mathsf{ILC}}|| \leq 6\Gamma^3\epsilon_t \tag{6.6}$$

where $\Gamma = 1 + \max_t\{||A_t||, ||B_t||, ||P_t^\star||, ||K_t^\star||\}$ and $\epsilon_t = \max\{\epsilon_A, \epsilon_B, f_{t+1}^{\mathsf{ILC}}(\epsilon_A, \epsilon_B)\}$.

The proof for the above lemma is given in Appendix 9.4.5.

Similar to MM, we need to bound the crucial term $f_{t+1}^{\mathsf{ILC}}(\epsilon_A, \epsilon_B)$ to bound the norm difference $\|K_t^\star - K_t^{\mathsf{ILC}}\|$ using Lemma 6.3.2. We will present perturbation bounds for the ILC recursion equation given in Section 6.2.2 in the finite horizon setting below:

Theorem 6.3.3. *If the cost-to-go matrices for the optimal controller and iterative learning control are specified by $\{P_t^\star\}$ and $\{P_t^{\mathsf{ILC}}\}$ such that $P_H^\star = P_H^{\mathsf{ILC}} = Q_f$ then we have under Assumption 6.2.3,*

$$\begin{aligned}||P_t^\star - P_t^{\mathsf{ILC}}|| \leq \|A_t\|^2\|P_{t+1}^\star\|^2\|B_t\|\|R^{-1}\|\epsilon_B + \|A_t\|\|P_{t+1}^\star\|\epsilon_A \\ + c_{P_{t+1}^\star}||A_t||(\|A_t\| + \epsilon_A)||P_{t+1}^\star - P_{t+1}^{\mathsf{ILC}}||\end{aligned} \tag{6.7}$$

for $t = 0, \cdots, H-1$ where $c_{P_{t+1}^\star} \in \mathbb{R}^+$ is a constant that is dependent only on $P_{t+1}^\star$ if ϵ_A, ϵ_B are small enough that $\|P_{t+1}^\star - P_{t+1}^{\mathsf{ILC}}\| \leq \|P_{t+1}^\star\|^{-1}$. Furthermore, the upper bound (6.7) is tight upto constants that depend only on the true dynamics A_t, B_t, cost matrix R, and $P_{t+1}^\star$.

The proof for the above theorem is given in Appendix 9.4.5.

The above theorem gives us a bound on f_t^{ILC} for $t = 0, \cdots, H-1$ in Lemma 6.3.2 with $f_H^{\mathsf{ILC}} = 0$. The resulting upper bound on $\|K_t^\star - K_t^{\mathsf{ILC}}\|$ from Lemma 6.3.2 combined with Theorem 6.3.1 gives us the cost suboptimality bound for iterative learning control. Similar to MM, the dominant error term in Lemma 6.3.2 turns out to be f_{t+1}^{ILC} especially for smaller t as the upper bound (6.7) grows quickly as t decreases.

6.4 Interpreting the Worst Case Bounds

The recursive bounds presented in (6.5) and (6.7) make it difficult to compute a concise bound in Theorem 6.3.1. In this section, we will explicitly compare the cost suboptimality bounds for MM and ILC under different scenarios, where the bound can be simplified.

Small Modeling Errors In the regime of small modeling errors $\epsilon_A << 1$ and $\epsilon_B << 1$, we can ignore quadratic terms ϵ_A^2 and ϵ_B^2 in upper bound for MM (6.5) which results in an upper bound that matches that of ILC (6.7) upto a constant. This suggests that when the modeling errors are small, both ILC and MM have almost the same worst case performance, with ILC having better performance over MM by a constant factor. Intuitively, this makes sense as the approximate model is a very good approximation of the true dynamics, and despite using only the model, MM can synthesize a near-optimal controller.

Highly Damped Systems The second scenario we consider is that of a system that is highly damped which implies $\|A_t\| << 1$ for all $t = 0, \cdots, H-1$. In this regime, the upper bound for ILC (6.7) goes down to zero resulting in ILC achieving near-optimal cost despite having non-zero modeling errors ϵ_A, ϵ_B. The suboptimality in ILC (from Lemma 6.3.2) only arises from ϵ_A, ϵ_B and not from f_{t+1}^{ILC} which is 0. In contrast, the upper bound for MM (6.5) does not go down to zero and has terms that depend on ϵ_A^2, which can be significant when ϵ_A is not small. Thus, for highly damped systems we have that the worst case performance of ILC can be significantly better than MM, especially when ϵ_A is large. Intuitively, this can be understood by observing that ILC removes the effect of modeling errors by always performing rollouts using true dynamics, while MM errors are exacerbated by using the approximate model for rollouts. Interestingly, we also notice that the modeling error ϵ_B does not affect the cost-suboptimality in upper bound for MM (6.5) when the system is highly damped.

Weakly Controlled Systems For systems with small $\|B_t\| << 1$, i.e. where the control inputs do not affect the dynamics of the system to a large extent, we can observe that the upper bound for ILC (6.7) reduces to a bound that does not depend ϵ_B. In other words, any modeling error ϵ_B in estimating the B_t matrices does not affect the upper bound (6.7) for ILC. In constrast, the upper bound for MM (6.5) reduces to an expression that has terms that depend on ϵ_B^2, which can become significant when ϵ_B is large. Thus, for systems with $\|B_t\| << 1$, ILC is robust to any modeling errors ϵ_B in the B_t matrices, whereas MM degrades its worst case performance with increasing ϵ_B.

Modeling Error only at the first time step Consider a scenario where the model is inaccurate only at $t = 0$, i.e. $\|A_0 - \hat{A}_0\| \leq \epsilon_A$ and $\|B_0 - \hat{B}_0\| \leq \epsilon_B$, while $\hat{A}_t = A_t$ and $\hat{B}_t = B_t$ for all $t = 1, \cdots, H-1$. In this case, the upper bounds (6.5) and (6.7) simplify greatly as $\|P_t^\star - P_t^{\mathsf{ILC}}\| = \|P_t^\star - P_t^{\mathsf{MM}}\| = 0$ for all $t = 1, \cdots, H-1$, and we only have upper bounds on $||P_0^\star - P_0^{\mathsf{MM}}||$ and $||P_0^\star - P_0^{\mathsf{ILC}}||$ as given by Theorems 6.3.2 and 6.3.3 which when combined with Theorem 6.3.1 gives us the suboptimality bounds:

$$\hat{V}_0^{\mathsf{MM}}(x_0) - V_0^\star(x_0) \leq \mathcal{O}(1) d\Gamma^9 \|x_0\|^2 (\epsilon_A + \epsilon_A^2 + \epsilon_B + \epsilon_B^2)^2 \tag{6.8}$$

$$\hat{V}_0^{\mathsf{ILC}}(x_0) - V_0^\star(x_0) \leq \mathcal{O}(1) d\Gamma^9 \|x_0\|^2 (\epsilon_A + \epsilon_B)^2 \tag{6.9}$$

The above two cost suboptimality bounds highlight the differences between MM and ILC in worst case performance. As described in Section 6.4, if ϵ_A and ϵ_B are small, then MM and ILC worst case performances match up to constants as we can ignore higher order terms. However, in cases where modeling errors ϵ_A and ϵ_B are large and higher order terms like $\epsilon_A^2\epsilon_B$, ϵ_A^4 etc. start becoming significant, the worst case performance of ILC tends to be better than MM as indicated by equations (6.8) and (6.9). Furthermore, the conditions for stability under synthesized control inputs, as stated in Theorem 6.3.1 (and in Lemma 9.4.1 in Appendix 9.4.1,) is harder to satisfy for MM when compared to ILC, especially when modeling errors are large.

6.5 Empirical Results

In this section, we present three empirical experiments: a linear dynamical system with an approximate model, a nonlinear inverted pendulum system with misspecified mass, and a nonlinear planar quadrotor system in the presence of wind. The aim of these experiments is to show that under high modeling errors, ILC is more efficient than MM, thus backing our theoretical findings.[1]

6.5.1 Linear Dynamical System with Approximate Model

In this experiment, we use a linear dynamical system with states $x \in \mathbb{R}^2$ and control inputs $u \in \mathbb{R}$. The dynamics of the system are specified by matrices: $A_t = \begin{bmatrix} 1 & 1 \\ -3 & 1 \end{bmatrix}, B_t = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$. The approximate model we use is constructed by perturbing the dynamics as follows: $\hat{A}_t = A_t + \epsilon\mathbb{I}$, $\hat{B}_t = B_t + \epsilon \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ for any $\epsilon \geq 0$. Observe that this satisfies $||\hat{A}_t - A_t|| \leq \epsilon$ and $||\hat{B}_t - B_t|| \leq \epsilon$. We use a quadratic cost as specified in equation 6.1 with matrices: $Q = Q_f = \mathbb{I}, R = 1$ (more details in Appendix 6.5.1). We can solve for the optimal controller $K^\star$ in closed form using true dynamics A_t, B_t as specified in Section 6.2. We compare MM controller K^{MM} and iterative learning controller K^{ILC} with approximate model $\hat{A}_t, \hat{B}_t$ in Figure 6.1 where we vary ϵ along the X-axis (in log scale) and report the cost suboptimality gap $V_0(x_0) - V_0^\star(x_0)$ on the Y-axis (in log scale) where $V_0(x_0)$ is the cost incurred by K^{MM} or K^{ILC}. To ensure that Assumption 6.2.3 is not violated, the X-axis is capped at $\epsilon = \frac{\underline{\sigma}(B_t^T R B_t)}{||B_t^T R||}$. It is important to note that to generate the plot in Figure 6.1 we directly used the closed form solution for K^{ILC} (as described in Section 6.2) and did not run a iterative learning control algorithm. This was done to ensure that our results do not have any dependence on how well the step size sequence was tuned for ILC.

We can observe that for small modeling errors $\epsilon < 10^{-1}$, ILC outperforms MM by a constant factor (about 4×10^2) as evidenced by the linear trend in log scale. However in the regime of high modeling errors $\epsilon > 10^{-1}$ we observe that the gap between ILC and MM is not a constant factor anymore and grows very quickly as ϵ increases. This can be explained by the fact that for high ϵ, the higher order terms in the gap between ILC and MM starts becoming significant and results in poor performance for MM when compared to ILC. For large epsilons, we also observe that the cost for MM blows up to really big values as the system is not stable anymore under K^{MM} due to violation of the condition in Theorem 6.3.1 (and in Lemma 9.4.1 in Appendix 9.4.1.) This experiment validates our claim from the analysis that ILC tends to perform better in terms of cost and is more robust when modeling errors are high.

[1]The code for all experiments can be found at `https://github.com/vvanirudh/ILC.jl`.

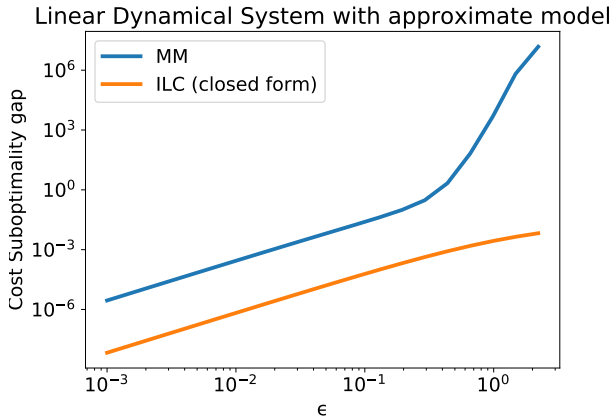

Figure 6.1: Cost suboptimality grap with varying modeling error ϵ for a linear dynamical system. Note that both X-axis and Y-axis are in log scale.

6.5.2 Nonlinear Inverted Pendulum with Misspecified Mass

For the second experiment, we use the nonlinear dynamical system of an inverted pendulum. The state space is specified by $x = \begin{bmatrix}\theta & \dot{\theta}\end{bmatrix} \in \mathbb{R}^2$ where θ is the angle between the pendulum and the vertical axis. The control input is $u = \tau \in \mathbb{R}$ specifying the torque τ to be applied at the base of the pendulum. The dynamics of the system are given by the ODE, $\ddot{\theta} = \frac{\bar{\tau}}{m\ell^2} - \frac{g\sin(\theta)}{\ell}$ where m is the mass of the pendulum, ℓ is the length of the pendulum, g is the acceleration due to gravity, and $\bar{\tau} = \max(\tau_{\min}, \min(\tau_{\max}, \tau))$ is the clipped torque based on torque limits (more details in Appendix 9.4.7). We use an approximate model of the dynamics where the mass of the pendulum is perturbed as $\hat{m} = m + \Delta m$. This results in dynamics that are nonlinearly perturbed from the true dynamics. Since the dynamics are nonlinear, we cannot obtain optimal controls, and MM controls in closed form. Instead, we approximate these controllers by running iLQR [LT04] (both forward and backward pass) on the true dynamics and the approximate dynamics respectively for 200 iterations. To obtain ILC control inputs, we run iLQR with forward pass (or rollouts) using the true dynamics, and backward pass computed using the approximate dynamics at each iteration. We chose step sizes for all iLQR runs using backtracking line search.

Figure 6.2 shows the cost suboptimality gap of MM and ILC as the perturbation Δm varies. Similar to our previous experiment, we observe that for small modeling errors $\Delta m < 0.07$ both ILC and MM perform similarly with ILC outperforming slightly. But as Δm grows, the cost of MM quickly grows saturating at a suboptimality gap around 57. In contrast, we observe that ILC is still able to compute near-optimal controls until $\Delta m = 0.15$ showcasing the robustness of ILC to higher modeling errors. Beyond $\Delta m = 0.15$, ILC performance also degrades significantly as the approximate model is not representative of the true dynamics anymore. Although our analysis in the previous sections was restricted to linear dynamical systems, we notice a similar trend between ILC and MM in the presence of nonlinear dynamics namely, in the regime of large modeling errors, ILC tends to perform better than MM.

6.5.3 Nonlinear Planar Quadrotor Control in Wind

In our final experiment, we compare MM and ILC on a planar quadrotor control task in the presence of wind. A similar setting was used in [Aga+21]. The quadrotor is controlled using

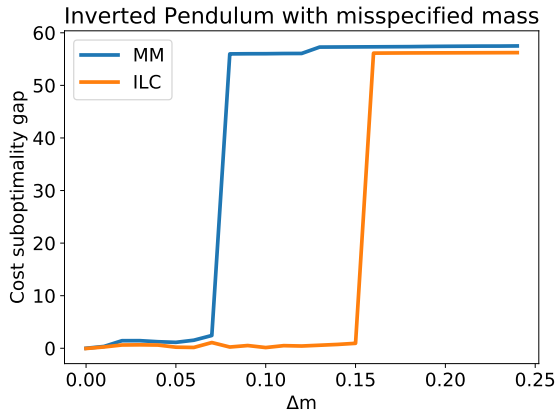

Figure 6.2: Cost suboptimality gap of CE and ILC with varying Δm for a nonlinear inverted pendulum system.

two propellers that provide upward thrusts (u_1, u_2) and allows movement in the 3D planar space described as (p_x, p_y, θ) where p_x, p_y are X, Y positions, and θ is the yaw of the quadrotor. The dynamics of the planar quadrotor is specified using a state vector $x \in \mathbb{R}^6$, and control input $u \in \mathbb{R}^2$ (more details in Appendix 9.4.7). The quadrotor is flying in the presence of wind which is not captured in modeled dynamics, but affects the true dynamics of the quadrotor as a dispersive force field $(\eta p_x \mathbf{i} + \eta p_y \mathbf{j})$ resulting in overall dynamics given by:

$$\ddot{p}_x = \frac{1}{m}(u_1 + u_2)\sin(\theta) + \eta p_x \qquad \ddot{p}_y = \frac{1}{m}(u_1 + u_2)\cos(\theta) - g + \eta p_y$$

where $\eta \in \mathbb{R}^+$ is a constant that captures magnitude of the wind force field.

The objective of the task is to move the quadrotor from an initial state x_0 to a final state x_f. Similar to previous experiment, the dynamics are nonlinear and we cannot obtain optimal controls and MM controls in closed form. Thus, we again approximate these by running iLQR on true dynamics and approximate dynamics respectively. We obtain ILC control inputs again by using iLQR with forward pass using true dynamics and backward pass using approximate dynamics. For all iLQR runs, we choose step sizes by performing backtracking line search and we initialize the control inputs as the hover controls. Figure 6.3 compares MM and ILC for planar quadratic control with varying magnitude of wind η. For small wind magnitudes, we observe that both MM and ILC have good performance. As the wind magnitude increases, MM quickly diverges and the cost of synthesized control inputs blows up quickly as the modeled dynamics are incapable of capturing the dispersive force field exerted by the wind. ILC, on the other hand, manages to keep the cost from blowing up even at large wind magnitudes. This reinforces our conclusion that ILC is robust to large modeling errors while MM can quickly result in the cost blowing up when the model is highly inaccurate.

6.6 Discussion

Iterative Learning Control is known popularly as a higher performance and more robust alternative to optimal control with misspecified model when given access to an inaccurate dynamical model. Our work takes the first steps in laying the theoretical evidence for why ILC has better

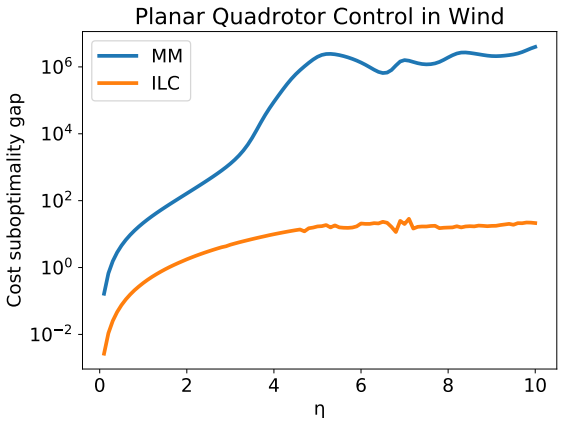

Figure 6.3: Cost suboptimality gap of MM and ILC for planar quadrotor control with varying magnitude of wind η

performance and is more robust when compared to MM. Unlike past work on analyzing the performance of MM in the infinite-horizon setting, we establish our suboptimality bounds in the finite horizon setting in which ILC is typically used. We use ricatti perturbation proof techniques to prove suboptimality bounds in terms of the modeling error ϵ_A, ϵ_B for both ILC and MM, enabling us to compare them. This allows us to identify the reasons for the performance and robustness of ILC.

Our analysis shows that the gap between ILC and MM is in higher order terms that can become significant when the modeling error ϵ_A, ϵ_B is large. This is backed by our empirical experiments where we observe that as the magnitude of modeling error increases, the performance gap between ILC and MM grows rapidly as MM is incapable of handling large modeling errors and the resulting cost diverges. Furthermore, the conditions needed for stability of the system under synthesized control inputs, are easier to satisfy for ILC when compared to MM, especially in the regime of large modeling errors. This explains the robustness of ILC over MM for complex control tasks when given access to highly inaccurate dynamical models. We also identify scenarios where the norms $\|A_t\|$ and $\|B_t\|$ are small, where ILC is provably more efficient and more robust to modeling errors, when compared to MM.

While our current analysis is restricted to the linear quadratic control setting, exploring similar suboptimality bounds in more complex and possibly, nonlinear settings is an exciting direction for future work. Recent work by [SF20] uses a self-bounding ODE method to establish perturbation bounds that sharpens previous bounds in the infinite horizon setting by only depending on natural control-theoretic quantities and not relying on controllability assumptions. It remains to be seen if we can rely on similar techniques to sharpen the bounds presented in this work. It would also be interesting to know whether fast rates for control are possible for cost functions other than quadratic costs. Finally, comparing iterative learning control and robust control approaches such as [Dea+20] would allow us to understand the regime of modeling errors in which ILC is more suitable than robust control approaches, and vice versa.

Chapter 7

Task-Aware Online Model Search with Misspecified Model Classes

More work is needed before planning with learned models can be effective. Environment models should be constructed judiciously with regard to both their states and dynamics with the goal of optimizing the planning process.

Rich Sutton and Andrew Barto (2018)

The algorithms presented in this thesis, so far, have not required any updates to the dynamics of the model. In contrast, most existing methods in the literature, such as [KS02; BT02; JS07; Lju10; AN05; DFR15; AQN06a; RB12; Jia18; RKK18], use experience acquired from executions to update the dynamics of the model or learn a model from scratch. Chapters 4 and 5 have argued that updating the dynamics of the model requires a large amount of experience in large state spaces and can be at the expense of completing the task. While this is generally true, there are major advantages of updating the dynamics of the model, especially in domains where it is feasible to do it online, as it allows the planner to compute solutions that exploit the true dynamics and potentially result in solutions with very low costs. Furthermore even in application domains where we require a large amount of experience to update the model, the improvement in task performance from planning on a more accurate model can outweigh the executions wasted to learn true dynamics. For example, there might be regions in the state space where updating the dynamics of the model can be done efficiently while in other regions we can resort to methods that update the behavior of planner such as Cmax and Cmax++. This motivates a trade-off between both sets of approaches and understanding this trade-off can result in intelligent use of online experience to achieve efficient planning and execution.

This chapter presents Toms an online model search algorithm that updates the dynamics of the model to directly optimize task performance, rather than optimizing prediction error that is commonly done in existing works. This allows Toms to use misspecified model classes, where none of the models are able to capture the true dynamics accurately, and still improve the task performance by updating the model dynamics. This chapter is based on ongoing work.

7.1 Problem Setup

We consider the deterministic shortest path problem represented using the tuple $M = (\mathbb{S}, \mathbb{A}, \mathbb{G}, f, c)$ where $\mathbb{S}$ is the state space, $\mathbb{A}$ is the action space, $\mathbb{G} \subset \mathbb{S}$ is the set of goals that we are interested in reaching, $f : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ is the deterministic dynamics, and $c : \mathbb{S} \times \mathbb{A} \to [0, 1]$ is the cost function. We assume that any goal state $g \in \mathbb{G}$ is a cost-free termination state. The objective of the shortest path problem is to find the least-cost path from a given start state $s_1 \in \mathbb{S}$ to any goal state $g \in \mathbb{G}$ in M. As is typical in shortest path problems, we assume that there exists at least one path from each state $s \in \mathbb{S}$ to one of the goal states in $\mathbb{G}$ [Ber05].

In this work, we focus on environments M with unknown transition dynamics f and known cost function c. Instead we have access to a dynamical model class $\mathcal{F} = \{\hat{f}_\theta : \mathbb{S} \times \mathbb{A} \to \mathbb{S}\}$ that is parameterized by $\theta \in \Theta$. We do not require that $f \in \mathcal{F}$, i.e. the model class $\mathcal{F}$ can be misspecified. This is usually true in real-world domains where the true dynamics can be time-varying and arbitrarily complex. Even in domains where the dynamics are not complex, we might desire to chose a small model class to achieve computational efficiency for methods to run in real-time. The robot gathers knowledge of the true dynamics over a single trajectory in the environment, and does not have access to any resets, ruling out any episodic approach.

We assume we have access to a planner P that when given a dynamical model $\hat{f}_\theta$ results in a policy $\pi_\theta : \mathbb{S} \to \mathbb{A}$ that optimizes the cost-to-go according to the cost function c and transition dynamics $\hat{f}_\theta$. Note that we do not require P to be an optimal planner. We will use the notation $V_\theta^{\pi_\theta}(s)$ to denote the cost-to-go (or state value function) from state s using policy π_θ under the transition dynamics given by $\hat{f}_\theta \in \mathcal{F}$. To denote the cost-to-go of any policy π_θ[1] in M under the true dynamics f, we use the notation V^{π_θ}. Similarly, we will use the notation $Q_\theta^{\pi_\theta}(s, a)$ and $Q^{\pi_\theta}(s, a)$ to denote the state-action value function (or cost-to-go after executing action a in state s) in the model $\hat{f}_\theta$ and in M respectively.

Our method, TOMS, relies crucially on having access to an optimistic dynamical model $f_{\mathsf{opt}} : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$. An optimistic dynamical model satisfies the following assumption [VBL20]:

Assumption 7.1.1. *For any policy π, the cost-to-go V_{opt}^π using the dynamics f_{opt} underestimates the cost-to-go V^π using the true dynamics f at all states, i.e. $V_{\mathsf{opt}}^\pi(s) \leq V^\pi(s)$ for all $s \in \mathbb{S}$.*

where we use the notation V_{opt}^π to denote the cost-to-go of policy π under the transition dynamics of the optimistic model f_{opt}.

7.2 Relevant Prior Work

7.2.1 Maximum Likelihood Model Learning

Most of the existing model learning methods [Lju10; AN05; RB12; Wan+19] use a maximum-likelihood estimation objective (MLE) that optimizes prediction error to choose the best model $\hat{f}_\theta$ among the class $\mathcal{F}$. Given a set of transitions obtained from execution in M, $\mathcal{D} = \{(s_i, a_i, s_{i+1})\}_{i=1}^N$ where $s_i \in \mathbb{S}, a_i \in \mathbb{A}$ and $s_{i+1} = f(s_i, a_i)$ we construct a loss function given as follows:

$$\mathcal{L}^{\mathsf{mle}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (s_{i+1} - \hat{f}_\theta(s_i, a_i))^2 \tag{7.1}$$

[1]Note that we use the same parameterization θ for both policies and dynamical models as we only consider the class of policies that result from applying the planner P on the dynamical model class $\mathcal{F}$.

While we deal with deterministic dynamics in this work, in the stochastic setting the above L2 norm prediction error can be derived from maximum likelihood estimator by modeling the stochastic one-step transition dynamics using a gaussian distribution.

Once we find $\theta^{\mathsf{mle}} = \arg\min_\theta \mathcal{L}_{\mathsf{mle}}(\theta)$, we use the planner P to find the corresponding policy $\pi_{\theta^{\mathsf{mle}}}$ that is used for future executions in M. There are some subtleties that are required in these approaches where we require that the data $\mathcal{D}$ is collected using a sufficiently exploratory policy to ensure good coverage. We will refer the curious reader to these works [Lju10; AN05; RB12; Wan+19] to understand these subtleties.

7.2.2 Reward Based Model Search

[Jos+13] presented an offline model search approach RBMS that eschews maximum likelihood learning and directly optimizes the cost-to-go in M. Similar to our work, they treat the dynamical model class as a parameterization of the policy (using planner P) and search for θ^{rbms} that achieves the least cumulative cost using a gradient descent procedure as follows:

$$\theta = \theta - \alpha \frac{\partial V^{\pi_\theta}(s_1)}{\partial \theta} \tag{7.2}$$

where $\alpha > 0$ is the stepsize.

To compute the update in (7.2) we require gradient of cost-to-go of policy π_θ in M w.r.t θ, i.e. $\frac{\partial V^{\pi_\theta}(s_1)}{\partial \theta}$ which is extremely difficult to compute as the operation that results in V^{π_θ} from θ can be highly nonlinear. [Jos+13] sidestep this difficulty by using a zeroth-order estimate of the gradient obtained from evaluating V^{π_θ} for small perturbations of θ.

Given a policy π_θ, RBMS evaluates V^{π_θ} in an off-policy fashion using offline collected dataset $\mathcal{D} = \{(s_i, a_i, s_{i+1})\}_{i=1}^N$ which consists of transitions in M collected by executing an exploratory policy different from π_θ. For a fixed horizon length H, V^{π_θ} is computed by starting at s_1 and sequentially adds transitions such that at time t when the robot is at state $\tilde{s}$, the next transition is $(s_t, a_t, s_{t+1}) = \arg\min_{(s,a,s')\in\mathcal{D}} \Delta((\tilde{s}, \pi_\theta(\tilde{s})), (s, a))$ where Δ is a user-defined distance metric in state-action space. Note that each transition in $\mathcal{D}$ can only be used once, and the episode is terminated after H steps. The cumulative cost of the resulting episode is used as a proxy for V^{π_θ}.

Given this estimate of V^{π_θ} for any θ, RBMS performs a hill climbing procedure (shown in Algorithm 15) by perturbing θ locally and repeating until we reach a local minima at which it returns the θ^{rbms} found. Since, the objective optimized V^{π_θ} directly corresponds to task performance, RBMS is able to find the best performing model in a misspecified model class that achieves good task performance.

While RBMS is a task-aware model search procedure, it is an offline method that requires the dataset $\mathcal{D}$ to have good coverage. As we show in our experiments, the online version of RBMS, where we are collecting the dataset $\mathcal{D}$ online while executing policies found by RBMS in M, can result in a very poor performance as the dataset $\mathcal{D}$ will not have a good coverage and can result in a highly inaccurate estimate of V^{π_θ}. We will show that our approach Toms sidesteps this by falling back on the optimistic dynamical model f_{opt} in state-action space regions where online data $\mathcal{D}$ does not have good coverage.

Algorithm 15 Model Search Using Derivative-Free Optimization [Jos+13]

```
1: procedure MODELSEARCH(𝒟)
2:     Initial perturbation δ^init, minimum perturbation δ^min, start parameters θ, Initial state s_1,
       δ ← δ^init, planner P
3:     while δ > δ^min do
4:         for each dimension of Θ do
5:             while True do
6:                 Compute {π_{θ−}, π_θ, π_{θ+}} ← {P(f̂_{θ−δ}), P(f̂_θ), P(f̂_{θ+δ})}
7:                 Evaluate {V^{π_{θ−}}, V^{π_θ}, V^{π_{θ+}}}
8:                 if min(V^{π_{θ−}}(s_1), V^{π_{θ+}}(s_1)) > V^{π_θ}(s_1) then
9:                     break
10:                if V^{π_{θ−}}(s_1) < V^{π_{θ+}}(s_1) then
11:                    θ ← θ − δ
12:                else
13:                    θ ← θ + δ
14:        δ ← δ/2
15: return θ
```

7.3 Approach

Our approach TOMS builds upon RBMS by extending to the online setting where policy execution and model updates are interleaved. In addition to the online setting, TOMS also uses a more informative cost-to-go evaluation procedure that allows it to use data collected online that might not have good coverage of state-action space while still improving the task performance of the policy. We will first describe the online setting and present the framework in Section 7.3.1. Subsequently in Section 7.3.2, we will explain our novel off-policy evaluation procedure that uses the optimistic model in state-action space regions where we do not have good data coverage.

7.3.1 Online Model Search

Extending RBMS to the online setting requires us to specify how the data is collected, and once the data is collected how the policy is updated. The online setting we consider is the deterministic shortest path problem described in Section 7.1, where the robot has no access to resets and can gain knowledge about the true dynamics of M through a single trajectory.

The online model search framework is given in Algorithm 16. Starting from the initial state, the robot executes the policy computed by the planner using the dynamical model $\hat{f}_\theta$ where θ is the parameterization of the initial model. For each execution in M using the policy π_θ, we store the resulting transition in the dataset $\mathcal{D}$. Once every ν executions, we use the data collected so far, and run a model search procedure to pick a new θ and update the policy using the planner P. This process is repeated until the robot reaches the goal.

Observe that we collect the dataset $\mathcal{D}$ using the current policy that is used for execution. Note that this is in contrast to the offline setting where we have a fixed dataset that is given to us prior to execution that is collected by executing a fixed exploratory policy. In the online setting, as the policy changes it results in a different set of states visited thereby changing the dataset $\mathcal{D}$. Furthermore, due to the lack of resets we obtain more data in regions where the current policy spends more executions. These characteristics of the online dataset $\mathcal{D}$ pose challenges to

Algorithm 16 Online Model Search Framework

Require: Initial state s_1, Planner P, Initial Model θ, Dataset $\mathcal{D} = \{\}$, Model update frequency $\nu \in \mathbb{Z}$

1: $t \leftarrow 1$, $\pi_\theta \leftarrow P(\hat{f}_\theta)$
2: **while** $s_t \notin \mathbb{G}$ **do**
3: Compute $a_t \leftarrow \pi_\theta(s_t)$
4: Execute a_t in M to get $s_{t+1} = f(s_t, a_t)$
5: Update $\mathcal{D} = \mathcal{D} \cup \{(s_t, a_t, s_{t+1})\}$
6: **if** t is a multiple of ν **then**
7: Update $\theta \leftarrow \mathsf{MODELSEARCH}(\mathcal{D})$
8: Update $\pi_\theta \leftarrow P(\hat{f}_\theta)$

maximum likelihood model learning methods and offline model search methods like RBMS.

Due to the non-stationary nature of the dataset $\mathcal{D}$, maximum likelihood methods find a model that best predicts the true dynamics for states that were visited during execution, and can have terrible prediction everywhere else. This can result in a bad policy that does not make any progress towards the goal. Meanwhile offline methods like RBMS rely on off-policy evaluation procedures (such as the one described in Section 7.2.2) that solely rely on the dataset $\mathcal{D}$. Due to the lack of good coverage in the dataset, these evaluation procedures can be highly inaccurate and pessimistic resulting in convergence to poor policies when used in model search. Our approach TOMS builds upon RBMS by using an optimistic off-policy evaluation procedure that is described in the next section.

7.3.2 Optimistic Off-Policy Evaluation

Given a policy π_θ and dataset $\mathcal{D}$ of transitions executed in M, our goal is to estimate the cost-to-go V^{π_θ} of the policy in M. RBMS estimates this by solely using the dataset $\mathcal{D}$, while our approach TOMS relies on $\mathcal{D}$ only in regions where it has good coverage and falls back on the optimistic dynamical model f_{opt} elsewhere. To determine if the dataset has good coverage for a state-action pair (s, a), we compute the distance to the closest state-action pair in the dataset $\mathcal{D}$. A user-defined distance threshold μ determines whether we use the transition in the dataset to obtain the successor or to fall back on the optimistic model f_{opt}. The optimistic off-policy evaluation procedure is given in Algorithm 17.

Starting from the initial state s_1, we compute the action computed by the policy and find the closest state-action pair from the dataset $\mathcal{D}$ as measured by the distance metric Δ. If the closest state-action pair is closer than μ, then we assign the successor of the closest state-action pair as the next state and remove it from the dataset $\mathcal{D}$. Else, we query the optimistic model f_{opt} to obtain the next state. Our evaluation procedure differs from the evaluation procedure used in RBMS in that we use μ to switch between using dataset $\mathcal{D}$ for state-action regions with good coverage and using optimistic model f_{opt} elsewhere.

Combining all the components so far, TOMS uses the novel optimistic off-policy evaluation procedure from Algorithm 17 in Line 7 of Algorithm 15. The resulting MODELSEARCH procedure is used in the online model search framework presented in Algorithm 16.

Algorithm 17 Optimistic Off-Policy Evaluation

Require: Policy π_θ, Dataset $\mathcal{D}$, start state s_1, horizon H, Distance metric Δ, Distance threshold $\mu \geq 0$

```
Initialize s̃ ← s_1, V̂^{π_θ}(s_1) ← 0
for t = 1 to H do
    Compute ã ← π_θ(s̃)
    Find (s_t, a_t, s_{t+1}) ← arg min_{(s,a,s')∈D} Δ((s̃, ã), (s, a))
    if Δ((s̃, ã), (s_t, a_t)) ≤ μ then
        D ← D \ (s_t, a_t, s_{t+1})
    else
        Compute s_{t+1} ← f_opt(s_t, a_t)
    V̂^{π_θ}(s_1) ← V̂^{π_θ}(s_1) + c(s_t, a_t), s̃ ← s_{t+1}
    if s̃ ∈ G then
        break
return V̂^{π_θ}(s_1)
```

7.4 Theoretical Guarantees

Our first guarantee shows that using optimistic off-policy evaluation as done in Algorithm 17 always results in a cost-to-go estimate that is not too pessimistic. In other words, we have

Theorem 7.4.1. *If the state-action value function Q^{π_θ} is L-lipschitz under the distance metric Δ for any policy π_θ, then we have that the estimate $\hat{V}^{\pi_\theta}$ satsfies*

$$\hat{V}^{\pi_\theta}(s_1) \leq V^{\pi_\theta}(s_1) + LH\mu \tag{7.3}$$

where H is the horizon, and V^{π_θ} is the true cost-to-go of the policy π_θ in M.

Proof Sketch. Observe that if we never use any transition in the dataset $\mathcal{D}$, then we completely rely on the optimistic model f_{opt} for evaluation which by definition results in a cost-to-go estimate $\hat{V}^{\pi_\theta}$ that is optimistic. Now, let's say we use a transition $(s, a, s') \in \mathcal{D}$ to evaluate the cost-to-go of $(\tilde{s}, \tilde{a})$, then we have that

$$\begin{aligned} Q^{\pi_\theta}(s, a) - Q^{\pi_\theta}(\tilde{s}, \tilde{a}) &\leq L\Delta((\tilde{s}, \tilde{a}), (s, a)) \\ &\leq L\mu \end{aligned}$$

Thus, we can introduce a pessimistic error of at most $L\mu$ everytime we use a transition from $\mathcal{D}$. In the worst case, we use H transitions from $\mathcal{D}$, and thus our resulting cost-to-go estimate $\hat{V}^{\pi_\theta}$ can overestimate the true cost-to-go V^{π_θ} by $LH\mu$. □

The above guarantee highlights the advantage of TOMS using an optimistic off-policy evaluation method. By not being severely pessimistic, TOMS never discounts a policy before obtaining enough data to do it with certainty. This allows TOMS to always be optimistic and not rely on inaccurate pessimistic value estimates. The potential downside of this is that we might not be exploiting the dataset $\mathcal{D}$ completely, but turns out collecting more data is not as harmful as switching to a much worse policy based on inaccurate evaluation as evidenced in our experiments.

We can also provide a task-completeness guarantee for TOMS in small state-action spaces (where we can set $\mu = 0$) when given access to unlimited computation (allowing us to solve the

hill-climbing procedure in Algorithm 15 a large number of times to reach global minima) which is stated as follows:

Theorem 7.4.2 (Task Completeness)**.** *In small state-action spaces* TOMS*, given access to unlimited computation, is guaranteed to reach a goal state if there exists at least a single model in the model class $\mathcal{F}$ that is good enough to result in a policy that can reach the goal in M.*

Proof Sketch. From Theorem 7.4.1 we have that when we set $\mu = 0$, the resulting cost-to-go estimates are always optimistic. Thus, as the robot collects more data it either reaches the goal or collects enough data to get the true cost-to-go estimate. In the latter case, since we are performing exact global optimization, TOMS is guaranteed to find the best model in the model class that minimizes the true cost-to-go. As long as this best model results in a policy that can reach the goal, TOMS will also reach the goal by picking this model and using its corresponding policy. □

7.5 Experiments

We compare TOMS against RBMS, maximum likelihood model learning (MLE) and CMAX on a variant of the standard mountain car domain. This variant was first introduced in [Jos+13]. In this variant of the classic mountain car (shown in Figure 7.1), there is a rock (shown in red in the Figure) which causes the car to decrease its speed by c whenever the car moves across the rock in either direction. The dynamics of mountain car are specified as follows:

$$x_{t+1} = x_t + \dot{x}_t$$
$$\dot{x}_{t+1} = \dot{x}_t + a + \theta_1 \cos(\theta_2 x_t)$$

where $a \in \{-0.001, 0.001\}$ is the action executed. We use a uniform discretization of x and $\dot{x}$ with a 150×150 grid. The model class used by all approaches is $\mathcal{F} = \{(\theta_1, \theta_2) | \theta_1, \theta_2 \in \mathbb{R}\}$. The environment M (as shown in Figure 7.1) uses $\theta_1 = -0.0025$ and $\theta_2 = 3$ for the dynamics everywhere except at the rock where the velocity dynamics differ. Therefore, increasing c corresponds to the model class $\mathcal{F}$ becoming increasingly misspecified. The initial position for the car is randomized across each trial by sampling uniformly at random from $[\frac{-\pi}{6} - 0.1, \frac{-\pi}{6} + 0.1]$ with the initial velocity set to 0. The rock is placed at $x = 0.25$ for all the trials with the goal at $x = 0.5$. We use the distance function Δ given by,

$$\Delta((s,a),(s',a')) = \begin{cases} \sum_{d=1}^{D} \frac{|s^d - s'^d|}{s^d_{max} - s^d_{min}} & \text{if } a = a' \\ \infty & \text{otherwise} \end{cases} \tag{7.4}$$

For TOMS, we use the threshold $\mu = 0.01$ as specified by the above distance metric. We use a dense cost function given by $c(s) = (x - 0.5)^4$ for any state $s = (x, \dot{x})$ of the car. We also test TOMS and the baselines when we use a sparse cost function given by $\bar{c}(s) = 1$ for any state $s \notin \mathbb{G}$ and 0 otherwise.

Through our experiments, we would like to answer the following two questions:

1. *Does using* TOMS *help in reaching the goal quickly despite lack of knowledge of the rock dynamics and the model class being misspecified?*
2. *Does using optimistic model for off-policy evaluation help in achieving better value estimates that are useful for reaching the goal quickly?*

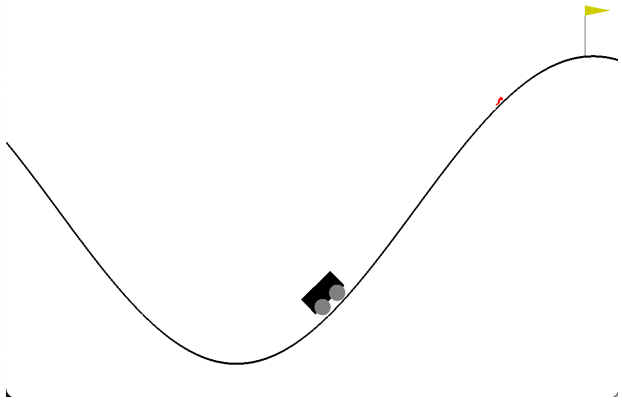

Figure 7.1: Mountain car domain with rock (in red) that decreases the speed of the car by c

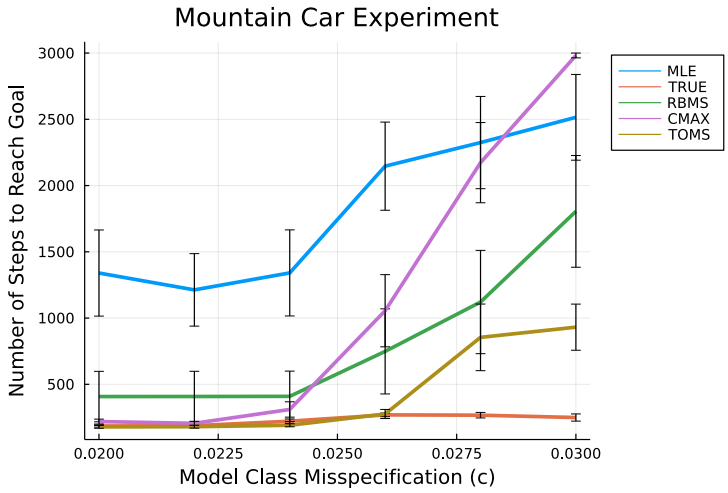

Figure 7.2: Performance versus misspecification on the mountain car domain with dense cost function. We run each method over 10 trials where the initial state of the car is picked at random. We cap each trial at 3000 steps.

Figure 7.2 answers the first question in the affirmative. For this experiment, we use the dense cost function $c(s)$. MLE does a poor job of capturing the dynamics of the car near the rock as most of the data collected online corresponds to the dynamics as specified by $\theta_1 = -0.0025$ and $\theta_2 = 3$, and is not good enough for the planner P to get the car over the rock. The performance of MLE quickly deteriorates and is not successful at reaching the goal in several trials at larger misspecifications. CMAX, on the other hand, does a good job and is successful at all misspecifications except $c = 0.03$. CMAX inflates the cost of any transition with discrepancy in dynamics around the rock until it executes a transition that allows the car to go beyond the rock and onto the goal. However, this take a large number of executions as evidenced by the result in Fiture 7.2. RBMS [Jos+13] does well at small misspecifications but deteriorates for larger misspecifications as the data collected online does not have good coverage of the state-action space. This results in overly pessimistic and inaccurate cost-to-go evaluations that result in convergence to poor policies when used in model search. TOMS does not have this disadvantage as it relies on the optimistic model (in this case, the model with $\theta_1 = -0.0025$ and $\theta_2 = 3$) in

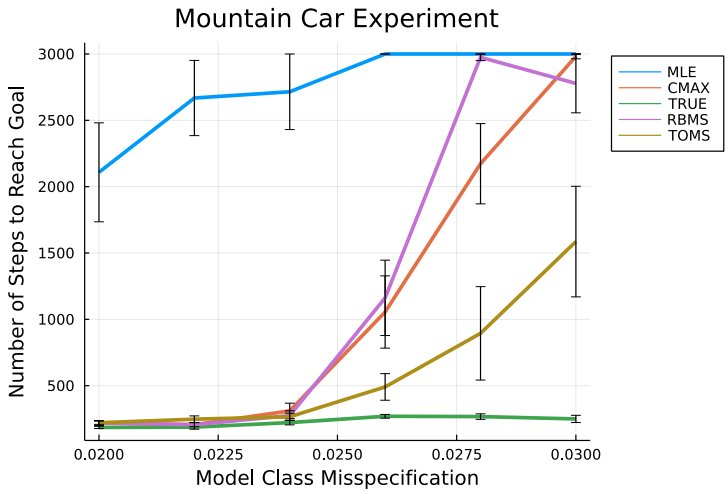

Figure 7.3: Performance versus misspecification on the mountain car domain with sparse cost function. We run each method over 10 trials where the initial state of the car is picked at random. We cap each trial at 3000 steps.

state-action space regions where the data does not have good coverage. This ensures that the cost-to-go evaluations remain optimistic and do not discount models quickly before we gain enough data. As a result, TOMS does better than all the baselines, especially at larger misspecifications, when model learning is essential. For comparison, we have also included a baseline (in green) of the policy that is obtained by planning using the true model with the rock. This shows that there is still a large scope for improvement that one can expect by quickly learning the true dynamics.

We perform a similar experiment, in Figure 7.3, as above but with the sparse cost function $\bar{c}(s)$. Note that with the sparse cost function, the planning problem is difficult as we need to reason in longer horizons and propagate values from the goal all the way to the start. Furthermore, for baselines like RBMS that only rely on dataset $\mathcal{D}$ for evaluation, unless we have already reached the goal (in which case, the task is done) the estimates $\hat{V}^{\pi_\theta}$ are completely uninformative as all policies have the same estimated cost-to-go. In contrast, TOMS can still distinguish between good and bad policies, by falling back on the optimistic model for regions where we do not have data. In other words, during evaluation, TOMS can utilize optimistic model dynamics to connect all the way to the goal and obtain informative cost-to-go estimates. Thus, as shown in Figure 7.3, TOMS outperforms all baselines in the sparse cost setting while the performance of RBMS and MLE have significantly deteriorated when compared to Figure 7.2.

To answer the second question posed above, we compared TOMS with optimistic off-policy evaluation with two variants that use a different off-policy evaluation method. The first one, termed as TOMS Current, where instead of falling back on the optimistic model in Algorithm 17 we use the model $\hat{f}_\theta$ that the evaluated policy π_θ is computed for. The second variant, termed as TOMS Non-optimistic, uses a non-optimistic model $f_{\mathsf{non-opt}}$ in Algorithm 17 instead of the optimistic model. Figure 7.4 presents the results of this experiment. Note that we use the dense cost function for this experiment, but we have obtained similar results with the sparse cost function as well.

From the results, we can observe that TOMS with off-policy evaluation where we fall back on any model that is non-optimistic results in poor performance in reaching the goal. This is also evidenced by Theorem 7.4.1 where TOMS is guaranteed to not result in cost-to-go that are

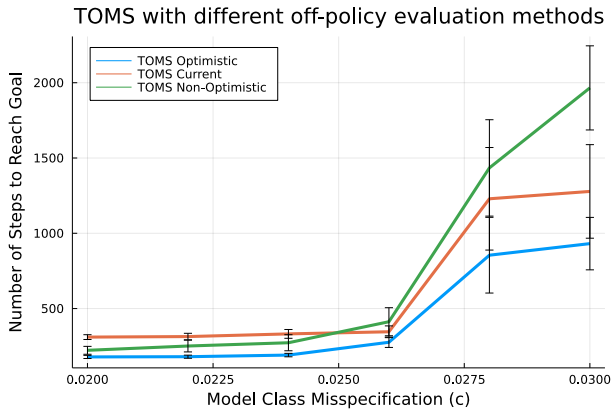

Figure 7.4: Performance versus misspecification on the mountain car domain with dense cost function. We compare TOMS with different off-policy evaluation methods to understand if using optimistic evaluation helps it reach goal quickly

too pessimistic only if we rely on an optimistic model in regions where data is scarce. It is also interesting to observe that relying on the model for which the evaluated policy is optimal for, performs much worse than using an arbitrary fixed non-optimistic model. This can be attributed to the added inaccuracy in evaluation coming from evaluating the cost-to-go using an inaccurate model.

7.6 Conclusion

In this chapter, we presented some preliminary work done in designing a task-aware online model search algorithm. Current model search approaches mostly rely on maximum likelihood learning, where the model is picked to minimize the prediction error and then is subsequently used for planning. This results in an objective mismatch which can result in such approaches picking a model, that while minimizing prediction error, can be very poor for computing a good policy. This is especially true in the regime of limited data which is often the case in online settings. To counter the objective mismatch issue, existing approaches such as RBMS present an offline algorithm that leverages an offline collected dataset of transitions executed in the real world to search in the model class for models that directly optimize task performance. However, such approaches perform poorly in the online setting where the collected dataset is highly correlated and does not have good coverage resulting in data scarcity in state-action regions.

Our proposed approach, TOMS, extends RBMS to the online model search setting by leveraging an optimistic but potentially inaccurate model of the real world. Using this optimistic model for off-policy evaluation allows TOMS to obtain cost-to-go estimates that are not too pessimistic and enables it to work with online collected datasets that can potentially be very scarce and highly correlated. As shown in our experiments, TOMS outperforms both RBMS and maximum likelihood learning approaches on a simple mountain car domain in the online setting. Furthermore our experiments also show that using an optimistic model in the off-policy evaluation procedure is where TOMS derives most of its improvements from.

Chapter 8

Future Work and Conclusion

The great scientists, when an opportunity opens up, get after it and they pursue it. They drop all other things. They get rid of other things and they get after an idea because they had already thought the thing through. Their minds are prepared; they see the opportunity and they go after it. Now of course lots of times it doesn't work out, but you don't have to hit many of them to do some great science. It's kind of easy. One of the chief tricks is to live a long time!

Richard Hamming (1986)

This chapter concludes the thesis by laying out directions for future work. The author has made some progress on some of these directions while for others, pointers are given to related work so that the reader can get started.

8.1 Future Work

8.1.1 A Unified Framework for Planning and Execution using Inaccurate Models

This thesis has presented two novel algorithms CMAX and CMAX++ that update the behavior of the planner, rather than updating the dynamics of the model, to allow robots to complete the task despite using an inaccurate model. CMAX enables the planner to stick to the state-action space regions where the model is accurate and biases it away from the inaccurately modeled regions. CMAX++, on the other hand, learns model-free value estimates for inaccurately modeled transitions and integrates them into a model-based planning procedure with the inaccurate model. Both approaches require the inaccurate model to be optimistic and have task-completeness guarantees. While our experiments have shown that they work very well empirically, there are domains where designing "good" optimistic models is difficult. By good, we mean a non-trivial optimistic model (a trivial optimistic model would be one that predicts any transition executed by robot would

complete the task) that is useful in most state-action space regions. In such domains, updating the dynamics of the model might be more efficient even in cases where the true dynamics does not lie in the model class considered. This thesis has taken preliminary steps in designing such an efficient model learning algorithm in Chapter 7 where we presented TOMS that performs better than CMAX in domains where we can update the dynamics of the model through low-dimensional parameterizations.

An important future direction would be to build upon TOMS to design efficient model learning algorithms that directly optimize task performance and enable choosing models that are useful for planning rather than prediction. Initial work in this direction has been done in [Gri+20; Gri+21; Ayo+20; Mod+21; Nik+21; Eys+21]. Given such algorithms, the author envisons a unified framework for planning and execution where the robot, at every time step, chooses to either update the dynamics of the model from executions (using algorithms like TOMS) or updates the behavior of the planner (using algorithms like CMAX and CMAX++.) This allows us to combine the advantages of both family of algorithms while retaining task completeness guarantees. One viable way of implementing this would be by using the multi-heuristic A* (MHA*) framework [Ain+16] where we treat each algorithm that the robot can use as a heuristic that it can follow to reach the goal. This would involve maintaining a different set of cost-to-go estimates for TOMS, CMAX and CMAX++. Intuitively, we prefer CMAX as it does not waste executions learning dynamics or learning model-free value estimates and quickly finds an alternative path. To encode this preference, we can design an anytime algorithm similar to A-CMAX++ (in Section 5.4.3) where if the cost-to-go following CMAX is not too worse compared to that of TOMS and CMAX++, we follow CMAX. Else, if the cost-to-go following CMAX++ is not too far from that of TOMS, then we follow CMAX++. If neither of those are true, then we follow TOMS. This encodes the preference that avoiding inaccurately modeled transitions is easier to learn than model-free value estimates which is easier to learn than the model dynamics. The goal is to create a unified framework where the robot, during the course of its execution, intelligently switches between (a) learning the true dynamics, (b) learning a model-free value estimate, or (c) biasing the planner away from an inaccurately modeled transition to guarantee task completeness while reducing the amount of real-world experience required.

8.1.2 Online Model Learning with Misspecified Model Classes

While TOMS was a first step in the direction of online model learning with misspecified model classes where we directly optimize task performance rather than prediction error, the author believes there is still a long way to go in this direction. Our main motivation for this comes from the simulation lemma, which was first introduced in [KS02], and can be reformulated in the undiscounted deterministic dynamics setting as follows:

Lemma 8.1.1 (Undiscounted Deterministic Dynamics Simulation Lemma)**.** *Let M, M' be two Markov Decision Processes with the same cost function. If we have a fixed start state s_0, a deterministic policy $\pi : \mathbb{S} \to \mathbb{A}$, and M, M' have deterministic dynamics $f, f' : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$. Then we have,*

$$J_M(\pi) = J_{M'}(\pi) + \sum_{t=0}^{\infty} c(s_t^M, \pi(s_t^M)) + V_{M'}^{\pi}(s_{t+1}^M) - V_{M'}^{\pi}(s_t^M) \tag{8.1}$$

$$= J_{M'}(\pi) + \sum_{t=0}^{\infty} V_{M'}^{\pi}(s_{t+1}^M) - V_{M'}^{\pi}(f'(s_t^M, \pi(s_t^M))) \tag{8.2}$$

where $s_0^M = s_0$ *and* $s_t^M = f(s_{t-1}^M, \pi(s_{t-1}^M))$.

In the case where M is the real world, and M' is any dynamical model that we consider, the above lemma states that the performance of any policy π in the real world M is equal to the sum of the performance of the policy in the model M' and the *model advantages* at each time step $V_{M'}^{\pi}(s_{t+1}^M) - V_{M'}^{\pi}(f'(s_t^M, \pi(s_t^M)))$. Thus, in order to find a model M' that captures the performance of a policy as the same as that of its performance in the real world, we need to minimize model advantages. However, most existing works that perform maximum likelihood learning do not consider this objective function [Lju10; AN05; RB12; Wan+19] and instead use a prediction error loss. For example, [RB12] present a simple iterative approach for agnostic system identification with strong guarantees that do not scale with the size of the MDP when given access to a good exploration distribution. The approach is very simple to implement and iterates between collecting new data about the real world M by executing a good policy under the current model M' as well as by sampling from the exploration distribution, and updating the model with the new data. The model is updated by minimizing negative log likelihood of the data under the model.

To understand why prediction error or maximum likelihood objective makes sense, let us take a closer look at the model advantages:

$$\begin{aligned} V_{M'}^{\pi}(s_{t+1}^M) - V_{M'}^{\pi}(f'(s_t^M, \pi(s_t^M))) &\leq L\|s_{t+1}^M - f'(s_t^M, \pi(s_t^M))\| \\ &\leq L\|f(s_t^M, \pi(s_t^M)) - f'(s_t^M, \pi(s_t^M))\| \end{aligned}$$

where we assumed that the value function of policy π in the model M' is L-lipschitz (any bounded function on a bounded domain is lipschitz.) Thus, instead of optimizing the model advantages one can optimize the prediction error which is an upper bound on the model advantage [RB12]. However, this can be a very weak upper bound resulting in a high sample complexity requirement.

There are two ways to tackle this: 1) directly optimize $J_M(\pi)$, or 2) optimize the model advantages instead of prediction error. RBMS and Toms take the first way by directly optimizing $J_M(\pi)$, i.e. the performance of the policy in the real world M. The policy class is parameterized by the model class (and the application of planner P) and the performance of policy in M is computed through an off-policy evaluation procedure. While this works for simple domains, off-policy evaluation is not always reliable as the data is collected under a different policy than the policy that is being evaluated resulting in a high variance estimate.

Another option is to take the second way. [VJY21] take this approach in the offline setting where given an offline collected dataset $\mathcal{D}$ they find a model M' that minimizes the model advantages as evaluated on $\mathcal{D}$. Once they find the best model in the model class, they use it for planning to obtain the policy that is then used for execution in the real world M. While this works well in offline settings, the online version would face similar difficulties as RBMS where the online collected dataset $\mathcal{D}$ might not have good coverage and can be highly correlated. Thus, there is a need for an online model learning algorithm that optimizes model advantages.

The author envisions an online iterative approach similar to [RB12] where the model is updated with the new data by minimizing model advantages rather than minimizing negative log-likelihood. The advantage of such an approach is evident in domains where there are no models that capture the true dynamics exactly in the model class (a.k.a misspecified) but there are several models that are useful for planning. Using model advantages, instead of prediction error, allows us to distinguish models that are useful for planning from models that are good at capturing true dynamics.

8.1.3 Extending CMAX and CMAX++ to Stochastic Dynamics

Most robotics systems are described using deterministic dynamics. However, when we move to robotics with complex dynamics (such as contact, friction etc.) we cannot hope to capture everything that is relevant to dynamics within the state description. This leads to partial observability. An easy way to tackle this is to formulate the dynamics as stochastic where the stochasticity results from the unobserved part of the state.

So far, in this thesis, we have restricted ourselves to deterministic dynamics (with the exception of Chapter 3) for ease of exposition and because most of the domains we dealt with had simple dynamics. A natural next step would be to extend the algorithms introduced, such as Cmax and Cmax++, to systems that are described using stochastic dynamics. The major component that is needed for such an extension would be an optimal planner that can work with stochastic dynamics. The planners, such as RTAA* and LRTA*, are restricted to deterministic dynamics. The author predicts that we can use general MDP planners such as value iteration, policy iteration [SB98] for such purposes. There are also more domain-specific planners that work with stochastic dynamics such as stochastic motion roadmaps [ASG07], stochastic extended LQR [SBA16], and stochastic ensemble simulation planning [CRT16], where the general idea is to maximize the probability of task success given stochastic dynamics.

In addition to using an optimal planner that can work with stochastic dynamics, we also need to change how a transition is classified as inaccurately modeled in Cmax and Cmax++. Rather than having a hard decision that is made the first time we execute the transition, we will have to maintain uncertainty estimates on how confident we are that the transition is not modeled accurately. This might incur an additional cost of executing the same transition more than once to classify it as inaccurate with certainty. Similar approaches have been taken in [Kid+20; Yu+20] to classify any transition as being inaccurately modeled. Once we classify a transition as inaccurate, we can proceed with penalization in Cmax and learning a model-free value estimate in Cmax++.

Another challenge in extending to stochastic dynamics would be achieving the task completeness guarantees for the stochastic versions of Cmax and Cmax++. The author envisions that by making modeling assumptions on the stochasticity of the dynamics, and using well-calibrated uncertainty estimates can result in probabilistic task completeness guarantees. However, this needs to be explored before further comments can be made.

8.1.4 Finite Data Performance Analysis

Our performance analysis in Chapter 6 analyzed the performance of the final controller that ILC converged to. However, to converge to this controller, ILC might theoretically require a very large amount of rollouts in the real world leading to a large number of executions. While this analysis is useful in understanding what sort of improvements we can expect over naively using the inaccurate models, it is not realistic of the performance we can expect given a finite amount of rollouts. Having a more fine grained performance analysis that tracks the quality of the controller as a function of both modeling error and the number of rollouts performed so far would be much more beneficial in understanding the usefulness of approaches like Cmax and Cmax++.

8.2 Conclusion

This thesis takes the first steps in the study of algorithms that achieve planning and execution using inaccurate models by updating the behavior of the planner, instead of updating the dynamics of the model. We present two novel algorithms, CMAX and CMAX++, that fit this category and also have provable guarantees on task completeness, i.e. given enough time the planner is bound to find a path to a goal state. In addition to our algorithmic contributions, this thesis also presents a performance analysis of such methods in a simple continuous linearized system setting with quadratic costs. Our analysis highlights the pitfalls of naively using inaccurate models for planning and control, and identifies a significant improvement in performance that can be achieved by updating the behavior of the planner in this setting. Finally, the thesis takes a small step in the alternative paradigm of updating the dynamics of the model by proposing a novel task-aware model learning algorithm TOMS that updates the dynamics to directly optimize task performance rather than prediction error in an online setting where the robot has no access to resets. We conclude the thesis by pointing out several directions for future work convincing the reader that there are still a bountiful of exciting challenges that remain to be solved in this space.

Chapter 9

Appendix

I'm not trying to send you out on the road in search of Valhalla, but merely pointing out that it is not necessary to accept the choices handed down to you by life as you know it. There is more to it than that no one HAS to do something he doesn't want to do for the rest of his life. But then again, if that's what you wind up doing, by all means convince yourself that you HAD to do it. You'll have lots of company.

Hunter S. Thompson (1958)

This chapter contains all the missing details from the previous chapters, especially the proofs, experiment descriptions, and other relevant information. This is done to ensure that the thesis is concise for all readers, and for readers who are interested in low-level details they can refer to this chapter as needed.

9.1 Appendix for Chapter 3

9.1.1 Proof of Theorem 3.3.1

Proof of Theorem 3.3.1. To prove Eq. 3.3 for Alg. 4, we use the proof techniques from [FKM05]. The proof is more simpler than the one in [FKM05] as we do not have to deal with shrinking and reshaping the predictor set Θ.

Denote $u \sim \mathbb{B}_b$ as uniformly sampling u from a b-dim unit ball, $u \sim \mathbb{S}_b$ as uniformly sampling u from the b-dim unit sphere, and $\delta \in (0, 1)$. Consider the loss function $\hat{c}_i(w_i) = \mathbb{E}_{v \sim \mathbb{B}_b}[c_i(\theta_i + \delta v)]$, which is a smoothed version of $c_i(w_i)$. It is shown in [FKM05] that the gradient of $\hat{c}_i$ with respect

to θ is:

$$\begin{aligned}
&\nabla_\theta \hat{c}_i(\theta)|_{\theta=\theta_i} \\
&= \frac{b}{\delta}\mathbb{E}_{u\sim\mathbb{S}_b}[c_i(\theta_i+\delta u)u] \\
&= \frac{b}{\delta}\mathbb{E}_{u\sim\mathbb{S}_b}[((\theta_i+\delta u)^T s_i - a_i)^2 u].
\end{aligned}$$

Hence, the descent direction we take in Alg. 4 is actually an unbiased estimate of $\nabla_\theta \hat{c}_i(\theta)|_{\theta=\theta_i}$. So Alg. 4 can be considered as running OGD with an unbiased estimate of gradient on the sequence of loss $\hat{c}_i(\theta_i)$. It is not hard to show that for an unbiased estimate of $\nabla_\theta \hat{c}_i(\theta)|_{\theta=\theta_i} = \frac{b}{\delta}((\theta_i+\delta u)^T s_i - a_i)^2 u$, the norm is bounded as $b(C^2+C_s^2)/\delta$. Now we can directly applying Lemma 3.1 from [FKM05], to get:

$$\mathbb{E}\left[\sum_{i=1}^T \hat{c}_i(\theta_i)\right] - \min_{\theta^\star\in\Theta}\sum_{i=1}^T \hat{c}_i(\theta^\star) \le \frac{C_\theta b(C^2+C_s^2)}{\delta}\sqrt{T}. \tag{9.1}$$

We can bound the difference between $\hat{c}_i(\theta)$ and $c_i(\theta)$ using the Lipschitiz continuous property of c_i:

$$\begin{aligned}
|\hat{c}_i(\theta) - c_i(\theta)| &= |\mathbb{E}_{v\sim\mathbb{B}_b}[c_i(\theta+\delta v) - c_i(\theta)]| \\
&\le \mathbb{E}_{v\sim\mathbb{B}_b}[|c_i(\theta+\delta v) - c_i(\theta)|] \le L\delta.
\end{aligned} \tag{9.2}$$

Substitute the above inequality back to Eq. 9.1, rearrange terms, we get:

$$\mathbb{E}\left[\sum_{i=1}^T c_i(\theta_i)\right] - \min_{\theta^\star\in\Theta}\sum_{i=1}^T c_i(w^\star) \le \frac{C_\theta b(C^2+C_s^2)}{\delta}\sqrt{T} + 2LT\delta. \tag{9.3}$$

By setting $\delta = T^{-0.25}\sqrt{\frac{C_\theta b(C^2+C_s^2)}{2L}}$, we get:

$$\mathbb{E}\left[\sum_{i=1}^T c_i(\theta_i)\right] - \min_{w^\star\in\Theta}\sum_{i=1}^T c_i(w^\star) \le \sqrt{C_\theta b(C^2+C_s^2)L}\,T^{3/4}.$$

To prove Eq. 3.4 for Alg. 7, we follow the similar strategy in the proof of Alg. 4.

Denote $\epsilon \sim [-1,1]$ as uniformly sampling ϵ from the interval $[-1,1]$, $e \sim \{-1,1\}$ as uniformly sampling e from the set containing -1 and 1. Consider the loss function $\tilde{c}_i(\theta) = \mathbb{E}_{\epsilon\sim[-1,1]}[(\theta^T s_i + \delta\epsilon - a_i)^2]$. One can show that the gradient of $\tilde{c}_i(\theta)$ with respect to θ is:

$$\nabla_\theta \tilde{c}_i(\theta) = \frac{1}{\delta}\mathbb{E}_{e\sim\{-1,1\}}[e(\theta^\top s_i + \delta e - a_i)^2 s_i]. \tag{9.4}$$

As we can see that the descent direction we take in Alg. 7 is actually an unbiased estimate of $\nabla_\theta \tilde{c}_i(\theta)|_{\theta=\theta_i}$. Hence Alg. 7 can be considered as running OGD with unbiased estimates of gradients on the sequence of loss functions $\tilde{c}_i(\theta)$. For an unbiased estimate of the gradient, $\frac{1}{\delta}e(\theta_i^\top s_i + \delta e - a_i)^2 s_i$, its norm is bounded as $(C^2+1)C_s/\delta$. Note that different from Alg. 4, here the maximum norm of the unbiased gradient *is independent of feature dimension b*. Now we apply Lemma 3.1 from [FKM05] on $\tilde{c}_i$, to get:

$$\mathbb{E}\left[\sum_{i=1}^T \tilde{c}_i(\theta_i)\right] - \min_{\theta^\star\in\Theta}\sum_{i=1}^T \tilde{c}_i(\theta^*) \le \frac{C_\theta(C^2+1)C_s}{\delta}\sqrt{T}. \tag{9.5}$$

Again we can bound the difference between $\tilde{c}_i(\theta)$ and $c_i(\theta)$ for any θ using the fact that $(\hat{a}_i - a_i)^2$ is Lipschitz continuous with respect to prediction $\hat{a}_i$ with Lipschitz constant C:

$$\begin{aligned}|\tilde{c}_i(\theta) - c_i(\theta)| &= |\mathbb{E}_{\epsilon\sim[-1,1]}[(\theta^\top s_i + \delta\epsilon - a_i)^2 - (\theta^\top s_i - a_i)^2]| \\ &\leq \mathbb{E}_{\epsilon\sim[-1,-1]}[C\delta|\epsilon|] \leq C\delta. \end{aligned} \tag{9.6}$$

Substitute the above inequality back to Eq. 9.5, rearrange terms:

$$\mathbb{E}\left[\sum_{i=1}^{T} \tilde{c}_i(\theta_i)\right] - \min_{\theta^*\in\Theta} \sum_{i=1}^{T} \tilde{c}_i(\theta^*) \leq \frac{C_\theta(C^2+1)C_s}{\delta}\sqrt{T} + 2C\delta T.$$

Set $\delta = T^{-0.25}\sqrt{\frac{C_\theta(C^2+1)C_s}{2C}}$, we get:

$$\mathbb{E}\left[\sum_{i=1}^{T} \tilde{c}_i(\theta_i)\right] - \min_{\theta^*\in\Theta} \sum_{i=1}^{T} \tilde{c}_i(\theta^*) \leq \sqrt{C_\theta(C^2+1)C_s C}T^{3/4}.$$

□

9.1.2 Proof of Theorem 3.4.1

We first present some useful lemmas below.

Consider the smoothed objective given by $\hat{J}(\theta) = \mathbb{E}_{v\sim\mathbb{B}_d}[J(\theta + \delta v)]$ where $\mathbb{B}_d$ is the unit ball in d dimensions and δ is a positive constant. Using the assumptions stated in Section 3.4.1, we obtain the following useful lemma:

Lemma 9.1.1. *If the objective $J(\theta)$ satisfies the assumptions in Section 3.4.1 and the smoothed objective $\hat{J}(\theta)$ is as given above, then we have that*

1. *$\hat{J}(\theta)$ is also G-Lipschitz and L-smooth*
2. *For all $\theta \in \mathbb{R}^d$, $\|\nabla_\theta J(\theta) - \nabla_\theta \hat{J}(\theta)\| \leq L\delta$*

Proof of Lemma 9.1.1. Consider for any $\theta_1, \theta_2 \in \mathbb{R}^d$,

$$\begin{aligned}|\hat{J}(\theta_1) - \hat{J}(\theta_2)| &= |\mathbb{E}_{v\sim\mathbb{B}_d}[J(\theta_1 + \delta v) - J(\theta_2 + \delta v)]| \\ &\leq \mathbb{E}_{v\sim\mathbb{B}_d}[|J(\theta_1 + \delta v) - J(\theta_2 + \delta v)|] \\ &\leq \mathbb{E}_{v\sim\mathbb{B}_d}[G\|\theta_1 - \theta_2\|] \\ &= G\|\theta_1 - \theta_2\|\end{aligned}$$

The above inequalities are due to the fact that expectation of absolute value is greater than absolute value of expectation, and the G-lipschitz assumption on $J(\theta)$. Thus, the smoothened loss function $\hat{J}(\theta)$ is also G-lipschitz. Similarly consider,

$$\begin{aligned}&\|\nabla_\theta \hat{J}(\theta_1) - \nabla_\theta \hat{J}(\theta_2)\| \\ &\quad= \|\nabla_\theta \mathbb{E}_{v\sim\mathbb{B}_d}[J(\theta_1 + \delta v)] - \nabla_\theta \mathbb{E}_{v\sim\mathbb{B}_d}[J(\theta_2 + \delta v)]\| \\ &\quad= \|\mathbb{E}_{v\sim\mathbb{B}_d}[\nabla_\theta J(\theta_1 + \delta v) - \nabla_\theta J(\theta_2 + \delta v)]\| \\ &\quad\leq \mathbb{E}_{v\sim\mathbb{B}_d}[\|\nabla_\theta J(\theta_1 + \delta v) - \nabla_\theta J(\theta_2 + \delta v)\|] \\ &\quad\leq \mathbb{E}_{v\sim\mathbb{B}_d}[L\|\theta_1 - \theta_2\|] \\ &\quad= L\|\theta_1 - \theta_2\|\end{aligned}$$

The above inequalities are due to the fact that expectation of norm is greater than norm of expectation, and the L-smoothness assumption on $J(\theta_1)$. We interchange the expectation and derivative using the assumptions on $J(\theta_1)$ and the dominated convergence theorem. Thus, the smoothened loss function $\hat{J}(\theta_1)$ is also L-smooth.

We know,

$$\begin{aligned}\nabla_\theta \hat{J}(\theta) &= \nabla_\theta \mathbb{E}_{v\sim\mathbb{B}_d}[J(\theta+\delta v)] \\ &= \mathbb{E}_{v\sim\mathbb{B}_d}[\nabla_\theta J(\theta+\delta v)]\end{aligned}$$

Note that the expectation and derivative can be interchanged using the dominated convergence theorem. Hence, we have

$$\begin{aligned}\|\nabla_\theta \hat{J}(\theta) - \nabla_\theta J(\theta)\| &= \|\mathbb{E}_{u\sim\mathbb{B}_d}[\nabla_\theta J(\theta+\delta v)] - \nabla_\theta J(\theta)\| \\ &\leq \mathbb{E}_{u\sim\mathbb{B}_d}\|\nabla_\theta J(\theta+\delta v) - \nabla_\theta J(\theta)\| \\ &\leq \mathbb{E}_{u\sim\mathbb{B}_d}[L||\delta v||] \\ &\leq L\delta\end{aligned}$$

□

The above lemma will be very useful later when we try to relate the convergence rate for the smoothed objective and the true objective. It is shown in [FKM05; ADX10] that the gradient estimate g_i is an unbiased estimator of the gradient $\nabla_\theta \hat{J}(\theta_i)$. Hence, Algorithm 6 is performing SGD on the smoothed objective $\hat{J}(\theta)$. Using this insight, we can use the convergence rate of SGD for nonconvex functions to stationary points from [GL13] which is given as follows

Lemma 9.1.2 ([GL13])**.** *Consider running SGD on the objective $\hat{J}(\theta)$ that is L-smooth and G-Lipschitz for T steps. Fix initial solution θ_0 and denote $\Delta_0 = \hat{J}(\theta_0) - \hat{J}(\theta^*)$ where θ^* is the point at which $\hat{J}(\theta)$ attains global minimum. Also, assume that the gradient estimate g_i is unbiased and has a bounded variance, i.e. for all i, $\mathbb{E}_i[\|g_i - \nabla_\theta \hat{J}(\theta_i)\|_2^2] \leq V \in \mathbb{R}^+$ where $\mathbb{E}_i$ denotes expectation with randomness only at iteration i conditioned on history upto iteration $i-1$. Then we have,*

$$\frac{1}{T}\sum_{i=1}^{T}\mathbb{E}\|\nabla_\theta \hat{J}(\theta_i)\|_2^2 \leq \frac{2\sqrt{2\Delta_0 L(V+G^2)}}{\sqrt{T}} \tag{9.7}$$

For completeness, we include a proof of the above lemma below.

Proof of Lemma 9.1.2. Denote $\xi_i = g_i - \nabla_\theta \hat{J}(\theta_i)$. Note that $\mathbb{E}_i[\xi_i] = 0$ since the stochastic gradient g_i is unbiased. From $\theta_{i+1} = \theta_i - \alpha g_i$, we have:

$$\begin{aligned}\hat{J}(\theta_{i+1}) &= \hat{J}(\theta_i - \alpha g_i) \\ &\leq \hat{J}(\theta_i) - \nabla_\theta \hat{J}(\theta_i)^\top(\alpha g_i) + \frac{L\alpha^2}{2}\|g_i\|_2^2 \\ &= \hat{J}(\theta_i) - \alpha\nabla_\theta \hat{J}(\theta_i)^\top g_i + \frac{L\alpha^2}{2}\|\xi_i + \nabla_\theta \hat{J}(\theta_i)\|_2^2 \\ &= \hat{J}(\theta_i) - \alpha\nabla_\theta \hat{J}(\theta_i)^\top g_i + \frac{L\alpha^2}{2}(\|\xi_i\|_2^2 + 2\xi_i^\top \nabla_\theta \hat{J}(\theta_i) + \|\nabla_\theta \hat{J}(\theta_i)\|_2^2)\end{aligned}$$

The first inequality above is obtained since the loss function $\hat{J}(\theta)$ is L-smooth. Adding $\mathbb{E}_i$ on both sides and using the fact that $\mathbb{E}_i[\xi_i] = 0$, we have:

$$\begin{aligned}\mathbb{E}_i[\hat{J}(\theta_{i+1})] &= \hat{J}(\theta_i) - \alpha\|\nabla_\theta \hat{J}(\theta_i)\|_2^2 + \frac{L\alpha^2}{2}\left(\mathbb{E}_i[\|\xi_i\|_2^2] + \|\nabla_\theta \hat{J}(\theta_i)\|_2^2\right) \\ &\leq \hat{J}(\theta_i) - \alpha\|\nabla_\theta \hat{J}(\theta_i)\|_2^2 + \frac{L\alpha^2}{2}\left(\mathbb{E}_i[\|\xi_i\|_2^2] + G^2\right)\end{aligned}$$

where the inequality is due to the lipschitz assumption. Rearranging terms, we get:

$$\begin{aligned}\alpha\|\nabla_\theta \hat{J}(\theta_i)\|_2^2 &= \hat{J}(\theta_i) - \mathbb{E}_i[\hat{J}(\theta_{i+1})] + \frac{L\alpha^2}{2}(\mathbb{E}_i[\|\xi_i\|_2^2] + G^2) \\ &\leq \hat{J}(\theta_i) - \mathbb{E}_i[\hat{J}(\theta_{i+1})] + \frac{L\alpha^2}{2}(V + G^2)\end{aligned}$$

Sum over from time step 1 to T, we get:

$$\alpha\sum_{t=1}^{T}\mathbb{E}\|\nabla_\theta \hat{J}(\theta_i)\|_2^2 \leq \mathbb{E}[\hat{J}(\theta_0) - \hat{J}(\theta_T)] + \frac{LT\alpha^2}{2}(V + G^2)$$

Divide α on both sides, we get:

$$\begin{aligned}\sum_{t=1}^{T}\mathbb{E}\|\nabla_\theta \hat{J}(\theta_i)\|_2^2 &\leq \frac{1}{\alpha}\mathbb{E}[\hat{J}(\theta_0) - \hat{J}(\theta_T)] + LT\alpha(V + G^2) \\ &\leq \frac{1}{\alpha}\mathbb{E}[\hat{J}(\theta_0) - \hat{J}(\theta^*)] + LT\alpha(V + G^2) \\ &= \frac{1}{\alpha}\Delta_0 + LT\alpha(V + G^2) \\ &\leq \sqrt{\frac{\Delta_0 LT(V + G^2)}{2}} + \sqrt{2\Delta_0 LT(V + G^2)} \\ &\leq 2\sqrt{2\Delta_0 LT(V + G^2)}\end{aligned}$$

with $\alpha = \sqrt{\frac{2\Delta_0}{LT(V+G^2)}}$. Hence, we have:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\|\nabla_\theta \hat{J}(\theta_i)\|_2^2 \leq \frac{2\sqrt{2\Delta_0 L(V + G^2)}}{\sqrt{T}}$$

□

The above lemma is useful as it gives us the following result:

$$\begin{aligned}\min_{1\leq i\leq T}\mathbb{E}\|\nabla_\theta \hat{J}(\theta_i)\|_2^2 &\leq \frac{1}{T}\sum_{i=1}^{T}\mathbb{E}\|\nabla_\theta \hat{J}(\theta_i)\|_2^2 \\ &\leq \frac{2\sqrt{2\Delta_0 L(V + G^2)}}{\sqrt{T}}\end{aligned} \tag{9.8}$$

since the minimum is always less than the average. We have then that using SGD to minimize a nonconvex objective finds a θ_i that is 'almost' a stationary point in bounded number of steps provided the stochastic gradient estimate has bounded variance.

We now show that the gradient estimate g_i used in Algorithm 6 indeed has a bounded variance. Observe that the estimate g_i in the algorithm is a two-point estimate, which should have substantially less variance than one-point estimates [ADX10]. However, the two evaluations, resulting in J_i^+ and J_i^-, have different independent noise. This is due to the fact that in policy search, stochasticity arises from the environment and cannot be controlled and we cannot obtain the significant variance reduction that is typical of two-point estimators. The following lemma quantifies the bound on the variance of gradient estimate g_i:

Lemma 9.1.3. *Consider a smoothed objective $\hat{J}(\theta) = \mathbb{E}_{v\sim\mathbb{B}_d}[J(\theta+\delta v)]$ where $\mathbb{B}_d$ is the unit ball in d dimensions, $\delta > 0$ is a scalar and the true objective $J(\theta)$ is G-lipschitz. Given gradient estimate $g_i = \frac{d(J_i^+ - J_i^-)}{2\delta}u$ where u is sampled uniformly from a unit sphere $\mathbb{S}_d$ in d dimensions, $J_i^+ = J(\theta_i + \delta u) + \eta_i^+$ and $J_i^- = J(\theta - \delta u) + \eta_i^-$ for zero mean random i.i.d noises η_i^+, η_i^-, we have*

$$\mathbb{E}_i[\|g_i - \nabla_\theta \hat{J}(\theta_i)\|_2^2] \leq 2d^2G^2 + 2\frac{d^2\sigma^2}{\delta^2} \tag{9.9}$$

where σ^2 is the variance of the random noise η.

Proof of Lemma 9.1.3. From [Sha17], we know that g_i is an unbiased estimate of the gradient of $\hat{J}(\theta_i)$, i.e. $\mathbb{E}_{u_i\sim\mathbb{S}_d}[g_i] = \nabla\hat{J}(\theta_i)$. Thus, we have

$$\begin{aligned}
&\mathbb{E}_{u_i\sim\mathbb{S}_d}\|g_i - \nabla\hat{J}(\theta_i)\|^2 \\
&\quad= \mathbb{E}_{u_i\sim\mathbb{S}_d}[\|g_i\|^2 + \|\nabla\hat{J}(\theta)_i\|^2 - 2g_i^T\nabla\hat{J}(\theta_i)] \\
&\quad= \mathbb{E}_{u_i\sim\mathbb{S}_d}\|g_i\|^2 + \|\nabla\hat{J}(\theta_i)\|^2 - 2\|\nabla\hat{J}(\theta_i)\|^2 \\
&\quad= \mathbb{E}_{u_i\sim\mathbb{S}_d}\|g_i\|^2 - \|\nabla\hat{J}(\theta_i)\|^2 \\
&\quad\leq \mathbb{E}_{u_i\sim\mathbb{S}_d}\|g_i\|^2 \\
&\quad= \frac{d^2}{4\delta^2}\mathbb{E}_{u_i\sim\mathbb{S}_d}\|(J(\theta_i+\delta u_i) - J(\theta_i - \delta u_i) + (\eta_i^+ - \eta_i^-))u_i\|^2 \\
&\quad\leq \frac{d^2}{2\delta^2}[\mathbb{E}_{u_i\sim\mathbb{S}_d}\|(J(\theta_i+\delta u_i) - J(\theta_i - \delta u_i)u_i\|_2^2 + \mathbb{E}_{u_i\sim\mathbb{S}_d}\|(\eta_i^+ - \eta_i^-))u_i\|^2] \\
&\quad\leq \frac{d^2}{2\delta^2}[\mathbb{E}_{u_i\sim\mathbb{S}_d}4G^2\delta^2\|u_i\|^2 + 4\mathbb{E}_{u_i\sim\mathbb{S}_d}\|\eta_i^+\|_2^2\|u_i\|_2^2] \\
&\quad= 2d^2G^2 + 2\frac{d^2\sigma^2}{\delta^2}
\end{aligned}$$

where the second inequality is true as $\|a+b\|_2^2 \leq 2(\|a\|_2^2 + \|b\|_2^2)$ and the last inequality is due to the Lipschitz assumption on $J(\theta)$. □

We are ready to prove Theorem 3.4.1.

Proof of Theorem 3.4.1. Fix initial solution θ_0 and denote $\Delta_0 = \hat{J}(\theta_0) - \hat{J}(\theta^*)$ where $\hat{J}(\theta)$ is the smoothed objective and θ^* is the point at which $\hat{J}(\theta)$ attains global minimum. Since the gradient estimate g_i used in Algorithm 6 is an unbiased estimate of the gradient $\nabla_\theta\hat{J}(\theta_i)$, we know that Algorithm 6 performs SGD on the smoothed objective. Moreover, from Lemma 9.1.3, we know that the variance of the gradient estimate g_i is bounded. Hence, we can use Lemma 9.1.2 on the smoothed objective $\hat{J}(\theta)$ to get

$$\frac{1}{T}\sum_{i=1}^{T}\mathbb{E}\|\nabla_\theta\hat{J}(\theta_i)\|_2^2 \leq \frac{2\sqrt{2\Delta_0 L(V+G^2)}}{\sqrt{T}} \tag{9.10}$$

where $V \leq 2d^2G^2 + 2\frac{d^2\sigma^2}{\delta^2}$ (from Lemma 9.1.3). We can relate $\nabla_\theta \hat{J}(\theta)$ and $\nabla_\theta J(\theta)$ - the quantity that we ultimately care about, as follows:

$$\begin{aligned}
&\frac{1}{T}\sum_{i=1}^{T}\mathbb{E}\|\nabla_\theta J(\theta_i)\|_2^2 \\
&= \frac{1}{T}\sum_{i=1}^{T}\mathbb{E}\|\nabla_\theta J(\theta_i) - \nabla_\theta \hat{J}(\theta_i) + \nabla_\theta \hat{J}(\theta_i)\|_2^2 \\
&\leq \frac{2}{T}\sum_{i=1}^{T}\mathbb{E}\|\nabla_\theta J(\theta_i) - \nabla_\theta \hat{J}(\theta_i)\|_2^2 + \mathbb{E}\|\nabla_\theta \hat{J}(\theta_i)\|_2^2
\end{aligned}$$

We can use Lemma 9.1.1 to bound the first term and Equation 9.10 to bound the second term. Thus, we have

$$\frac{1}{T}\sum_{i=1}^{T}\mathbb{E}\|\nabla_\theta J(\theta_i)\|_2^2 \leq \frac{2}{T}[TL^2\delta^2 + 2\sqrt{2\Delta_0 L(V+G^2)T}]$$

Substituting the bound for V from Lemma 9.1.3, using the inequality $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ for $a, b \in \mathbb{R}^+$, optimizing over δ, and using $\Delta_0 \leq \mathcal{Q}$ we get

$$\frac{1}{T}\sum_{i=1}^{T}\mathbb{E}\|\nabla_\theta J(\theta_i)\|_2^2 \leq \mathcal{O}(\mathcal{Q}^{\frac{1}{2}} d T^{\frac{-1}{2}} + \mathcal{Q}^{\frac{1}{3}} d^{\frac{2}{3}} T^{\frac{-1}{3}} \sigma)$$

□

9.1.3 Proof of Theorem 3.4.2

The bound on the bias of the gradient estimate is given by the following lemma:

Lemma 9.1.4. *If the assumptions in Section 3.4.2 are satisfied, then for the gradient estimate g_i used in Algorithm 7 and the gradient of the objective $J(\theta)$ given in equation 3.6, we have*

$$\|\mathbb{E}[g_i] - \nabla_\theta J(\theta_i)\| \leq KUH\delta \tag{9.11}$$

Proof of Lemma 9.1.4. To prove that the bias is bounded, let's consider for any i

$$\begin{aligned}
&\|\mathbb{E}[g_i] - \nabla_\theta J(\theta_i)\|_2 = \|\sum_{t=0}^{H-1}\mathbb{E}_{s_t\sim d^t_{\pi_{\theta_i}}}[\nabla_\theta \pi(\theta_i, s_t)\nabla_a(\mathbb{E}_{v\sim\mathbb{B}_p} Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t) + \delta v) - Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t)))]\|_2 \\
&\leq \sum_{t=0}^{H-1}\mathbb{E}_{s_t\sim d^t_{\pi_{\theta_i}}, v\sim\mathbb{B}_p}\|\nabla_\theta \pi(\theta_i, s_t)\|_2\|[\nabla_a Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t) + \delta v) - \nabla_a Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t))]\|_2 \\
&\leq \sum_{t=0}^{H-1} KU\delta\mathbb{E}_{v\sim\mathbb{B}_p}\|v\|_2 \\
&\leq KUH\delta
\end{aligned}$$

The first inequality above is obtained by using the fact that $\|\mathbb{E}[X]\|_2 \leq \mathbb{E}\|X\|_2$, and the second inequality using the K-lipschitz assumption on $\pi(\theta, s)$ and U-smooth assumption on $Q^t_{\pi_\theta}(s, a)$ in a. Also, observe that we interchanged the derivative and expectation above by using the assumptions on $Q^t_{\pi_\theta}$ as stated in Section 3.4.2. □

We will now show that the gradient estimate g_i used in Algorithm 7 has a bounded variance. Note that the gradient estimate constructed in Algorithm 7 is a one-point estimate, unlike policy search in parameter space where we had a two-point estimate. Thus, the variance would be higher and the bound on the variance of such a one-point estimate is given below

Lemma 9.1.5. *Given a gradient estimate g_i as shown in Algorithm 7, the variance of the estimate can be bounded as*

$$\mathbb{E}\|g_i - \mathbb{E}[g_i]\|_2^2 \leq \frac{2H^2p^2K^2}{\delta^2}((\mathcal{Q} + W\delta)^2 + \sigma^2) \tag{9.12}$$

where σ^2 is the variance of the random noise $\tilde{\eta}$.

Proof of Lemma 9.1.5. To bound the variance of the gradient estimate g_i in Algorithm 7, lets consider

$$\begin{aligned}
&\mathbb{E}_i\|g_i - \mathbb{E}[g_i]\|_2^2 = \mathbb{E}_i\|g_i\|_2^2 - \|\mathbb{E}_i[g_i]\|_2^2 \leq \mathbb{E}_i\|g_i\|_2^2 \\
&= \frac{H^2p^2}{\delta^2}\mathbb{E}_i\|\nabla_\theta\pi(\theta_i, s_t)(Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t) + \delta u) + \tilde{\eta}_i)u\|_2^2 \\
&\leq \frac{K^2p^2H^2}{\delta^2}\mathbb{E}_i\|Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t) + \delta u)u + \tilde{\eta}_i u\|_2^2
\end{aligned}$$

where $\mathbb{E}_i$ denotes expectation with respect to the randomness at iteration i and the inequality is obtained using K-lipschitz assumption on $\pi(\theta, s)$. Note that we can express $Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t) + \delta u) \leq Q^t_{\pi_{\theta_i}}(s_t, \pi(\theta_i, s_t)) + W\delta\|u\|_2 \leq \mathcal{Q} + W\delta$ where we used the W-lipschitz assumption on $Q^t_{\pi_\theta}(s, a)$ in a and that it is bounded everywhere by constant $\mathcal{Q}$. Thus, we have

$$\begin{aligned}
&\mathbb{E}_i\|g_i - \mathbb{E}[g_i]\|_2^2 \\
&\leq \frac{K^2p^2H^2}{\delta^2}\mathbb{E}_i\|(\mathcal{Q} + W\delta)u + \tilde{\eta}_i u\|_2^2 \\
&\leq \frac{2K^2p^2H^2}{\delta^2}(\mathbb{E}_i\|(\mathcal{Q} + W\delta)u\|_2^2 + \mathbb{E}_i\|\tilde{\eta}_i u\|_2^2 \\
&\leq \frac{2K^2p^2H^2}{\delta^2}((\mathcal{Q} + W\delta)^2 + \sigma^2)
\end{aligned}$$

□

We are now ready to prove theorem 3.4.2

Proof of Theorem 3.4.2. Fix initial solution θ_0 and denote $\Delta_0 = J(\theta_0) - J(\theta^*)$ where θ^* is the point at which $J(\theta)$ attains global minimum. Denote $\xi_i = g_i - \mathbb{E}_i[g_i]$ and $\beta_i = \mathbb{E}_i[g_i] - \nabla_\theta J(\theta_i)$. From Lemma 9.1.4, we know $\|\beta_i\| \leq KUH\delta$ and from lemma 9.1.5, we know $\mathbb{E}\|\xi_i\|_2^2 = V \leq \frac{2K^2p^2H^2}{\delta^2}((\mathcal{Q} + W\delta)^2 + \sigma^2)$ and $\mathbb{E}_i[\xi_i] = 0$ from definition. From $\theta_{i+1} = \theta_i - \alpha g_i$ we have:

$$\begin{aligned}
J(\theta_{i+1}) &= J(\theta_i - \alpha g_i) \\
&\leq J(\theta_i) - \alpha\nabla_\theta J(\theta_i)^T g_i + \frac{L\alpha^2}{2}\|g_i\|_2^2 \\
&= J(\theta_i) - \alpha\nabla_\theta J(\theta_i)^T g_i + \frac{L\alpha^2}{2}\|\xi_i + \mathbb{E}_i[g_i]\|_2^2 \\
&= J(\theta_i) - \alpha\nabla_\theta J(\theta_i)^T g_i + \frac{L\alpha^2}{2}(\|\mathbb{E}_i[g_i]\|_2^2 + \|\xi_i\|_2^2 + 2\mathbb{E}_i[g_i]^T\xi_i)
\end{aligned}$$

Taking expectation on both sides with respect to randomness at iteration i, we have

$$\begin{aligned}
&\mathbb{E}_i[J(\theta_{i+1})] = J(\theta_i) - \alpha\nabla_\theta J(\theta_i)^T\mathbb{E}_i[g_i] + \frac{L\alpha^2}{2}(\|\mathbb{E}_i[g_i]\|_2^2 + \mathbb{E}_i\|\xi_i\|_2^2 + 2\mathbb{E}_i[g_i]^T\mathbb{E}_i[\xi_i]) \\
&\leq J(\theta_i) - \alpha\nabla_\theta J(\theta_i)^T(\beta_i + \nabla_\theta J(\theta_i)) + \frac{L\alpha^2}{2}(\|\beta_i + \nabla_\theta J(\theta_i)\|_2^2 + V) \\
&= J(\theta_i) - \alpha\|\nabla_\theta J(\theta_i)\|_2^2 + \frac{L\alpha^2}{2}(\|\nabla_\theta J(\theta_i)\|_2^2 + V + \|\beta_i\|_2^2) + (L\alpha^2 - \alpha)\nabla_\theta J(\theta_i)^T\beta_i \\
&\leq J(\theta_i) - \alpha\|\nabla_\theta J(\theta_i)\|_2^2 + \frac{L\alpha^2}{2}(G^2 + V + K^2H^2U^2\delta^2) + (L\alpha^2 - \alpha)\nabla_\theta J(\theta_i)^T\beta_i \\
&\leq J(\theta_i) - \alpha\|\nabla_\theta J(\theta_i)\|_2^2 + \frac{L\alpha^2}{2}(G^2 + V + K^2H^2U^2\delta^2) + (L\alpha^2 + \alpha)\|\nabla_\theta J(\theta_i)\|\|\beta_i\| \\
&\leq J(\theta_i) - \alpha\|\nabla_\theta J(\theta_i)\|_2^2 + \frac{L\alpha^2}{2}(G^2 + V + K^2H^2U^2\delta^2) + (L\alpha^2 + \alpha)GKUH\delta
\end{aligned}$$

Rearranging terms and summing over timestep 1 to T, we get

$$\begin{aligned}
&\alpha\sum_{i=1}^{T}\|\nabla_\theta J(\theta_i)\|_2^2 \leq J(\theta_0) - \mathbb{E}_T[J(\theta_T)] + \frac{LT\alpha^2}{2}(G^2 + V + K^2H^2U^2\delta^2) + (L\alpha^2 + \alpha)GKUHT\delta \\
&\leq \Delta_0 + \frac{LT\alpha^2}{2}(G^2 + V + K^2H^2U^2\delta^2) + (L\alpha^2 + \alpha)GKUHT\delta \\
&\sum_{i=1}^{T}\|\nabla_\theta J(\theta_i)\|_2^2 \leq \frac{\Delta_0}{\alpha} + \frac{LT\alpha}{2}(G^2 + V + K^2H^2U^2\delta^2) + (L\alpha + 1)GKUHT\delta \\
&\leq \frac{\Delta_0}{\alpha} + \frac{LT\alpha}{2}(G^2 + K^2H^2U^2\delta^2 + 2GKUH\delta) + GKUHT\delta + \frac{LT\alpha}{2}V \\
&\leq \frac{\Delta_0}{\alpha} + \frac{LT\alpha}{2}(G + KHU\delta)^2 + GKUHT\delta + \frac{LT\alpha K^2p^2H^2}{\delta^2}((\mathcal{Q} + W\delta)^2 + \sigma^2) \\
&\leq \frac{\Delta_0}{\alpha} + LT\alpha(G^2 + K^2H^2U^2\delta^2) + GKUHT\delta + 2\frac{LT\alpha K^2p^2H^2}{\delta^2}(\mathcal{Q}^2 + W^2\delta^2 + \sigma^2)
\end{aligned}$$

Using $\Delta_0 \leq \mathcal{Q}$ and optimizing over α and δ, we get $\alpha = \mathcal{O}(\mathcal{Q}^{\frac{3}{4}}T^{-\frac{3}{4}}H^{-1}p^{-\frac{1}{2}}(\mathcal{Q}^2 + \sigma^2)^{-\frac{1}{4}})$ and $\delta = \mathcal{O}(T^{-\frac{1}{4}}p^{\frac{1}{2}}(\mathcal{Q}^2 + \sigma^2)^{\frac{1}{4}})$. This gives us

$$\frac{1}{T}\sum_{i=1}^{T}\|\nabla_\theta J(\theta_i)\|_2^2 \leq \mathcal{O}(T^{-\frac{1}{4}}Hp^{\frac{1}{2}}(\mathcal{Q}^3 + \sigma^2\mathcal{Q})^{\frac{1}{4}}) \tag{9.13}$$

□

9.1.4 Implementation Details

One-step Control Experiments

Tuning Hyperparameters for ARS We tune the hyperparameters for ARS [MGR18] in both MNIST and linear regression experiments, by choosing a candidate set of values for each hyperparameter: stepsize, number of directions sampled, number of top directions chosen and the perturbation length along each direction. The candidate hyperparameter values are shown in Table 9.1.

Hyperparameter	Candidate Values
Stepsize	$0.001, 0.005, 0.01, 0.02, 0.03$
# Directions	$10, 50, 100, 200, 500$
# Top Directions	$5, 10, 50, 100, 200$
Perturbation	$0.001, 0.005, 0.01, 0.02, 0.03$

Table 9.1: Candidate hyperparameters used for tuning in ARS experiments

We use the hyperparameters shown in Table 9.2 chosen through this tuning for each of the experiments in this work. The hyperparameters are chosen by averaging the test squared loss across three random seeds (different from the 10 random seeds used in actual experiments) and chosing the setting that has the least mean test squared loss after 100000 samples.

Experiment	Stepsize	# Dir.	# Top Dir.	Perturbation
MNIST	0.02	50	20	0.03
LR $d = 10$	0.03	10	10	0.03
LR $d = 100$	0.03	10	10	0.02
LR $d = 1000$	0.03	200	200	0.03

Table 9.2: Hyperparameters chosen for ARS in each experiment. LR is short-hand for Linear Regression.

Experiment	Learning Rate	Batch size
MNIST	0.001	512
LR $d = 10$	0.08	512
LR $d = 100$	0.03	512
LR $d = 1000$	0.01	512

Table 9.3: Learning rate and batch size used for REINFORCE experiments. We use an ADAM [KB14] optimizer for these experiments.

MNIST Experiments The CNN architecture used is as shown in Figure 9.1[1]. The total number of parameters in this model is $d = 21840$. For supervised learning, we use a cross-entropy loss on the softmax output with respect to the true label. To train this model, we use a batch size of 64 and a stochastic gradient descent (SGD) optimizer with learning rate of 0.01 and a momentum factor of 0.5. We evaluate the test accuracy of the model over all the 10000 images in the MNIST test dataset.

[1]This figure is generated by adapting the code from https://github.com/gwding/draw_convnet

Experiment	Learning Rate	Batch size
LR $d = 10$	2.0	512
LR $d = 100$	2.0	512

Table 9.4: Learning rate and batch size used for Natural REINFORCE experiments. Note that we decay the learning rate after each batch by $\sqrt{T}$ where T is the number of batches seen.

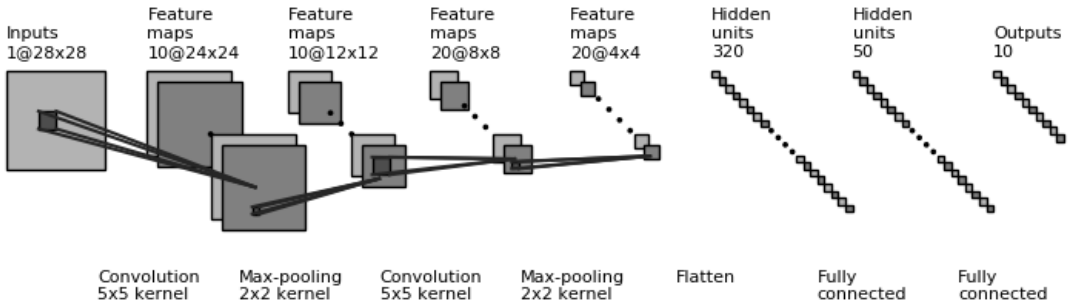

Figure 9.1: CNN architecture used for the MNIST experiments

For REINFORCE, we use the same architecture as before. We train the model by sampling from the categorical distribution parameterized by the softmax output of the model and then computing a ± 1 reward based on whether the model predicted the correct label. The loss function is the REINFORCE loss function given by,

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} r_i \log(\mathbb{P}(\hat{y}_i | x_i, \theta)) \tag{9.14}$$

where θ is the parameters of the model, r_i is the reward obtained for example i, $\hat{y}_i$ is the predicted label for example i and x_i is the input feature vector for example i. The reward r_i is given by $r_i = 2 * \mathbb{I}[\hat{y}_i = y_i] - 1$, where $\mathbb{I}$ is the $0-1$ indicator function and y_i is the true label for example i.

For ARS, we use the same architecture and reward function as before. The hyperparameters used are shown in Table 9.2 and we closely follow the algorithm outlined in [MGR18].

Linear Regression Experiments We generate training and test data for the linear regression experiments as follows: we sampled a random $d+1$ dimensional vector w where d is the input dimensionality. We also sampled a random $d \times d$ covariance matrix C. The training and test dataset consists of $d+1$ vectors x whose first element is always 1 (for the bias term) and the rest of the d terms are sampled from a multivariate normal distribution with mean $\mathbf{0}$ and covariance matrix C. The target vectors y are computed as $y = w^T x + \epsilon$ where ϵ is sampled from a univariate normal distribution with mean 0 and standard deviation 0.001.

We implemented both SGD and Newton Descent on the mean squared loss, for the supervised learning experiments. For SGD, we used a learning rate of 0.1 for $d = 10, 100$ and a learning rate of 0.01 for $d = 1000$, and a batch size of 64. For Newton Descent, we also used a batch size of 64. To frame it as a one-step MDP, we define a reward function r which is equal to the negative of mean squared loss. Both REINFORCE and ARS use this reward function. To compute the

REINFORCE loss, we take the prediction of the model $\hat{w}^T x$, add a mean 0 standard deviation $\beta = 0.5$ Gaussian noise to it, and compute the reward (negative mean squared loss) for the noise added prediction. The REINFORCE loss function is then given by

$$J(w) = \frac{1}{N} \sum_{i=1}^{N} r_i \frac{-(y_i - \hat{w}^T x_i)^2}{2\beta^2} \tag{9.15}$$

where $r_i = -(y_i - \hat{y}_i)^2$, $\hat{y}_i$ is the noise added prediction and $\hat{w}^T x_i$ is the prediction by the model. We use an Adam optimizer with learning rate and batch size as shown in Table 9.3. For the natural REINFORCE experiments, we estimate the fisher information matrix and compute the descent direction by solving the linear system of equations $Fx = g$ where F is the fisher information matrix and g is the REINFORCE gradient. We use SGD with a $O(1/\sqrt{T})$ learning rate, where T is the number of batches seen, and batch size as shown in Table 9.4.

For ARS, we closely follow the algorithm outlined in [MGR18].

Multi-step Control Experiments

Tuning Hyperparameters for ARS We tune the hyperparameters for ARS [MGR18] in both mujoco and LQR experiments, similar to the one-step control experiments. The candidate hyperparameter values are shown in Tables 9.5 and 9.6. We have observed that using all the directions in ARS is always preferable under the low horizon settings that we explore. Hence, we do not conduct a hyperparameter search over the number of top directions and instead keep it the same as the number of directions.

Hyperparameter	**Swimmer-v2**	**HalfCheetah-v2**
Stepsize	$0.03, 0.05, 0.08, 0.1, 0.15$	$0.001, 0.003, 0.005, 0.008, 0.01$
# Directions	$5, 10, 20$	$5, 10, 20$
Perturbation	$0.05, 0.1, 0.15, 0.2$	$0.01, 0.03, 0.05, 0.08$

Table 9.5: Candidate hyperparameters used for tuning in ARS experiments

Hyperparameter	**LQR**
Stepsize	$0.0001, 0.0003, 0.0005, 0.0008, 0.001, 0.003, 0.005, 0.008, 0.01$
# Directions	10
Perturbation	$0.01, 0.05, 0.1$

Table 9.6: Candidate hyperparameters used for tuning in ARS experiments

We use the hyperparameters shown in Tables 9.7 and 9.8 chosen through tuning for each of the multi-step experiments. The hyperparameters are chosen by averaging the total reward obtained across three random seeds (different from the 10 random seeds used in experiments presented in Figure 3.3) and chosing the setting that has the highest total reward after 10000 episodes of training..

Tuning Hyperparameters for ExAct We tune the hyperparameters for ExAct (Algorithm 7) in both mujoco and LQR experiments, similar to ARS. The candidate hyperparameter values

Horizon	Stepsize	# Directions	Perturbation
$H = 1$	0.15	5	0.2
$H = 2$	0.08	5	0.2
$H = 3$	0.15	5	0.2
$H = 4$	0.08	5	0.2
$H = 5$	0.05	5	0.2
$H = 6$	0.08	5	0.2
$H = 7$	0.08	5	0.2
$H = 8$	0.08	5	0.2
$H = 9$	0.1	5	0.2
$H = 10$	0.08	5	0.2
$H = 11$	0.08	5	0.2
$H = 12$	0.1	5	0.2
$H = 13$	0.08	5	0.2
$H = 14$	0.08	5	0.2
$H = 15$	0.08	10	0.2

Table 9.7: Hyperparameters chosen for multi-step experiments for ARS in Swimmer-v2

are shown in Tables 9.9 and 9.10. Similar to ARS, we do not conduct a hyperparameter search over the number of top directions and instead keep it the same as the number of directions.

Hyperparameter	Swimmer-v2	HalfCheetah-v2
Stepsize	$0.005, 0.008, 0.01, 0.015, 0.02, 0.025, 0.03$	$0.0001, 0.0003, 0.0005, 0.0008, 0.001, 0.002, 0.003$
# Directions	$5, 10, 20$	$5, 10, 20$
Perturbation	$0.15, 0.2, 0.3, 0.5$	$0.15, 0.2, 0.3, 0.5$

Table 9.9: Candidate hyperparameters used for tuning in ExAct experiments

We use the hyperparameters shown in Tables 9.11 and 9.12 chosen through tuning for each of the multi-step experiments, similar to ARS.

Mujoco Experiments For all the mujoco experiments, both ARS and ExAct use a linear policy with the same number of parameters as the dimensionality of the state space. The hyperparameters for both algorithms are chosen as described above. Each algorithm is run on both environments (Swimmer-v2 and HalfCheetah-v2) for 10000 episodes of training across 10 random seeds (different from the ones used for tuning). This is repeated for each horizon value $H \in \{1, 2, \cdots, 15\}$. In each experiment, we record the mean evaluation return obtained after training and plot the results in Figure 3.3. For more details on the environments used, we refer the reader to [Bro+16b].

LQR Experiments In the LQR experiments, we constructed a linear dynamical system $x_{t+1} = Ax_t + Bu_t + \xi_t$ where $x_t \in \mathbb{R}^{100}$, $A \in \mathbb{R}^{100\times100}$, $B \in \mathbb{R}^{100}$, $u_t \in \mathbb{R}$ and the noise $\xi_t \sim \mathcal{N}(0_{100}, cI_{100\times100})$ with a small constant $c \in \mathbb{R}^+$. We explicitly make sure that the maximum eigenvalue of A is less than 1 to avoid instability. We fix a quadratic cost function

Horizon	Stepsize	# Directions	Perturbation
$H = 1$	0.001	20	0.08
$H = 2$	0.008	5	0.08
$H = 3$	0.008	10	0.08
$H = 4$	0.003	5	0.05
$H = 5$	0.003	5	0.05
$H = 6$	0.003	10	0.05
$H = 7$	0.008	20	0.05
$H = 8$	0.008	5	0.05
$H = 9$	0.01	20	0.03
$H = 10$	0.005	10	0.03
$H = 11$	0.008	20	0.03
$H = 12$	0.005	5	0.05
$H = 13$	0.008	20	0.03
$H = 14$	0.01	10	0.03
$H = 15$	0.008	20	0.03

Table 9.8: Hyperparameters chosen for multi-step experiments for ARS in HalfCheetah-v2

Hyperparameter	LQR
Stepsize	$0.0001, 0.0003, 0.0005, 0.0008, 0.001, 0.003, 0.005, 0.008, 0.01$
# Directions	10
Perturbation	$0.01, 0.05, 0.1$

Table 9.10: Candidate hyperparameters used for tuning in ExAct experiments

$c(x, u) = x^T Q x + uRu$, where $Q = 10^{-3} I_{100\times100}$ and $R = 1$. The hyperparameters chosen for both algorithms are chosen as described above.

For each algorithm, we run it for noise covariance values $c \in \{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}, 10^{-1}, 5 \times 10^{-1}\}$ until we reach a stationary point where $\|\nabla_\theta J(\theta)\|_2^2 \leq 0.05$. The number of interactions with the environment allowed is capped at 10^6 steps for each run. This is repeated across 10 random seeds (different from the ones used for tuning). The number of interactions needed to reach the stationary point as the noise covariance is increased is recorded and shown in Figure 3.3.

Horizon	Stepsize	# Directions	Perturbation
$H = 1$	0.02	5	0.2
$H = 2$	0.02	5	0.2
$H = 3$	0.015	10	0.2
$H = 4$	0.015	10	0.2
$H = 5$	0.01	10	0.2
$H = 6$	0.015	10	0.2
$H = 7$	0.01	20	0.2
$H = 8$	0.015	20	0.2
$H = 9$	0.02	20	0.2
$H = 10$	0.008	5	0.2
$H = 11$	0.02	5	0.15
$H = 12$	0.02	20	0.2
$H = 13$	0.015	5	0.15
$H = 14$	0.02	10	0.15
$H = 15$	0.01	5	0.1

Table 9.11: Hyperparameters chosen for multi-step experiments for ExAct in Swimmer-v2

Horizon	Stepsize	# Directions	Perturbation
$H = 1$	0.0001	20	0.2
$H = 2$	0.001	5	0.2
$H = 3$	0.001	5	0.2
$H = 4$	0.001	5	0.2
$H = 5$	0.001	10	0.2
$H = 6$	0.001	5	0.2
$H = 7$	0.001	10	0.2
$H = 8$	0.001	5	0.2
$H = 9$	0.001	5	0.2
$H = 10$	0.001	5	0.2
$H = 11$	0.0008	5	0.15
$H = 12$	0.001	5	0.2
$H = 13$	0.001	10	0.2
$H = 14$	0.001	5	0.2
$H = 15$	0.0008	10	0.2

Table 9.12: Hyperparameters chosen for multi-step experiments for ExAct in HalfCheetah-v2

9.2 Appendix for Chapter 4

9.2.1 4D Planar Pushing Experiment Details

In this experiment, the task is for a robotic gripper to push a cube from a start location to a goal location in the presence of static obstacles without any resets, as shown in Figure 4.3 (right). This can be represented as a planning problem in 4D continuous state space $\mathbb{S}$ with any state represented as the tuple $s = (g_x, g_y, o_x, o_y)$ where (g_x, g_y) are the xy-coordinates of the gripper and (o_x, o_y) are the xy-coordinates of the object. The model $\hat{M}$ used for planning *does not* have the static obstacles and the robot can only discover the state-action pairs that are affected due to the obstacles through real world executions. The action space $\mathbb{A}$ is a discrete set of 4 actions that move the gripper end-effector in the 4 cardinal directions by a fixed offset using an IK-based controller. The cost of each transition is 1 when the object is not at the goal location, and 0 otherwise.

For all the approaches (except Q-learning), we use the following neural network architecture for cost-to-go approximation: a feedforward network with 3 hidden layers each of 64 units, the network takes as input a 15D feature representation of the 4D state $s = (o_x, o_y, g_x, g_y)$ that is constructed as follows:

- Relative position of the object w.r.t gripper $\frac{\mathbf{o}-\mathbf{g}}{\|\mathbf{o}-\mathbf{g}\|_2}$, where $\mathbf{o} = (o_x, o_y)$ is the 2D object position and $\mathbf{g} = (g_x, g_y)$ is the 2D gripper position
- Distance between position of the object and gripper $\|\mathbf{o} - \mathbf{g}\|_2$
- Relative position of the object w.r.t. goal $\frac{\mathbf{o}-\mathbf{t}}{\|\mathbf{o}-\mathbf{t}\|_2}$ where $\mathbf{t} = (t_x, t_y)$ is the 2D goal location
- Distance between position of the object and goal location $\|\mathbf{o} - \mathbf{t}\|_2$
- Relative position of the gripper w.r.t goal $\frac{\mathbf{g}-\mathbf{t}}{\|\mathbf{g}-\mathbf{t}\|_2}$
- Distance between position of the gripper and goal location $\|\mathbf{g} - \mathbf{t}\|_2$
- Relative position of the object w.r.t center of the table $\frac{\mathbf{o}-\mathbf{c}}{\|\mathbf{o}-\mathbf{c}\|_2}$
- Distance between position of the object and center of the table $\|\mathbf{o} - \mathbf{c}\|_2$
- Relative position of the gripper w.r.t center of the table $\frac{\mathbf{g}-\mathbf{c}}{\|\mathbf{g}-\mathbf{c}\|_2}$
- Distance between position of the gripper and center of the table $\|\mathbf{g} - \mathbf{c}\|_2$

The output of the network is a single scalar value representing the cost-to-go of the input state. We use ReLU activations after each layer except the last layer. Instead of learning the cost-to-go from scratch, we start with an initial cost-to-go estimate that is hardcoded and the neural network function approximator is used to learn a residual on top of it. The hardcoded initial cost-to-go estimate is obtained as follows:

- For the given object position, construct a target position for the gripper to go to as follows:
  - Get the angle of the vector pointing from the object to the goal location: $\theta = \tan^{-1}(\frac{t_x - o_x}{t_y - o_y})$
  - The target position for gripper is then given by $\mathbf{gt} = (o_x - \frac{\sin(\theta)w}{2}, o_y - \frac{\cos(\theta)w}{2})$ where w is the width of the object
- We compute the manhattan distance from the gripper to its target position $M(\mathbf{g}, \mathbf{gt})$, and from the object to the goal location $M(\mathbf{o}, \mathbf{t})$
- The hardcoded heuristic is obtained as $\hat{V}(s) = \frac{M(\mathbf{g},\mathbf{gt}) + M(\mathbf{o},\mathbf{t})}{d}$, where d is the fixed offset

distance the gripper moves for each action

The residual cost-to-go function approximator is initialized in such a way that it outputs 0 initially for all $s \in \mathbb{S}$. We use a similar residual Q-value function approximator for Q-learning with the same architecture but that takes as input the above feature representation and outputs a vector in $\mathbb{R}^{|\mathbb{A}|}$, where each element corresponds to the Q-value for that action in the input state. We also use hardcoded initial Q-values that are constructed in a similar fashion $\hat{Q}(s,a) = c(s,a) + \hat{V}(\hat{f}(s,a))$. To ensure a fair comparison across all baselines, we use the same neural network function approximator for cost-to-go, and start with the same initial cost-to-go estimates.

For the model learning baseline that uses Neural network function approximator, we use a feedforward neural network with 2 hidden layers each of 32 units, the network takes as input the 4D state s and a one-hot encoding of the discrete action a and outputs a 4D residual vector. The residual vector is added to the next state predicted by the model $\hat{f}(s,a)$ to get the learned next state. The loss function used to train the residual is mean squared loss.

For the model learning baseline that uses KNN function approximator, we use a radius of 0.02, and average the next state residual vector observed for any state within this radius to obtain the prediction for a new state residual vector. In the same way as above, this residual vector is added to the next state predicted by the model $\hat{f}(s,a)$ to obtain the learned next state.

For all the neural network function approximators, we use an Adam optimizer with learning rate of 0.001, and an L2 regularization constant of 0.01. We use a batch size of 64 for training all the neural network function approximators. For Q-learning, we use an random exploration probability of $\epsilon = 0.1$ and change the target network by a polyak coefficient of 0.9.

For all the approaches, we use a limited expansion search planner with $K = 5$ expansions, $N = 5$ planning updates, batch size $B = 64$, and an Adam optimizer [KB15] with learning rate $\eta = 0.001$.

In training the cost-to-go function approximation, we use the hindsight experience replay trick with a probability of 0.8 for sampling any future state in the trajectory as the desired goal. This helps in keeping the function approximation stable and also helps in generalization.

9.2.2 3D Pick-and-Place Experiment Details

The task of this physical robot experiment (Figure 4.4) is to pick and place a heavy object using a PR2 arm from a start pick location to a goal place location while avoiding an obstacle. This can be represented as a planning problem in 3D discrete state space $\mathbb{S}$ where each state corresponds to the 3D location of the end-effector. In our experiment, we discretize each dimension into 20 bins and plan in the resulting discrete state space of size 20^3. Since it is a relatively small state space, we use exact planning updates without any function approximation following Algorithm 12 with $K = 3$ expansions. The action space is a discrete set of 6 actions corresponding to a fixed offset movement in positive or negative direction along each dimension. We use a RRT-based motion planner [LJ01] to plan the path of the arm between states, while avoiding collision with the obstacle. The model $\hat{M}$ used by planning *does not* model the object as heavy and hence, does not capture the dynamics of the arm correctly when it holds the heavy object. The cost of each transition is 1 if object is not at the goal place location, otherwise it is 0.

9.2.3 7D Arm Planning Experiment Details

The task of this physical robot experiment (Figure 4.5) is to move the PR2 arm with a non-operational joint from a start configuration so that the end-effector reaches a goal location,

specified as a 3D region. We represent this as a planning problem in 7D discrete statespace $\mathbb{S}$ where each dimension corresponds to a joint of the arm bounded by its joint limits. Each dimension is discretized into 10 bins resulting in a large state space of size 10^7. The action space $\mathbb{A}$ is a discrete set of size 14 corresponding to moving each joint by a fixed offset in the positive or negative direction. We use an IK-based controller to navigate between discrete states. The model $\hat{M}$ used for planning *does not* know that a joint is non-operational and assumes that the arm can attain any configuration within the joint limits. In the real world, if the robot tries to move the non-operational joint, the arm does not move. Thus, the robot realizes unreachable states only through real world executions.

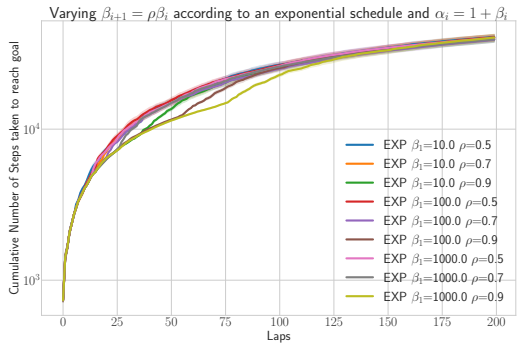

Figure 9.2: Sensitivity experiments with an exponential schedule

9.3 Appendix for Chapter 5

9.3.1 Sensitivity Experiments

In this section, we present the results of our sensitivity experiments examining the performance of A-CMAX++ with the choice of the sequence $\{\alpha_i\}$. We compare the performance of different choices of the sequence $\{\alpha_i\}$ on the 3D mobile robot navigation task. For each run, we average the results across 5 instances with randomly placed ice patches and present the mean and standard errors. To keep the figures concise, we plot the cumulative number of steps taken to reach the goal from the start of the first lap to the current lap across all laps. In all our runs, A-CMAX++ successfully completes all 200 laps and hence, we do not report the number of successful instances in our results.

We choose 4 schedules for the sequence $\{\alpha_i\}$:

1. **Exponential Schedule**: In this schedule, we vary $\beta_{i+1} = \rho\beta_i$ where $\rho < 1$ is a constant that is tuned and $\alpha_i = 1 + \beta_i$. Observe that as $i \to \infty$, $\alpha_i \to 1$ and that the sequence $\{\alpha_i\}$ is a decreasing sequence.

   We vary both the initial β_1 chosen and the constant ρ in our experiments. For β_1 we choose among values $[10, 100, 1000]$ and ρ is chosen among $[0.5, 0.7, 0.9]$. The results are shown in Figure 9.2.

   All choices have almost the same performance with $\beta_1 = 1000$ and $\rho = 0.9$ having the best performance initially but has slightly worse performance in the last several laps. The choice of $\beta_1 = 100$ and $\rho = 0.9$ seems to be a good choice with great performance in both initial and final laps.

2. **Linear Schedule**: In this schedule, we vary $\beta_{i+1} = \beta_i - \eta$ where $\alpha_i = 1 + \beta_i$ and $\eta > 0$ is a constant that is determined so that $\beta_{200} = 0$, i.e. $\alpha_{200} = 1$. Hence, we have $\eta = \frac{\beta_1}{200}$.

   We vary the initial β_1 and choose among values $[10, 100, 200]$. The results are shown in Figure 9.3.

   All three choices have the same performance except in the last few laps where $\beta_1 = 10$ degrades while the other two choices perform well.

3. **Time Decay Schedule**: In this schedule, we vary $\beta_{i+1} = \frac{\beta_1}{i+1}$ where $\alpha_i = 1 + \beta_i$. In other words, we decay β at the rate of $\frac{1}{i}$ where i is the lap number. Again, observe that as $i \to \infty$, we have $\alpha_i \to 1$.

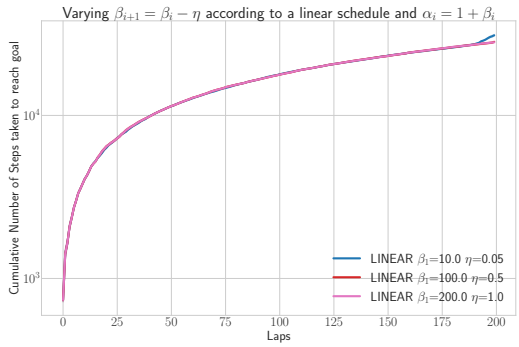

Figure 9.3: Sensitivity experiments with a linear schedule

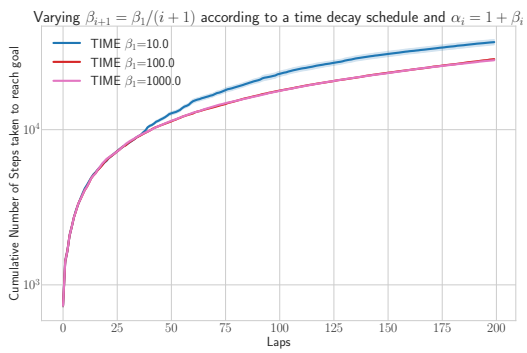

Figure 9.4: Sensitivity experiments with a time decay schedule

We vary the initial β_1 and choose among values $[10, 100, 1000]$. The results are shown in Figure 9.4.

The choices of $\beta_1 = 100$ and $\beta_1 = 1000$ have the best (and similar) performance while $\beta_1 = 10$ has a poor performance as it quickly switches to CMAX++ in the early laps and wastes executions learning accurate Q-values.

4. **Step Schedule**: In this schedule, we vary β as a step function with $\beta_{i+1} = \beta_i - \delta$ if i is a multiple of ξ where ξ is the step frequency, $\alpha_i = 1 + \beta_i$ and δ is a constant that is determined so that $\beta_{200} = 0$, i.e. $\alpha_{200} = 1$. Hence, we have $\delta = \frac{\beta_1 \xi}{200}$.

   We vary both the initial β_1 and the step frequency ξ. For β_1 we choose among values $[10, 100, 200]$ and for ξ we choose among $[5, 10, 20]$. The results are shown in Figure 9.5.

   All choices have the same performance and A-CMAX++ seems to be robust to the choice of step size frequency.

For our final comparison, we will pick the best performing choice among all the schedules and compare performance among these selected choices. The results are shown in Figure 9.6.

We can observe that all schedules have the same performance except the exponential schedule which has worse performance. This can be attributed to the rapid decrease in the value of β compared to other schedules and thus, around lap 50 A-CMAX++ switches to CMAX++ resulting in a large number of executions wasted to learn accurate Q-value estimates. This does not happen

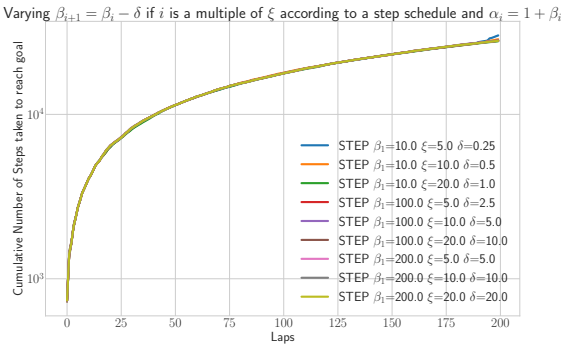

Figure 9.5: Sensitivity experiments with a step schedule

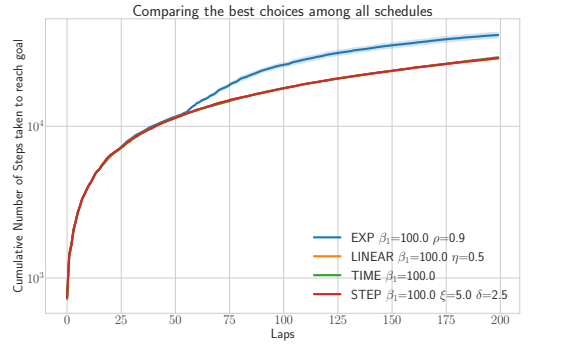

Figure 9.6: Sensitivity experiments with best choices among all schedules

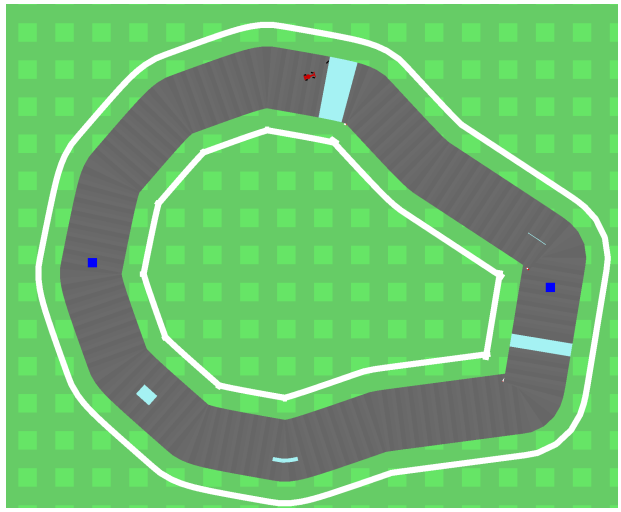

Figure 9.7: 3D Mobile Robot experiment example track

for other schedules as they decrease β gradually and thus, spreading out the executions used to learn accurate Q-value estimates across several laps and not performing poorly in any single lap.

9.3.2 Experiment Details

All experiments were implemented using Python 3.6 and run on a 3.1GHz Intel Core i5 machine. We use PyTorch [Pas+19] to train neural network function approximators in our 7D experiments, and use Box2D [Cat07] for our 3D mobile robot simulation (similar to OpenAI Gym [Bro+16b] `car_racing` environment) and use PyBullet [Cou+13] for our 7D PR2 experiments.

3D Mobile Robot Navigation with Icy Patches

An example track used in the 3D experiment is shown in Figure 9.7. We generate 66 motion primitives offline using the following procedure: (a) We first define the primitive action set for the robot by discretizing the steering angle into 3 cells, one corresponding to zero and the other two corresponding to $+0.6$ and -0.6 radians. We also discretize the speed of the robot to 2 cells corresponding to $+2$m/s and -2m/s, (b) We then discretize the state space into a 100×100 grid in XY space and 16 cells in θ dimension. Thus, we have a $100 \times 100 \times 16$ grid in $XY\theta$ space., (c) We then initialize the robot at $(0, 0)$ xy location with different headings chosen among $[0, \cdots, 15]$ and roll out all possible sequences of primitive actions for all possible motion primitive lengths from 1 to 15 time steps, (d) We filter out all motion primitives whose end point is very close to a cell center in the $XY\theta$ grid. During execution, we use a pure pursuit controller to track the motion primitive so that the robot always starts and ends on a cell center. During planning, we simply use the discrete offsets stored in the motion primitive to compute the next state (and thus, the model dynamics are pre-computed offline during motion primitive generation.)

The cost function used is as follows: for any motion primitive a and state s, the cost of executing a from s is given by $c(s, a) = \sum_{s'} c'(s')$ where c' is a pre-defined cost map over the $100 \times 100 \times 16$ grid and s' is all the intermediate states (including the final state) that the robot goes through while executing the motion primitive a from s. The pre-defined cost map is defined as follows: $c'(s) = 1$ if state s lies on the track (i.e. xy location corresponding to s lies on the

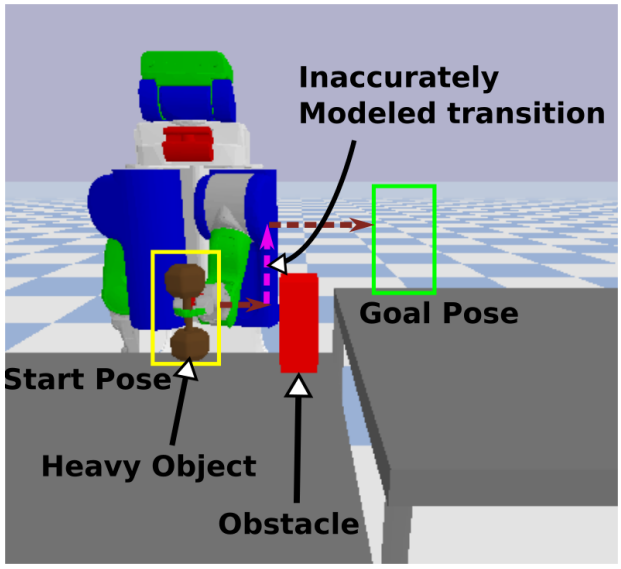

Figure 9.8: 7D Pick-and-Place Experiment

track) and $c'(s) = 100$ otherwise (i.e. all xy locations corresponding to grass or wall has a cost of 100). This encourages the planner to come up with a path that lies completely on the track.

We define two checkpoints on the opposite ends of the track (shown as blue squares in Figure 9.7.) The goal of the robot is to reach the next checkpoint incurring least cost while staying on the track. Note that this requires the robot to complete laps around the track as quickly as possible. Since the state space is small, we maintain value estimates $V, Q, \tilde{V}$ using tables and update the appropriate table entry for each value update. The tables are initialized with value estimates obtained by planning in the model $\hat{M}$ using a planner with $K = 100$ expansions until the robot can efficiently complete the laps using the optimal paths. However, this does not mean that the initial value estimates are the optimal values for $\hat{M}$ dynamics since the planner looks ahead and can achieve optimal paths with underestimated value functions. Nevertheless, these estimates are highly informative.

7D Pick-and-Place with a Heavy Object

For our 7D experiments, we make use of Bullet Physics Engine through the pyBullet interface. For motion planning and other simulation capabilities we make use of `ss-pybullet` library [Gar18]. The task is shown in Figure 9.8. The goal is for the robot to pick the heavy object from its start pose and place it at its goal pose while avoiding the obstacle, without any resets. Since the object is heavy, the robot fails to lift the object in certain configurations where it cannot generate the required torque to lift the object. Thus, the robot while lifting the object might fail to reach the goal waypoint and onky reach an intermediate waypoint resulting in discrepancies between modeled and true dynamics.

This is represented as a planning problem in 7D statespace. The first 6 dimensions correspond to the 6DOF pose of the object (or gripper,) and the last dimension corresponds to the redundant DOF in the arm (in our case, it is the upper arm roll joint.) Given a 7D configuration, we use IKFast library [Dia10] to compute the corresponding 7D joint angle configuration. The action space consists of 14 motion primitives that move the arm by a fixed offset in each of the 7

dimensions in positive and negative directions. The discrepancies in this experiment are only in the Z dimension corresponding to lifting the object. For planning, we simply use a kinematic model of the arm and assume that the object being lifted is extremely light. Thus, we do not need to explicitly consider dynamics during planning. However, during execution we take the dynamics into account by executing the motion primitives in the simulator. The cost of any transition is 1 if the object is not at goal pose, 0 if the object is at goal pose. We start the next repetition only if the robot reached the goal pose in the previous repetition.

The 7D state space is discretized into 10 cells in each dimension resulting in 10^7 states. Since the state space is large we use neural network function approximators to maintain the value functions $V, Q, \tilde{V}$. For the state value functions $V, \tilde{V}$ we use the following neural network approximator: a feedforward network with 3 hidden layers consisting of 64 units each, we use ReLU activations after each layer except the last layer, the network takes as input a 34D feature representation of the 7D state computed as follows:

- For any discrete state s, we compute a continuous 10D representation $r(s)$ that is used to construct the features
  - The discrete state is represented as $(xd, yd, zd, rd, pd, yd, rjointd)$ where (xd, yd, zd) represents the 3D discrete location of the object (or gripper,) (rd, pd, yd) represents the discrete roll, pitch, yaw of the object (or gripper,) and $rjointd$ represents the discrete redundant joint angle
  - We convert (xd, yd, zd) to a continuous representation by simply dividing by the grid size in those dimensions, i.e. $(xc, yc, zc) = (xd/10, yd/10, zd/10)$
  - We do a similar construction for $rjointc$, i.e. $rjointc = rjointd/10$
  - However, note that rd, pd, yd are angular dimensions and simply dividing by grid size would not encode the wrap around nature that is inherent in angular dimensions (we did not have this problem for $rjointd$ as the redundant joint angle has lower and upper limits, and is always recorded as a value between those limits.) To account for this, we use a sine-cosine representation defined as $(rc1, rc2, pc1, pc2.yc1, yc2) = (sin(rc), cos(rc), sin(pc), cos(pc), sin(yc), cos(yc))$ where rc, pc, yc are the roll, pitch, yaw angles corresponding to the cell centers of the grid cells rd, pd, yd.
  - Thus, the final 10D representation of state s is given by $r(s) = (xc, yc, zc, rc1, rc2, pc1, pc2, yc1, yc2, rjoi$
  - We also define a truncated 9D representation $r'(s) = (xc, yc, zc, rc1, rc2, pc1, pc2, yc1, yc2)$ and a 3D representation $r''(s) = (xc, yc, zc)$
- The first feature is the 9D relative position of the 6D goal pose w.r.t the object $f1 = r'(g) - r'(s)$
- The second feature is the 10D relative position of the object w.r.t the gripper home state h, $f2 = r(s) - r(h)$
- The third feature is the 9D relative position of the goal w.r.t the gripper home state h, $f3 = r'(g) - r'(h)$
- The fourth feature is the 3D relative position of the obstacle left top corner $o1$ w.r.t the object, $f4 = r''(o1) - r''(s)$
- The fifth and final feature is the 3D relative position of the object right bottom corner $o2$ w.r.t. the object, $f5 = r''(o2) - r''(s)$

- Thus, the final 34D feature representation is given by $f(s) = (f1, f2, f3, f4, f5)$.

The output of the network is a single scalar value representing the cost-to-goal of the input state. Instead of learning the cost-to-goal/value from scratch, we start with an initial value estimate that is hardcoded (manhattan distance to goal in the 7D discrete grid) and the neural network approximator is used to learn a residual on top of it. A similar trick was used in CMAX [Vem+20]. The residual state value function approximator was initialized to output 0 for all $s \in \mathbb{S}$. We use a similar architecture for the residual Q-value function approximator but it takes as input the 34D state feature representation and outputs a vector in $\mathbb{R}^{|\mathbb{A}|}$ (in our case, $\mathbb{R}^{14}$) to represent the cost-to-goal estimate for each action $a \in \mathbb{A}$. We also use the same hardcoded value estimates as before in addition to the residual approximator to construct the Q-values. All baselines and proposed approaches use the same function approximator and same initial hardcoded value estimates to ensure fair comparison. The value function approximators are trained using mean squared loss.

The residual model learning baseline with neural network (NN) function approximator uses the following architecture: 2 hidden layers each with 64 units and all layers are followed by ReLU activations except the last layer. The input of the network is the 34D feature representation of the state and a one-hot encoding of the action in $\mathbb{R}^14$. The output of the network is the 7D continuous state which is added to the state predicted by the model $\hat{M}$. The loss function used to train the network is a simple mean squared loss. The residual model learning baseline with K-Nearest Neighbor regression approximator (KNN) uses a manhattan radius of 3 in the discrete 7D state space. We compute the prediction by averaging the next state residual vector observed in the past for any state that lies within the radius of the current state. The averaged residual is added to the next state predicted by model $\hat{M}$ to obtain the learned next state.

We use Adam optimizer [KB15] with a learning rate of 0.001 and a weight decay (L2 regularization coefficient) of 0.001 to train all the neural network function approximators in all approaches. We use a batch size of 32 for the state value function approximators and a batch size of 128 for the Q-value function approximators. We perform $U = 3$ updates for state value function and $U = 5$ updates for state-action value function for each time step. We update the parameters of all neural network approximators using a polyak averaging coefficient of 0.5.

Finally, we use hindsight experience replay trick [And+17] in training all the value function approximators with the probability of sampling any future state in past trajectories as the goal set to 0.7. This is crucial as our cost function used is extremely sparse.

9.4 Appendix for Chapter 6

9.4.1 General Results

In this section, we will present general results that bound the cost suboptimality of any time-varying controller $\hat{K}$ in terms of the norm differences $||K_t^\star - \hat{K}_t||$. Our first lemma makes use of Assumption 6.2.2 to show that if the norm differences $||K_t^\star - \hat{K}_t||$ are small, then the true system can be stable under $\hat{K}$:

Lemma 9.4.1. *If Assumption 6.2.2 holds and if $\hat{K}$ satisfies $||K_i^\star - \hat{K}_i|| \leq \frac{\delta}{2||B_i||}$ for all $i \in \{0, \cdots, H-1\}$, then we have*

$$||L_t(\hat{K})|| \leq \left(1 - \frac{\delta}{2}\right)^{t+1} \leq e^{-\frac{\delta}{2}(t+1)} \tag{9.16}$$

Proof. Observe that,

$$\begin{aligned}
||L_t(\hat{K})|| &= ||\prod_{i=0}^{t} M_i(\hat{K})|| = ||\prod_{i=0}^{t} A_i + B_i\hat{K}_i|| \\
&= ||\prod_{i=0}^{t} A_i + B_iK_i^\star + B_i(\hat{K}_i - K_i^\star)|| = ||\prod_{i=0}^{t} M_i(K^\star) + \Delta_i||
\end{aligned}$$

where $\Delta_i = B_i(\hat{K}_i - K_i^\star)$. Since the spectral norm is sub-multiplicative we can see that

$$\begin{aligned}
||\prod_{i=0}^{t} M_i(K^\star) + \Delta_i|| &\leq \prod_{i=0}^{t} ||M_i(K^\star) + \Delta_i|| \\
&\leq \prod_{i=0}^{t} (||M_i(K^\star)|| + ||\Delta_i||)
\end{aligned}$$

where we used the triangle inequality. Now note that $||M_i(K^\star)|| \leq 1 - \delta$ from assumption 6.2.2,

$$\begin{aligned}
\|\Delta_i\| = \|B_i(\hat{K}_i - K_i^\star)\| &\leq \|B_i\|\|\hat{K}_i - K_i^\star\| \\
&\leq \kappa\frac{\delta}{2\kappa} \\
&\leq \frac{\delta}{2}
\end{aligned}$$

The last inequality above is from our assumption on model errors in the lemma statement. Combining all of this above, we get

$$||L_t(\hat{K})|| \leq \prod_{i=0}^{t}\left(1 - \delta + \frac{\delta}{2}\right) \leq \left(1 - \frac{\delta}{2}\right)^{t+1}$$

□

The next lemma is very similar to the performance difference lemma that was first proposed in [KL02]. We borrow the version presented in [Faz+18] and extend it to the finite horizon setting below:

Lemma 9.4.2. *Let $\hat{x}_0, \hat{u}_0, \cdots, \hat{x}_H, \hat{u}_H$ be the trajectory generated by controller $\hat{K}$ using the true dynamics such that $\hat{x}_0 = x_0$, $\hat{u}_t = \hat{K}_t \hat{x}_t$ for $t = 0, \cdots, H-1$. Then we have:*

$$\hat{V}_0(x_0) - V_0^\star(x_0) = \sum_{t=0}^{H-1} A_t^\star(\hat{x}_t, \hat{u}_t) - V_H^\star(\hat{x}_H) \tag{9.17}$$

where $\hat{V}_t$ is the cost-to-go using controller $\hat{K}$ from time step t, $V_t^\star$ is the cost-to-go using the optimal controller $K^\star$ from time step t, and $A_t^\star(x, u) = Q_t^\star(x, u) - V_t^\star(x)$ is the advantage of the controller $K^\star$ at time step t. Furthermore, we have that for any x

$$A_t^\star(x, \hat{K}_t x) = x^T(\hat{K}_t - K_t^\star)^T(R + B_t^T P_{t+1}^\star B_t)(\hat{K}_t - K_t^\star)x$$

Proof. For proof, we refer the readers to [Faz+18]. □

We use the performance difference lemma, as stated above, in the finite horizon LQR setup and make use of Lemma 9.4.1 to establish the suboptimality bound in terms of the norm differences $||K_t^\star - \hat{K}_t||$:

Theorem 6.3.1. *Suppose $d \leq n$. Denote $\Gamma = 1 + \max_t\{||A_t||, ||B_t||, ||P_t^\star||, ||K_t^\star||\}$. Then under Assumption 6.2.2 and if $||K_t^\star - \hat{K}_t|| \leq \frac{\delta}{2||B_i||}$ for all $t = 0, \cdots, H-1$, we have*

$$\hat{V}_0(x_0) - V_0^\star(x_0) \leq d\Gamma^3 \|x_0\|^2 \sum_{t=0}^{H-1} e^{-\delta t} \|K_t^\star - \hat{K}_t\|^2 \tag{6.3}$$

Proof. From Lemma 9.4.2 we have

$$A_t(\hat{x}_t, \hat{K}_t \hat{x}_t) = \hat{x}_t^T(\hat{K}_t - K_t^\star)^T(R + B_t^T P_{t+1} B_t)(\hat{K}_t - K_t^\star)\hat{x}_t$$

We know $\hat{x}_t = L_{t-1}(\hat{K})x_0$ and using the trace identity we get

$$\begin{aligned} A_t(\hat{x}_t, \hat{K}_t \hat{x}_t) &= \mathrm{Tr}(L_{t-1}(\hat{K})x_0 x_0^T (L_{t-1}(\hat{K}))^T (\hat{K}_t - K_t^\star)^T (R + B_t^T P_{t+1} B_t)(\hat{K}_t - K_t^\star)) \\ &\leq \|L_{t-1}(\hat{K})x_0 x_0^T\| \|R + B_t^T P_{t+1} B_t\| \|\hat{K}_t - K_t^\star\|_F^2 \\ &\leq \|L_{t-1}(\hat{K})\|^2 \|x_0\|^2 \|R + B_t^T P_{t+1} B_t\| \|\hat{K}_t - K_t^\star\|_F^2 \end{aligned}$$

We can bound $\|R + B_t^T P_{t+1} B_t\| \leq \Gamma^3$ and $\|\hat{K}_t - K_t^\star\|_F^2 \leq \min\{n, d\}\|\hat{K}_t - K_t^\star\|^2$. We can also use Lemma 9.4.1 to bound $\|L_{t-1}(\hat{K})\| \leq \exp\left(-\frac{\delta}{2}t\right)$. Combining all of this above we get

$$A_t(\hat{x}_t, \hat{K}_t \hat{x}_t) \leq \min\{n, d\}\Gamma^3 \exp(-\delta t) \|\hat{K}_t - K_t^\star\|^2 \|x_0\|^2$$

Summing over all time steps we obtain (using $d \leq n$)

$$\hat{V}_0(x_0) - V_0(x_0) \leq d\Gamma^3 \|x_0\|^2 \sum_{t=0}^{H-1} \exp(-\delta t) \|\hat{K}_t - K_t^\star\|^2$$

□

9.4.2 Helpful Lemmas

Before we dive into the results, let us present a helpful lemma borrowed from [MTR19]:

Lemma 9.4.3. *Let f_1, f_2 be μ-strongly convex twice differentiable functions. Let $x_1 = \arg\min_x f_1(x)$ and $x_2 = \arg\min_x f_2(x)$. Suppose $||\nabla f_1(x_2)|| \leq \epsilon$, then $||x_1 - x_2|| \leq \frac{\epsilon}{\mu}$*

Proof. Taylor expanding ∇f_1 we get

$$\begin{aligned}\nabla f_1(x_2) &= \nabla f_1(x_1) + \nabla^2 f_1(\tilde{x})(x_2 - x_1) \\ &= \nabla^2 f_1(\tilde{x})(x_2 - x_1)\end{aligned}$$

for some $\tilde{x} = tx_1 + (1-t)x_2$ where $t \in [0, 1]$. Thus we have

$$\|\nabla f_1(x_2)\| = \|\nabla^2 f_1(\tilde{x})(x_2 - x_1)\| \leq \epsilon$$

But we know $\|\nabla^2 f_1(\tilde{x})\| \geq \mu$ which gives us

$$\|x_2 - x_1\| \leq \frac{\epsilon}{\mu}$$

□

The next lemma is a useful fact about positive semi-definite matrices, also from [MTR19],

Lemma 9.4.4. *Given matrices $A, \hat{A}$ such that $\|A - \hat{A}\| \leq \epsilon_A$, and positive-semidefinite matrices $Q, S, \hat{S}$ we have*

$$\|A^T Q(I + SQ)^{-1} A - \hat{A}^T Q(I + \hat{S}Q)^{-1} \hat{A}\| \leq \|A\|^2 \|Q\|^2 \|\hat{S} - S\| + 2\|A\|\|Q\|\epsilon_A + \|Q\|\epsilon_A^2 \quad (9.18)$$

Proof. We can rewrite the expression,

$$\begin{aligned}A^T Q(I + SQ)^{-1} A - \hat{A}^T Q(I + \hat{S}Q)^{-1} \hat{A} = \\ & A^T Q(I + SQ)^{-1}(\hat{S} - S)Q(I + \hat{S}Q)^{-1} A - A^T Q(I + \hat{S}Q)^{-1}(\hat{A} - A) \\ & - (\hat{A} - A)^T Q(I + \hat{S}Q)^{-1} A - (\hat{A} - A)^T Q(I + \hat{S}Q)^{-1}(\hat{A} - A)\end{aligned}$$

Now we make use of Lemma 7 from [MTR19] which states that for any two positive semidefinite matrices M, N of the same dimension, we have $\|N(I + MN)^{-1}\| \leq \|N\|$. Thus, we have $\|Q(I + SQ)^{-1}\| \leq \|Q\|$ and $\|Q(I + \hat{S}Q)^{-1}\| \leq \|Q\|$.

Using the above facts we get,

$$\|A^T Q(I + SQ)^{-1} A - \hat{A}^T Q(I + \hat{S}Q)^{-1} \hat{A}\| \leq \|A\|^2 \|Q\|^2 \|\hat{S} - S\| + 2\|A\|\|Q\|\epsilon_A + \|Q\|\epsilon_A^2$$

□

Finally, we have a lemma that will be useful in proving ricatti perturbation bounds,

Lemma 9.4.5. *Given positive semidefinite matrices N_1, N_2, M of the same dimensions, we have*

$$||N_1(I + MN_1)^{-1} - N_2(I + MN_2)^{-1}|| \leq ||(I + MN_1)^{-1}||||N_1 - N_2||||(I + MN_2)^{-1}|| \quad (9.19)$$

Proof. We can rewrite the expression as,

$$\begin{aligned}
&N_1(I+MN_1)^{-1} - N_2(I+MN_2)^{-1} \\
&= \left[N_1(I+MN_1)^{-1} - N_1(I+MN_2)^{-1}\right] + \left[N_1(I+MN_2)^{-1} - N_2(I+MN_2)^{-1}\right] \\
&= N_1(I+MN_1)^{-1}\left[(I+MN_2)-(I+MN_1)\right](I+MN_2)^{-1} + (N_1-N_2)(I+MN_2)^{-1} \\
&= N_1(I+MN_1)^{-1}M(N_2-N_1)(I+MN_2)^{-1} + (N_1-N_2)(I+MN_2)^{-1} \\
&= \left[I - N_1(I+MN_1)^{-1}M\right](N_1-N_2)(I+MN_2)^{-1} \\
&= (I+N_1M)^{-1}(N_1-N_2)(I+MN_2)^{-1}
\end{aligned}$$

The rest follows by taking norm on both sides, and using the submultiplicative property of the induced norm.

$$\begin{aligned}
||N_1(I+MN_1)^{-1} - N_2(I+MN_2)^{-1}|| &\leq ||(I+N_1M)^{-1}||||N_1-N_2||||(I+MN_2)^{-1}|| \\
&= ||(I+N_1M)^{-T}||||N_1-N_2||||(I+MN_2)^{-1}|| \\
&= ||(I+MN_1)^{-1}||||N_1-N_2||||(I+MN_2)^{-1}||
\end{aligned}$$

□

9.4.3 Optimal Control with Misspecified Model Results

The next lemma, from [MTR19], applies the above result to quadratic functions that are observed in linear quadratic control:

Lemma 9.4.6. *Define* $f_1(x,u) = \frac{1}{2}u^TRu + \frac{1}{2}(A_1x+B_1u)^TP_1(A_1x+B_1u)$ *and similarly define* $f_2(x,u)$ *where* R, P_1, P_2 *are positive-definite matrices. Let* K_1 *be such that* $u_1 = \arg\min_u f_1(x,u) = K_1x$ *for any vector* x*. Define the matrix* K_2 *in a similar fashion. Also, denote* $\Gamma = 1+\max\{||A_1||, ||B_1||, ||P_1||, ||K_1||\}$*. Suppose there exists* $\epsilon_A, \epsilon_B, \epsilon_P > 0$ *(and* $< \Gamma$*) such that* $||A_1 - A_2|| \leq \epsilon_A$*,* $||B_1 - B_2|| \leq \epsilon_B$*, and* $||P_1 - P_2|| \leq \epsilon_P$*. Then we have,*

$$\|K_1 - K_2\| \leq \frac{\Gamma^2\epsilon_A + (3\Gamma^3 + 2\Gamma^2)\epsilon_B + 4(\Gamma^3+\Gamma^2)\epsilon_P}{\underline{\sigma}(R)} \tag{9.20}$$

Proof. Consider

$$\begin{aligned}
\nabla_u f_1(x,u) &= (B_1^TP_1B_1 + R)u + B_1^TP_1A_1x \\
\nabla_u f_2(x,u) &= (B_2^TP_2B_2 + R)u + B_2^TP_2A_2x
\end{aligned}$$

Let us bound the difference $\|\nabla_u f_1(x,u) - \nabla_u f_2(x,u)\|$ by bounding each term separately. First consider the term

$$\begin{aligned}
\|B_1^TP_1B_1 - B_2^TP_2B_2\| &= \|B_1^TP_1(B_1-B_2) + (B_1-B_2)^TP_1B_2 + B_2^T(P_1-P_2)B_2\| \\
&\leq \|B_1^TP_1(B_1-B_2)\| + \|(B_1-B_2)^TP_1B_2\| + \|B_2^T(P_1-P_2)B_2\| \\
&\leq \Gamma^2\epsilon_B + \Gamma\epsilon_B(\Gamma+\epsilon_B) + (\Gamma+\epsilon_B)^2\epsilon_P \\
&\leq \Gamma^2(3\epsilon_B + 4\epsilon_P)
\end{aligned}$$

where we used the fact that $\|B_2\| \leq \Gamma + \epsilon_B$. We can similarly bound the term

$$\|B_1^TP_1A_1 - B_2^TP_2A_2\| \leq \Gamma^2(\epsilon_A + 2\epsilon_B + 4\epsilon_P)$$

Thus, we have for any vector x such that $\|x\| \leq 1$

$$\|\nabla_u f_1(x,u) - \nabla_u f_2(x,u)\| \leq \Gamma^2(3\epsilon_B + 4\epsilon_P)\|u\| + \Gamma^2(\epsilon_A + 2\epsilon_B + 4\epsilon_P)$$

Substituting $u = u_1$ we get

$$\|\nabla_u f_2(x,u_1)\| \leq \Gamma^2(3\epsilon_B + 4\epsilon_P)\|u_1\| + \Gamma^2(\epsilon_A + 2\epsilon_B + 4\epsilon_P)$$

We can bound $\|u_1\| \leq \|K_1\|\|x\| \leq \|K_1\| \leq \Gamma$. Then from Lemma 9.4.3 we have,

$$\|u_1 - u_2\| \leq \frac{\Gamma^3(3\epsilon_B + 4\epsilon_P) + \Gamma^2(\epsilon_A + 2\epsilon_B + 4\epsilon_P)}{\underline{\sigma}(R)}$$
$$\|K_1 - K_2\| \leq \frac{\Gamma^2\epsilon_A + (3\Gamma^3 + 2\Gamma^2)\epsilon_B + 4(\Gamma^3 + \Gamma^2)\epsilon_P}{\underline{\sigma}(R)}$$

□

Now we will prove Lemma 6.3.1,

Lemma 6.3.1. *If $||A_t - \hat{A}_t|| \leq \epsilon_A$ and $||B_t - \hat{B}_t|| \leq \epsilon_B$ for $t = 0, \cdots, H-1$, and we have $||P^\star_{t+1} - P^{\mathsf{MM}}_{t+1}|| \leq f^{\mathsf{MM}}_{t+1}(\epsilon_A, \epsilon_B)$ for some function f^{MM}_{t+1}. Then we have under Assumption 6.2.1 for all $t = 0, \cdots, H-1$,*

$$||K^\star_t - K^{\mathsf{MM}}_t|| \leq 14\Gamma^3\epsilon_t \tag{6.4}$$

where $\Gamma = 1 + \max_t\{||A_t||, ||B_t||, ||P^\star_t||, ||K^\star_t||\}$ and $\epsilon_t = \max\{\epsilon_A, \epsilon_B, f^{\mathsf{MM}}_{t+1}(\epsilon_A, \epsilon_B)\}$.

Proof. Use Assumption 6.2.1 and Lemma 9.4.6 for every $t = 0, \cdots, H-1$ with $\epsilon_P = f^{\mathsf{MM}}_{t+1}(\epsilon_A, \epsilon_B)$ and choosing $\epsilon_t = \max\{\epsilon_A, \epsilon_B, f^{\mathsf{MM}}_{t+1}(\epsilon_A, \epsilon_B)\}$. □

All that is left is to prove Theorem 6.3.2 which we will do now,

Theorem 6.3.2. *If the cost-to-go matrices for the optimal controller and MM controller are specified by $\{P^\star_t\}$ and $\{P^{\mathsf{MM}}_t\}$ such that $P^\star_H = P^{\mathsf{MM}}_H = Q_f$ then,*

$$\begin{aligned} ||P^\star_t - P^{\mathsf{MM}}_t|| \leq\ & \|A_t\|^2\|P^\star_{t+1}\|^2(2\|B_t\|\|R^{-1}\|\epsilon_B + \|R^{-1}\|\epsilon_B^2) \\ & + 2\|A_t\|\|P^\star_{t+1}\|\epsilon_A + \|P^\star_{t+1}\|\epsilon_A^2 \\ & + c_{P^\star_{t+1}}(\|A_t\| + \epsilon_A)^2||P^\star_{t+1} - P^{\mathsf{MM}}_{t+1}|| \end{aligned} \tag{6.5}$$

for $t = 0, \cdots, H-1$ where $c_{P^\star_{t+1}} \in \mathbb{R}^+$ is a constant that is dependent only on $P^\star_{t+1}$ if ϵ_A, ϵ_B are small enough such that $\|P^\star_{t+1} - P^{\mathsf{MM}}_{t+1}\| \leq \|P^\star_{t+1}\|^{-1}$. Furthermore, the upper bound (6.5) is tight up to constants that only depend on the true dynamics A_t, B_t, cost matrix R, and $P^\star_{t+1}$.

Proof. We know $P^\star_t$ satisfies,

$$\begin{aligned} P^\star_t &= Q + A_t^T P^\star_{t+1} A_t - A_t^T P^\star_{t+1} B_t(R + B_t^T P^\star_{t+1} B_t)^{-1} B_t^T P^\star_{t+1} A_t \\ &= Q + A_t^T P^\star_{t+1}(I + B_t R^{-1} B_t^T P^\star_{t+1})^{-1} A_t \end{aligned}$$

where we used the matrix inversion lemma.

Similarly we have,

$$P^{\mathsf{MM}}_t = Q + \hat{A}_t^T P^{\mathsf{MM}}_{t+1}(I + \hat{B}_t R^{-1} \hat{B}_t^T P^{\mathsf{MM}}_{t+1})^{-1} \hat{A}_t$$

Consider the difference,

$$\begin{aligned}
P_t^\star - P_t^{\mathsf{MM}} &= A_t^T P_{t+1}^\star (I + B_t R^{-1} B_t^T P_{t+1}^\star)^{-1} A_t - \hat{A}_t^T P_{t+1}^{\mathsf{MM}} (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{MM}})^{-1} \hat{A}_t \\
&= A_t^T P_{t+1}^\star (I + B_t R^{-1} B_t^T P_{t+1}^\star)^{-1} A_t - \hat{A}_t^T P_{t+1}^\star (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1} \hat{A}_t \\
&+ \hat{A}_t^T \left(P_{t+1}^\star (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1} - P_{t+1}^{\mathsf{MM}} (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{MM}})^{-1} \right) \hat{A}_t
\end{aligned}$$

To bound the above expression, we will make use of Lemma 9.4.4 with $S = B_t R^{-1} B_t^T$, $\hat{S} = \hat{B}_t R^{-1} \hat{B}_t^T$, $Q = P_{t+1}^\star$ and observing that $\|\hat{S} - S\| \leq 2\|B_t\|\|R^{-1}\|\epsilon_B + \|R^{-1}\|\epsilon_B^2$ we obtain

$$\begin{aligned}
\|P_t^{\mathsf{MM}} - P_t^\star\| \leq & \|A_t\|^2 \|P_{t+1}^\star\|^2 (2\|B_t\|\|R^{-1}\|\epsilon_B + \|R^{-1}\|\epsilon_B^2) + 2\|A_t\|\|P_{t+1}^\star\|\epsilon_A + \|P_{t+1}^\star\|\epsilon_A^2 \\
& + \|\hat{A}_t^T \left(P_{t+1}^\star (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1} - P_{t+1}^{\mathsf{MM}} (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{MM}})^{-1} \right) \hat{A}_t\|
\end{aligned}$$

All that remains is to bound the second expression. We will use Lemma 9.4.5 with $N_1 = P_{t+1}^\star$, $N_2 = P_{t+1}^{\mathsf{MM}}$ and $M = \hat{B}_t R^{-1} \hat{B}_t^T$ gives us,

$$\begin{aligned}
& ||P_{t+1}^\star (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1} - P_{t+1}^{\mathsf{MM}} (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{MM}})^{-1}|| \\
& \leq ||(I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1}||||P_{t+1}^\star - P_{t+1}^{\mathsf{MM}}||||(I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{MM}})^{-1}||
\end{aligned}$$

Thus, we have

$$\begin{aligned}
||P_t^\star - P_t^{\mathsf{MM}}|| \leq & \|A_t\|^2 \|P_{t+1}^\star\|^2 (2\|B_t\|\|R^{-1}\|\epsilon_B + \|R^{-1}\|\epsilon_B^2) + 2\|A_t\|\|P_{t+1}^\star\|\epsilon_A + \|P_{t+1}^\star\|\epsilon_A^2 \\
& + \|\hat{A}_t\|^2 ||(I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1}||||P_{t+1}^\star - P_{t+1}^{\mathsf{MM}}||||(I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{MM}})^{-1}||
\end{aligned}$$

Observe that we can bound

$$\begin{aligned}
||(I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1}|| &= ||(P_{t+1}^\star)^{-1} P_{t+1}^\star (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1}|| \\
&\leq ||(P_{t+1}^\star)^{-1}||||P_{t+1}^\star (I + \hat{B}_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1}|| \\
&\leq ||(P_{t+1}^\star)^{-1}||||P_{t+1}^\star|| = \kappa_{P_{t+1}^\star}
\end{aligned}$$

where $\kappa_{P_{t+1}^\star}$ is the condition number of the matrix $P_{t+1}^\star$. This gives us the bound

$$\begin{aligned}
||P_t^\star - P_t^{\mathsf{MM}}|| \leq & \|A_t\|^2 \|P_{t+1}^\star\|^2 (2\|B_t\|\|R^{-1}\|\epsilon_B + \|R^{-1}\|\epsilon_B^2) + 2\|A_t\|\|P_{t+1}^\star\|\epsilon_A + \|P_{t+1}^\star\|\epsilon_A^2 \\
& + \|\hat{A}_t\|^2 \kappa_{P_{t+1}^\star} \kappa_{P_{t+1}^{\mathsf{MM}}} ||P_{t+1}^\star - P_{t+1}^{\mathsf{MM}}||
\end{aligned}$$

Using the fact that $||\hat{A}_t||^2 \leq (\|A_t\| + \epsilon_A)^2$ gives us

$$\begin{aligned}
||P_t^\star - P_t^{\mathsf{MM}}|| \leq & \|A_t\|^2 \|P_{t+1}^\star\|^2 (2\|B_t\|\|R^{-1}\|\epsilon_B + \|R^{-1}\|\epsilon_B^2) + 2\|A_t\|\|P_{t+1}^\star\|\epsilon_A + \|P_{t+1}^\star\|\epsilon_A^2 \\
& + (\|A_t\| + \epsilon_A)^2 \kappa_{P_{t+1}^\star} \kappa_{P_{t+1}^{\mathsf{MM}}} ||P_{t+1}^\star - P_{t+1}^{\mathsf{MM}}|| \qquad (9.21)
\end{aligned}$$

If ϵ_A, ϵ_B are small enough that $\|P_{t+1}^\star - P_{t+1}^{\mathsf{MM}}\|\|P_{t+1}^\star\| \leq 1$ then we can bound

$$\begin{aligned}
\|(P_{t+1}^{\mathsf{MM}})^{-1} - (P_{t+1}^\star)^{-1}\| &\leq \frac{\|(P_{t+1}^\star)^{-1}\|}{1 - \|(P_{t+1}^\star)^{-1}\|\|P_{t+1}^\star - P_{t+1}^{\mathsf{MM}}\|} \\
&\leq \frac{\|(P_{t+1}^\star)^{-1}\|\|P_{t+1}^\star\|}{\|P_{t+1}^\star\| - \|(P_{t+1}^\star)^{-1}\|} \\
&\leq \frac{\kappa_{P_{t+1}^\star}}{\|P_{t+1}^\star\|^2 - \kappa_{P_{t+1}^\star}}
\end{aligned}$$

The above result is from [HJ12] (Section 5.8 page 381). Now we can bound the condition number $\kappa_{P_{t+1}^{\mathsf{MM}}}$ by observing that $\|(P_{t+1}^{\mathsf{MM}})^{-1}\| \leq \|(P_{t+1}^{\star})^{-1}\| + \frac{\kappa_{P_{t+1}^{\star}}}{\|P_{t+1}^{\star}\|^2 - \kappa_{P_{t+1}^{\star}}}$ and $\|P_{t+1}^{\mathsf{MM}}\| \leq \|P_{t+1}^{\star}\| + \|P_{t+1}^{\mathsf{MM}} - P_{t+1}^{\star}\| \leq \|P_{t+1}^{\star}\| + \frac{1}{\|P_{t+1}^{\star}\|}$ giving us

$$\begin{aligned}
\kappa_{P_{t+1}^{\star}}\kappa_{P_{t+1}^{\mathsf{MM}}} &= \kappa_{P_{t+1}^{\star}}\|(P_{t+1}^{\mathsf{MM}})^{-1}\|\|P_{t+1}^{\mathsf{MM}}\| \leq \kappa_{P_{t+1}^{\star}}(\|(P_{t+1}^{\star})^{-1}\| + \frac{\kappa_{P_{t+1}^{\star}}}{\|P_{t+1}^{\star}\|^2 - \kappa_{P_{t+1}^{\star}}})(\|P_{t+1}^{\star}\| + \frac{1}{\|P_{t+1}^{\star}\|}) \\
&= \kappa_{P_{t+1}^{\star}}^2 + \frac{\kappa_{P_{t+1}^{\star}}^2}{\|P_{t+1}^{\star}\|^2} + \frac{\kappa_{P_{t+1}^{\star}}^2}{\|P_{t+1}^{\star}\|^2 - \kappa_{P_{t+1}^{\star}}}(\|P_{t+1}^{\star}\| + \frac{1}{\|P_{t+1}^{\star}\|})
\end{aligned}$$

Denoting $c_{P_{t+1}^{\star}}$ as the right hand side expression in the above inequality we get the desired result.The example that realizes the upper bound is given in Appendix 9.4.6.

□

9.4.4 Note on Assumption 6.2.3

Consider the cost-to-go matrix P_t^{ILC} given by

$$\begin{aligned}
P_t^{\mathsf{ILC}} &= Q + \hat{A}_t^T P_{t+1}^{\mathsf{ILC}} A_t - \hat{A}_t^T P_{t+1}^{\mathsf{ILC}} B_t (R + \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} B_t)^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} A_t \\
&= Q + \hat{A}_t^T P_{t+1}^{\mathsf{ILC}} (I + B_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}})^{-1} A_t
\end{aligned}$$

and the cost-to-go from any state x is given by

$$V_t(x) = x^T P_t^{\mathsf{ILC}} x$$

Since this is a quadratic, for it to be convex (and thus, have a minima) we require the leading coefficient to be positive semi-definite. In other words, P_t^{ILC} should have eigenvalues with non-negative real parts. Assuming P_{t+1}^{ILC} to be positive semi-definite, and observing the fact that Q is a positive semi-definite matrix, we require that $B_t R^{-1} \hat{B}_t^T$ to have eigenvalues with non-negative real parts for P_t^{ILC} to be positive semi-definite. Note that this is trivially satisfied for MM as the leading coefficient there contains a similar term $B_t R^{-1} B_t^T$ which is positive semi-definite.

Intuitively, if $B_t R^{-1} \hat{B}_t^T$ does not have eigenvalues with non-negative real parts, then the resulting quadratic cost-to-go function need not be convex, and ILC will not converge.

9.4.5 Iterative Learning Control Results

Our first lemma derives a similar result as Lemma 9.4.6 but for the iterative learning control setting,

Lemma 9.4.7. *Given functions $f_1(x,u)$ and $f_2(x,u)$ such that $\nabla_u f_1(x,u) = (B_1^T P_1 B_1 + R)u + B_1^T P_1 A_1 x$ and $\nabla_u f_2(x,u) = (B_2^T P_2 B_1 + R)u + B_2^T P_2 A_1 x$ where R, P_1, P_2 are positive-definite matrices. Let K_1 and K_2 be unique matrices such that $\nabla_u f_1(x, K_1 x) = 0$ and $\nabla_u f_2(x, K_2 x) = 0$ for any vector x. Also, denote $\Gamma = 1 + \max\{||A_1||, ||B_1||, ||P_1||, ||K_1||\}$. Suppose there exists $\epsilon_A, \epsilon_B, \epsilon_P > 0$ (and $< \Gamma$) such that $||A_1 - A_2|| \leq \epsilon_A$, and $||B_1 - B_2|| \leq \epsilon_B$, and $||P_1 - P_2|| \leq \epsilon_P$. Then we have,*

$$\|K_1 - K_2\| \leq \frac{2\Gamma^3(\epsilon_B + 2\epsilon_P)}{\underline{\sigma}(R)} \tag{9.22}$$

Proof. Let us bound the difference $\|\nabla_u f_1(x,u) - \nabla_u f_2(x,u)\|$ by bounding each term separately. First consider the term

$$\begin{aligned}\|B_1^T P_1 B_1 - B_2^T P_2 B_1\| &= \|(B_1 - B_2)^T P_1 B_1 + B_2^T (P_1 - P_2) B_1\| \\ &\leq \Gamma^2 \epsilon_B + \Gamma(\Gamma + \epsilon_B)\epsilon_P \\ &\leq \Gamma^2(\epsilon_B + 2\epsilon_P)\end{aligned}$$

where we used the fact that $\|B_2\| \leq \Gamma + \epsilon_B$. We can similarly bound the term

$$\|B_1^T P_1 A_1 - B_2^T P_2 A_1\| \leq \Gamma^2(\epsilon_B + 2\epsilon_P)$$

Thus, we have for any vector x such that $\|x\| \leq 1$

$$\|\nabla_u f_1(x,u) - \nabla_u f_2(x,u)\| \leq \Gamma^2(\epsilon_B + 2\epsilon_P)(\|u\| + 1)$$

Substituting $u = u_1$ we get

$$\|\nabla_u f_2(x,u_1)\| \leq \Gamma^2(\epsilon_B + 2\epsilon_P)(\|u_1\| + 1)$$

We can bound $\|u_1\| \leq \|K_1\|\|x\| \leq \|K_1\| \leq \Gamma$. Then from Lemma 9.4.3 we have,

$$\begin{aligned}\|u_1 - u_2\| &\leq \frac{\Gamma^2(\epsilon_B + 2\epsilon_P)(\Gamma + 1)}{\underline{\sigma}(R)} \\ \|K_1 - K_2\| &\leq \frac{2\Gamma^3(\epsilon_B + 2\epsilon_P)}{\underline{\sigma}(R)}\end{aligned}$$

□

Now we will prove Lemma 6.3.2,

Lemma 6.3.2. *If $||A_t - \hat{A}_t|| \leq \epsilon_A$ and $||B_t - \hat{B}_t|| \leq \epsilon_B$ for $t = 0, \cdots, H-1$, and we have $||P_{t+1} - P_{t+1}^{\mathsf{ILC}}|| \leq f_{t+1}^{\mathsf{ILC}}(\epsilon_A, \epsilon_B)$ for some function f_{t+1}^{ILC}. Then we have under Assumption 6.2.1 for all $t = 0, \cdots, H-1$,*

$$||K_t^\star - K_t^{\mathsf{ILC}}|| \leq 6\Gamma^3 \epsilon_t \tag{6.6}$$

where $\Gamma = 1 + \max_t\{||A_t||, ||B_t||, ||P_t^\star||, ||K_t^\star||\}$ and $\epsilon_t = \max\{\epsilon_A, \epsilon_B, f_{t+1}^{\mathsf{ILC}}(\epsilon_A, \epsilon_B)\}$.

Proof. Use Assumption 6.2.1 and Lemma 9.4.7 for $t = 0, \cdots, H-1$ with $\epsilon_P = f_{t+1}^{\mathsf{ILC}}(\epsilon_A, \epsilon_B)$ and choosing $\epsilon_t = \max\{\epsilon_A, \epsilon_B, f_{t+1}^{\mathsf{ILC}}(\epsilon_A, \epsilon_B)\}$. □

Our final task is to prove Theorem 6.3.3,

Theorem 6.3.3. *If the cost-to-go matrices for the optimal controller and iterative learning control are specified by $\{P_t^\star\}$ and $\{P_t^{\mathsf{ILC}}\}$ such that $P_H^\star = P_H^{\mathsf{ILC}} = Q_f$ then we have under Assumption 6.2.3,*

$$\begin{aligned}||P_t^\star - P_t^{\mathsf{ILC}}|| \leq \|A_t\|^2 \|P_{t+1}^\star\|^2 \|B_t\| \|R^{-1}\| \epsilon_B + \|A_t\| \|P_{t+1}^\star\| \epsilon_A \\ + c_{P_{t+1}^\star} ||A_t|| (\|A_t\| + \epsilon_A) ||P_{t+1}^\star - P_{t+1}^{\mathsf{ILC}}||\end{aligned} \tag{6.7}$$

for $t = 0, \cdots, H-1$ where $c_{P_{t+1}^\star} \in \mathbb{R}^+$ is a constant that is dependent only on $P_{t+1}^\star$ if ϵ_A, ϵ_B are small enough that $\|P_{t+1}^\star - P_{t+1}^{\mathsf{ILC}}\| \leq \|P_{t+1}^\star\|^{-1}$. Furthermore, the upper bound (6.7) is tight upto constants that depend only on the true dynamics A_t, B_t, cost matrix R, and $P_{t+1}^\star$.

Proof. We know P_t^{ILC} satisfies,

$$\begin{aligned} P_t^{\mathsf{ILC}} &= Q + \hat{A}_t^T P_{t+1}^{\mathsf{ILC}} A_t - \hat{A}_t^T P_{t+1}^{\mathsf{ILC}} B_t (R + \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} B_t)^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}} A_t \\ &= Q + \hat{A}_t^T P_{t+1}^{\mathsf{ILC}} (I + B_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}})^{-1} A_t \end{aligned}$$

where we used the matrix inversion lemma.

Consider the difference,

$$\begin{aligned} P_t^\star - P_t^{\mathsf{ILC}} &= A_t^T P_{t+1}^\star (I + B_t R^{-1} B_t^T P_{t+1}^\star)^{-1} A_t - \hat{A}_t^T P_{t+1}^{\mathsf{ILC}} (I + B_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}})^{-1} A_t \\ &= A^T P_{t+1}^\star (I + B_t R^{-1} B_t^T P_{t+1}^\star)^{-1} A_t - \hat{A}_t^T P_{t+1}^\star (I + B_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1} A_t \\ &+ \hat{A}_t^T \left(P_{t+1}^\star (I + B_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1} - P_{t+1}^{\mathsf{ILC}} (I + B_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}})^{-1} \right) A_t \end{aligned}$$

Here again we can use Lemma 9.4.4 with $S = B_t R^{-1} B_t^T$, $\hat{S} = B_t R^{-1} \hat{B}_t^T$, $Q = P_{t+1}^\star$ and observing that $||\hat{S} - S|| \leq ||B_t||||R^{-1}||\epsilon_B$ to get

$$\begin{aligned} ||P_t^{\mathsf{ILC}} - P_t^\star|| &\leq ||A_t||^2 ||P_{t+1}^\star||^2 ||B_t||||R^{-1}||\epsilon_B + ||A_t||||P_{t+1}^\star||\epsilon_A \\ &+ ||A_t||||\hat{A}_t||||P_{t+1}^\star (I + B_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1} - P_{t+1}^{\mathsf{ILC}} (I + B_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}})^{-1}|| \end{aligned}$$

Here again we use Lemma 9.4.5 to bound the second expression giving us

$$\begin{aligned} ||P_t^{\mathsf{ILC}} - P_t^\star|| &\leq ||A_t||^2 ||P_{t+1}^\star||^2 ||B_t||||R^{-1}||\epsilon_B + ||A_t||||P_{t+1}^\star||\epsilon_A \\ &+ ||A_t||||\hat{A}_t||||(I + B_t R^{-1} \hat{B}_t^T P_{t+1}^\star)^{-1}||||P_{t+1}^{\mathsf{ILC}} - P_{t+1}^\star||||(I + B_t R^{-1} \hat{B}_t^T P_{t+1}^{\mathsf{ILC}})^{-1}|| \end{aligned}$$

This can be rewritten as the final bound,

$$\begin{aligned} ||P_t^\star - P_t^{\mathsf{ILC}}|| &\leq ||A_t||^2 ||P_{t+1}^\star||^2 ||B_t||||R^{-1}||\epsilon_B + ||A_t||||P_{t+1}^\star||\epsilon_A \\ &+ (||A_t||^2 + \epsilon_A ||A_t||) \kappa_{P_{t+1}^\star} \kappa_{P_{t+1}^{\mathsf{ILC}}} ||P_{t+1}^\star - P_{t+1}^{\mathsf{ILC}}|| \end{aligned} \tag{9.23}$$

The constant $c_{P_{t+1}^\star}$ can be derived very similarly as we have done in the proof of Theorem 6.3.2. The example that realizes the upper bound is given in Appendix 9.4.6. □

9.4.6 Scalar Example that Realizes Upper Bounds

General Formulation

Consider a 1D linear dynamical system given by,

$$x_t = a x_t + b u_t \tag{9.24}$$

where $x_t, u_t, a, b \in \mathbb{R}$. The cost function is given by,

$$V_0(x_0) = \sum_{t=0}^{H-1} q x_t^2 + r u_t^2 + q x_H^2 \tag{9.25}$$

We are given access to an approximate model specified using $\hat{a}, \hat{b} \in \mathbb{R}$.

The optimal cost-to-go is specified using

$$p_H^\star = q \tag{9.26}$$

$$p_t^\star = q + \frac{a^2 p_{t+1}^\star}{1 + b^2 r^{-1} p_{t+1}^\star} = q + \frac{a^2 r p_{t+1}^\star}{r + b^2 p_{t+1}^\star} \tag{9.27}$$

For MM, the cost-to-go is specified using

$$p_H^{\mathsf{MM}} = q \tag{9.28}$$

$$p_t^{\mathsf{MM}} = q + \frac{\hat{a}^2 r p_{t+1}^{\mathsf{MM}}}{r + \hat{b}^2 p_{t+1}^{\mathsf{MM}}} \tag{9.29}$$

For ILC, the cost-to-go is specified using

$$p_h^{\mathsf{ILC}} = q \tag{9.30}$$

$$p_t^{\mathsf{ILC}} = q + \frac{a\hat{a} r p_{t+1}^{\mathsf{ILC}}}{r + b\hat{b} p_{t+1}^{\mathsf{ILC}}} \tag{9.31}$$

In the next two subsections, we will show that an example dynamical system where $\hat{b} = 0$, i.e. the approximate model thinks that the system is not controllable will realize the worst case upper bounds for both MM and ILC as presented in Theorems 6.3.2 and 6.3.3 respectively.

Optimal Control with Misspecified Model

Consider the difference

$$\begin{aligned} p_t^\star - p_t^{\mathsf{MM}} &= \frac{a^2 r p_{t+1}^\star}{r + b^2 p_{t+1}^\star} - \frac{\hat{a}^2 r p_{t+1}^{\mathsf{MM}}}{r + \hat{b}^2 p_{t+1}^{\mathsf{MM}}} \\ &= \left(\frac{a^2 r p_{t+1}^\star}{r + b^2 p_{t+1}^\star} - \frac{\hat{a}^2 r p_{t+1}^\star}{r + \hat{b}^2 p_{t+1}^\star} \right) + \left(\frac{\hat{a}^2 r p_{t+1}^\star}{r + \hat{b}^2 p_{t+1}^\star} - \frac{\hat{a}^2 r p_{t+1}^{\mathsf{MM}}}{r + \hat{b}^2 p_{t+1}^{\mathsf{MM}}} \right) \end{aligned}$$

Let us look at each term separately. The first term can be simplified as

$$\left(\frac{a^2 r p_{t+1}^\star}{r + b^2 p_{t+1}^\star} - \frac{\hat{a}^2 r p_{t+1}^\star}{r + \hat{b}^2 p_{t+1}^\star} \right) = \frac{p_{t+1}^\star (a^2 - \hat{a}^2)}{1 + \hat{b}^2 r^{-1} p_{t+1}^\star} + \frac{a^2 r^{-1} (\hat{b}^2 - b^2)(p_{t+1}^\star)^2}{(1 + b^2 r^{-1} p_{t+1}^\star)(1 + \hat{b}^2 r^{-1} p_{t+1}^\star)} \tag{9.32}$$

Similarly, the second term can be simplified as

$$\left(\frac{\hat{a}^2 r p_{t+1}^\star}{r + \hat{b}^2 p_{t+1}^\star} - \frac{\hat{a}^2 r p_{t+1}^{\mathsf{MM}}}{r + \hat{b}^2 p_{t+1}^{\mathsf{MM}}} \right) = \frac{\hat{a}^2 (p_{t+1}^\star - p_{t+1}^{\mathsf{MM}})}{(1 + \hat{b}^2 r^{-1} p_{t+1}^\star)(1 + \hat{b}^2 r^{-1} p_{t+1}^{\mathsf{MM}})} \tag{9.33}$$

Now, consider the example dynamical system where $a - \hat{a} = \epsilon_a$, $b - \hat{b} = \epsilon_b$, and $\hat{b} = 0$. Our upper bound in Theorem 6.3.2 states that,

$$|p_t^\star - p_t^{\mathsf{MM}}| \leq a^2 r^{-1} (p_{t+1}^\star)^2 (2b\epsilon_b + \epsilon_b^2) + p_{t+1}^\star (2a\epsilon_a + \epsilon_a^2) + (a + \epsilon_a)^2 |p_{t+1}^\star - p_{t+1}^{\mathsf{MM}}| \tag{9.34}$$

For the example system equation (9.32) simplifies to,

$$\frac{p^\star_{t+1}(a^2-\hat{a}^2)}{1+\hat{b}^2r^{-1}p^\star_{t+1}} + \frac{a^2r^{-1}(\hat{b}^2-b^2)(p^\star_{t+1})^2}{(1+b^2r^{-1}p^\star_{t+1})(1+\hat{b}^2r^{-1}p^\star_{t+1})} = p^\star_{t+1}(2a\epsilon_a+\epsilon_a^2) + \frac{a^2r^{-1}(p^\star_{t+1})^2(2b\epsilon_b+\epsilon_b^2)}{(1+b^2r^{-1}p^\star_{t+1})}$$

which matches the first two terms in the upper bound (equation (9.34)) upto a constant. Now, let's look at how equation (9.33) simplifies

$$\frac{\hat{a}^2(p^\star_{t+1}-p^{\mathsf{MM}}_{t+1})}{(1+\hat{b}^2r^{-1}p^\star_{t+1})(1+\hat{b}^2r^{-1}p^{\mathsf{MM}}_{t+1})} = (a+\epsilon_a)^2(p^\star_{t+1}-p^{\mathsf{MM}}_{t+1})$$

which matches the last term in the upper bound (equation (9.34)) exactly. Thus, we found an example where $|p^\star_t - p^{\mathsf{MM}}_t|$ matches the upper bound specified in Theorem 6.3.2 upto a constant.

Iterative Learning Control

Consider the difference

$$\begin{aligned} p^\star_t - p^{\mathsf{ILC}}_t &= \frac{a^2rp^\star_{t+1}}{r+b^2p^\star_{t+1}} - \frac{a\hat{a}rp^{\mathsf{ILC}}_{t+1}}{r+b\hat{b}p^{\mathsf{ILC}}_{t+1}} \\ &= \left(\frac{a^2rp^\star_{t+1}}{r+b^2p^\star_{t+1}} - \frac{a\hat{a}rp^\star_{t+1}}{r+b\hat{b}p^\star_{t+1}}\right) + \left(\frac{a\hat{a}rp^\star_{t+1}}{r+b\hat{b}p^\star_{t+1}} - \frac{a\hat{a}rp^{\mathsf{ILC}}_{t+1}}{r+b\hat{b}p^{\mathsf{ILC}}_{t+1}}\right) \end{aligned}$$

Once again let us look at each term separately. The first term can be simplified as

$$\left(\frac{a^2rp^\star_{t+1}}{r+b^2p^\star_{t+1}} - \frac{a\hat{a}rp^\star_{t+1}}{r+b\hat{b}p^\star_{t+1}}\right) = \frac{ap^\star_{t+1}(a-\hat{a})}{(1+b\hat{b}r^{-1}p^\star_{t+1})} + \frac{a^2br^{-1}(p^\star_{t+1})^2(\hat{b}-b)}{(1+b^2r^{-1}p^\star_{t+1})(1+b\hat{b}r^{-1}p^\star_{t+1})} \tag{9.35}$$

Similarly, the second term can be simplified as

$$\left(\frac{a\hat{a}rp^\star_{t+1}}{r+b\hat{b}p^\star_{t+1}} - \frac{a\hat{a}rp^{\mathsf{ILC}}_{t+1}}{r+b\hat{b}p^{\mathsf{ILC}}_{t+1}}\right) = \frac{a\hat{a}(p^\star_{t+1}-p^{\mathsf{ILC}}_{t+1})}{(1+b\hat{b}r^{-1}p^\star_{t+1})(1+b\hat{b}r^{-1}p^{\mathsf{ILC}}_{t+1})} \tag{9.36}$$

Similar to MM in the previous section, consider the example dynamical system where $a-\hat{a}=\epsilon_a$, $b-\hat{b}=\epsilon_b$ and $\hat{b}=0$. Our upper bound in Theorem 6.3.3 states that

$$|p^\star_t - p^{\mathsf{ILC}}_t| \le a^2(p^\star_{t+1})^2br^{-1}\epsilon_b + ap^\star_{t+1}\epsilon_a + a(a+\epsilon_a)|p^\star_{t+1}-p^{\mathsf{ILC}}_{t+1}| \tag{9.37}$$

For the example dynamical system, equation (9.35) simplifies to

$$\frac{ap^\star_{t+1}(a-\hat{a})}{(1+b\hat{b}r^{-1}p^\star_{t+1})} + \frac{a^2br^{-1}(p^\star_{t+1})^2(\hat{b}-b)}{(1+b^2r^{-1}p^\star_{t+1})(1+b\hat{b}r^{-1}p^\star_{t+1})} = ap^\star_{t+1}\epsilon_a + \frac{a^2(p^\star_{t+1})^2br^{-1}\epsilon_b}{(1+b^2r^{-1}p^\star_{t+1})}$$

which matches the first two terms in the upper bound (equation (9.37)) upto a constant. Now, let's look at how equation (9.36) simplifies

$$\frac{a\hat{a}(p^\star_{t+1}-p^{\mathsf{ILC}}_{t+1})}{(1+b\hat{b}r^{-1}p^\star_{t+1})(1+b\hat{b}r^{-1}p^{\mathsf{ILC}}_{t+1})} = a(a+\epsilon_a)(p^\star_{t+1}-p^{\mathsf{ILC}}_{t+1})$$

which matches the last term in the upper bound (equation (9.37)) exactly. Thus, we found that the same example also matches the upper bound specified in Theorem 6.3.3 upto a constant.

9.4.7 Experiment Details

Linear Dynamical System with Approximate Model

We use a horizon $H = 10$ and initial state $x_0 = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$.

Nonlinear Inverted Pendulum with Misspecified Mass

For the second experiment, we use the nonlinear dynamical system of an inverted pendulum. The state space is specified by $x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \in \mathbb{R}^2$ where θ is the angle between the pendulum and the vertical axis. The control input is $u = \tau \in \mathbb{R}$ specifying the torque τ to be applied at the base of the pendulum. The dynamics of the system are given by the ODE, $\ddot{\theta} = \frac{\bar{\tau}}{m\ell^2} - \frac{g\sin(\theta)}{\ell}$ where m is the mass of the pendulum, ℓ is the length of the pendulum, g is the acceleration due to gravity, and $\bar{\tau} = \max(\tau_{\min}, \min(\tau_{\max}, \tau))$ is the clipped torque based on torque limits. We use $\ell = 1$m, $\tau_{\max} = 8$Nm, $\tau_{\min} = -8$Nm, and $m = 1$kg.

We use a per time step cost function defined as $c(\theta, \tau) = 0.1\tau^2 + \theta^2$ where $\theta \in [-\pi, \pi]$, an initial state $x_0 = \begin{bmatrix} \frac{\pi}{2} \\ 0.5 \end{bmatrix}$, and a horizon $H = 20$. For all algorithms, we start with an initial control sequence consisting of zero torques for the entire horizon.

Nonlinear Planar Quadrotor Control in Wind

In our final experiment, we compare MM and ILC on a planar quadrotor control task in the presence of wind. The quadrotor is controlled using two propellers that provide upward thrusts (u_1, u_2) and allows movement in the 3D planar space described as (p_x, p_y, θ) where p_x, p_y are X, Y positions, and θ is the yaw of the quadrotor. The dynamics of the planar quadrotor is specified using a state vector $x \in \mathbb{R}^6$, control input $u \in \mathbb{R}^2$ as

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \\ \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix}, u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \dot{x} = \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \\ \frac{1}{m}(u_1 + u_2)\sin(\theta) \\ \frac{1}{m}(u_1 + u_2)\cos(\theta) - g \\ \frac{\ell}{2J}(u_2 - u_1) \end{bmatrix}$$

where m is the mass of the quadrotor, ℓ is the distance between the propellers, g is acceleration due to gravity, and J is the moment of inertia of the quadrotor. We use $m = 1$kg, $\ell = 0.3$m, and $J = 0.2m\ell^2$. The objective of the task is to move the quadrotor from an initial state x_0 at $(-3, 1)$ with zero velocity to a final state x_f at $(3, 1)$ with zero velocity. This is achieved using the per time-step cost function $c(x, u) = (x - x_f)^T Q(x - x_f) + (u - u_h)^T R(u - u_h)$ where $u_h = [\frac{1}{2}mg, \frac{1}{2}mg]$ are the hover controls. We use a horizon of $H = 60$ with a step size of 0.025 for RK4 integration.

Bibliography

This bibliography contains 150 references.

[AAH88] Chae H An, Christopher G Atkeson, and John M Hollerbach. *Model-based control of a robot manipulator*. MIT press, 1988.

[ADX10] Alekh Agarwal, Ofer Dekel, and Lin Xiao. "Optimal Algorithms for Online Convex Optimization with Multi-Point Bandit Feedback." In: *COLT*. Citeseer. 2010, pp. 28–40.

[Aga+14] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. "Taming the monster: A fast and simple algorithm for contextual bandits". In: *International Conference on Machine Learning*. 2014, pp. 1638–1646.

[Aga+21] Naman Agarwal, Elad Hazan, Anirudha Majumdar, and Karan Singh. "A Regret Minimization Approach to Iterative Learning Control". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 100–109. URL: http://proceedings.mlr.press/v139/agarwal21b.html.

[Ain+16] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. "Multi-Heuristic A". In: *Int. J. Robotics Res.* 35.1-3 (2016), pp. 224–243. DOI: 10.1177/0278364915594029. URL: https://doi.org/10.1177/0278364915594029.

[AKM84] Suguru Arimoto, Sadao Kawamura, and Fumio Miyazaki. "Bettering operation of Robots by learning". In: *J. Field Robotics* 1.2 (1984), pp. 123–140. DOI: 10.1002/rob.4620010203. URL: https://doi.org/10.1002/rob.4620010203.

[AM07] Brian DO Anderson and John B Moore. *Optimal control: linear quadratic methods*. Courier Corporation, 2007.

[AMS97a] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. "Locally Weighted Learning for Control". In: *Artif. Intell. Rev.* 11.1-5 (1997), pp. 75–113. DOI: 10.1023/A:1006511328852.

[AMS97b] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. "Locally Weighted Learning". In: *Artif. Intell. Rev.* 11.1-5 (1997), pp. 11–73. DOI: 10.1023/A:1006559212014. URL: https://doi.org/10.1023/A:1006559212014.

[AN05] Pieter Abbeel and Andrew Y. Ng. "Exploration and apprenticeship learning in reinforcement learning". In: *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*. Ed. by Luc De Raedt and Stefan Wrobel. Vol. 119. ACM International Conference Proceeding Series. ACM, 2005, pp. 1–8. DOI: 10.1145/1102351.1102352. URL: https://doi.org/10.1145/1102351.1102352.

[And+17] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 2017, pp. 5048–5058.

[AQN06a] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. "Using inaccurate models in reinforcement learning". In: *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*. 2006, pp. 1–8. DOI: 10.1145/1143844.1143845.

[AQN06b] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. "Using inaccurate models in reinforcement learning". In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 1–8.

[AS97] Christopher G. Atkeson and Stefan Schaal. "Learning tasks from a single demonstration". In: *Proceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico, USA, April 20-25, 1997*. 1997, pp. 1706–1712. DOI: 10.1109/ROBOT.1997.614389.

[ASG07] Ron Alterovitz, Thierry Siméon, and Kenneth Y. Goldberg. "The Stochastic Motion Roadmap: A Sampling Framework for Planning with Markov Motion Uncertainty". In: *Robotics: Science and Systems III, June 27-30, 2007, Georgia Institute of Technology, Atlanta, Georgia, USA*. Ed. by Wolfram Burgard, Oliver Brock, and Cyrill Stachniss. The MIT Press, 2007. DOI: 10.15607/RSS.2007.III.030. URL: http://www.roboticsproceedings.org/rss03/p30.html.

[ÅW13] Karl J Åström and Björn Wittenmark. *Adaptive control*. Courier Corporation, 2013.

[Ayo+20] Alex Ayoub, Zeyu Jia, Csaba Szepesvári, Mengdi Wang, and Lin Yang. "Model-Based Reinforcement Learning with Value-Targeted Regression". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 463–474. URL: http://proceedings.mlr.press/v119/ayoub20a.html.

[Bag+04] J Andrew Bagnell, Sham M Kakade, Jeff G Schneider, and Andrew Y Ng. "Policy search by dynamic programming". In: *Advances in neural information processing systems*. 2004, pp. 831–838.

[BAG12] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. "A robot path planning framework that learns from experience". In: *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*. IEEE, 2012, pp. 3671–3678. DOI: 10.1109/ICRA.2012.6224742. URL: https://doi.org/10.1109/ICRA.2012.6224742.

[BBS95] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. "Learning to Act Using Real-Time Dynamic Programming". In: *Artif. Intell.* 72.1-2 (1995), pp. 81–138. DOI: 10.1016/0004-3702(94)00011-O.

[BEL57] RICHARD BELLMAN. "A Markovian Decision Process". In: *Journal of Mathematics and Mechanics* 6.5 (1957), pp. 679–684. ISSN: 00959057, 19435274. URL: http://www.jstor.org/stable/24900506.

[Ber+19] Christopher Berner et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *CoRR* abs/1912.06680 (2019). arXiv: 1912.06680. URL: http://arxiv.org/abs/1912.06680.

[Ber05] Dimitri P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005. ISBN: 1886529264.

[Bro+16a] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[Bro+16b] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[BS01] J Andrew Bagnell and Jeff G Schneider. "Autonomous helicopter control using reinforcement learning policy search methods". In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 2. IEEE. 2001, pp. 1615–1620.

[BS10] Andrey Bernstein and Nahum Shimkin. "Adaptive-resolution reinforcement learning with polynomial exploration in deterministic domains". In: *Machine Learning* 81.3 (2010), pp. 359–397. DOI: 10.1007/s10994-010-5186-7.

[BT02] Ronen I. Brafman and Moshe Tennenholtz. "R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning". In: *J. Mach. Learn. Res.* 3 (2002), pp. 213–231.

[BT95] Dimitri P Bertsekas and John N Tsitsiklis. "Neuro-dynamic programming: an overview". In: *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*. Vol. 1. IEEE. 1995, pp. 560–564.

[CAN08] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. "Learning for control from multiple demonstrations". In: *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*. 2008, pp. 144–151. DOI: 10.1145/1390156.1390175.

[Car+17a] Yair Carmon, John C Duchi, Oliver Hinder, and Aaron Sidford. "Lower bounds for finding stationary points I". In: *arXiv preprint arXiv:1710.11606* (2017).

[Car+17b] Yair Carmon, John C Duchi, Oliver Hinder, and Aaron Sidford. "Lower Bounds for Finding Stationary Points II: First-Order Methods". In: *arXiv preprint arXiv:1711.00841* (2017).

[Cat07] Erin Catto. *Box2d physics engine*. 2007.

[Cho+05] Howie Choset, Kevin M Lynch, Seth Hutchinson, George A Kantor, and Wolfram Burgard. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[Coh+11] Benjamin J. Cohen, Gokul Subramania, Sachin Chitta, and Maxim Likhachev. "Planning for Manipulation with Adaptive Motion Primitives". In: *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*. 2011, pp. 5478–5485. DOI: 10.1109/ICRA.2011.5980550.

[Coh+18] Alon Cohen, Avinatan Hassidim, Tomer Koren, Nevena Lazic, Yishay Mansour, and Kunal Talwar. "Online Linear Quadratic Control". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1028–1037. URL: http://proceedings.mlr.press/v80/cohen18b.html.

[Cou+13] Erwin Coumans et al. "Bullet physics library". In: *Open source: bulletphysics. org* 15.49 (2013), p. 5.

[CRT16] Hao-Tien Chiang, Nathanael Rackley, and Lydia Tapia. "Runtime SES planning: Online motion planning in environments with stochastic dynamics and uncertainty". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016*. IEEE, 2016, pp. 4802–4809. DOI: 10.1109/IROS.2016.7759705. URL: https://doi.org/10.1109/IROS.2016.7759705.

[Dea+20] Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. "On the Sample Complexity of the Linear Quadratic Regulator". In: *Found. Comput. Math.* 20.4 (2020), pp. 633–679. DOI: 10.1007/s10208-019-09426-y. URL: https://doi.org/10.1007/s10208-019-09426-y.

[DFR15] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 37.2 (2015), pp. 408–423. DOI: 10.1109/TPAMI.2013.218.

[Dia10] Rosen Diankov. "Automated Construction of Robotic Manipulation Programs". PhD thesis. Carnegie Mellon University, Robotics Institute, Aug. 2010. URL: http://www.programmingvision.com/rosen_diankov_thesis.pdf.

[DR10] Marc Peter Deisenroth and Carl Edward Rasmussen. "Reducing model bias in reinforcement learning". In: (2010).

[Duc+15] J. C. Duchi, M. I. Jordan, M. J. Wainwright, and A. Wibisono. "Optimal Rates for Zero-Order Convex Optimization: The Power of Two Function Evaluations". In: *IEEE Transactions on Information Theory* 61.5 (May 2015), pp. 2788–2806. ISSN: 0018-9448. DOI: 10.1109/TIT.2015.2409256.

[EBG12] Nabil Essahbi, Belhassen Chedli Bouzgarrou, and Grigore Gogu. "Soft Material Modeling for Robotic Manipulation". In: *Mechanisms, Mechanical Transmissions and Robotics*. Vol. 162. Applied Mechanics and Materials. Trans Tech Publications Ltd, Apr. 2012, pp. 184–193. DOI: 10.4028/www.scientific.net/AMM.162.184.

[Eys+21] Benjamin Eysenbach, Alexander Khazatsky, Sergey Levine, and Ruslan Salakhutdinov. "Mismatched No More: Joint Model-Policy Optimization for Model-Based RL". In: *CoRR* abs/2110.02758 (2021). arXiv: 2110.02758. URL: https://arxiv.org/abs/2110.02758.

[Far18] Amir Massoud Farahmand. "Iterative value-aware model learning". In: *Advances in Neural Information Processing Systems* 2018-December.Section 3 (2018), pp. 9072–9083. ISSN: 10495258.

[Faz+18] Maryam Fazel, Rong Ge, Sham M. Kakade, and Mehran Mesbahi. "Global Convergence of Policy Gradient Methods for the Linear Quadratic Regulator". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1466–1475. URL: http://proceedings.mlr.press/v80/fazel18a.html.

[FKM05] Abraham D Flaxman, Adam Tauman Kalai, and H Brendan McMahan. "Online convex optimization in the bandit setting: gradient descent without a gradient". In: *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2005, pp. 385–394.

[Gar18] Caelan Reed Garrett. *ss-pybullet library*. 2018. URL: https://github.com/caelan/ss-pybullet.

[GL13] Saeed Ghadimi and Guanghui Lan. "Stochastic first-and zeroth-order methods for nonconvex stochastic programming". In: *SIAM Journal on Optimization* 23.4 (2013), pp. 2341–2368.

[Gri+20] Christopher Grimm, André Barreto, Satinder Singh, and David Silver. "The Value Equivalence Principle for Model-Based Reinforcement Learning". In: *Advances in Neural Information Processing Systems* 33 (2020).

[Gri+21] Christopher Grimm, André Barreto, Gregory Farquhar, David Silver, and Satinder Singh. "Proper Value Equivalence". In: *CoRR* abs/2106.10316 (2021). arXiv: 2106.10316. URL: https://arxiv.org/abs/2106.10316.

[Hau+06] Kris K. Hauser, Timothy Bretl, Kensuke Harada, and Jean-Claude Latombe. "Using Motion Primitives in Probabilistic Sample-Based Planning for Humanoid Robots". In: *Algorithmic Foundation of Robotics VII, Selected Contributions of the Seventh International Workshop on the Algorithmic Foundations of Robotics, WAFR 2006, July 16-18, 2006, New York, NY, USA*. 2006, pp. 507–522. DOI: 10.1007/978-3-540-68405-3_32.

[Hau12] Kris K. Hauser. "On responsiveness, safety, and completeness in real-time motion planning". In: *Auton. Robots* 32.1 (2012), pp. 35–48. DOI: 10.1007/s10514-011-9254-z. URL: https://doi.org/10.1007/s10514-011-9254-z.

[Hes+18] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 3215–3222. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17204.

[HGS16] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. Ed. by Dale Schuurmans and Michael P. Wellman. AAAI Press, 2016, pp. 2094–2100. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389.

[HI08] Verena Heidrich-Meisner and Christian Igel. "Evolution strategies for direct policy search". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2008, pp. 428–437.

[HJ12] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[HY15] Sehoon Ha and Katsu Yamane. "Reducing hardware experiments for model learning and policy optimization". In: *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*. 2015, pp. 2620–2626. DOI: 10.1109/ICRA.2015.7139552.

[Isl+20] Fahad Islam, Anirudh Vemula, Sung-Kyun Kim, Andrew Dornbush, Oren Salzman, and Maxim Likhachev. "Planning, Learning and Reasoning Framework for Robot Truck Unloading". In: *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*. IEEE, 2020, pp. 5011–

5017. DOI: 10.1109/ICRA40945.2020.9196604. URL: https://doi.org/10.1109/ICRA40945.2020.9196604.

[Jan+19] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. "When to Trust Your Model: Model-Based Policy Optimization". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. 2019, pp. 12498–12509.

[Jia18] Nan Jiang. "PAC Reinforcement Learning With an Imperfect Model". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. 2018, pp. 3334–3341.

[Jos+13] Joshua Mason Joseph, Alborz Geramifard, John W. Roberts, Jonathan P. How, and Nicholas Roy. "Reinforcement learning with misspecified model classes". In: *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*. IEEE, 2013, pp. 939–946. DOI: 10.1109/ICRA.2013.6630686. URL: https://doi.org/10.1109/ICRA.2013.6630686.

[JS07] Nicholas K. Jong and Peter Stone. "Model-based function approximation in reinforcement learning". In: *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, USA, May 14-18, 2007*. 2007, p. 95. DOI: 10.1145/1329125.1329242.

[JX01] Xuerong Ji and Jing Xiao. "Planning Motions Compliant to Complex Contact States". In: *IJ Robotics Res.* 20.6 (2001), pp. 446–465. DOI: 10.1177/02783640122067480. URL: https://doi.org/10.1177/02783640122067480.

[Kak02] Sham Kakade. "A natural policy gradient". In: *NIPS* (2002).

[Kak03] Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

[KB14] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[KBP13] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[Kid+20] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. "MOReL: Model-Based Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/f7efa4f864ae9b88d43527f4b14f750f-Abstract.html.

[KKL03] Sham M. Kakade, Michael J. Kearns, and John Langford. "Exploration in Metric State Spaces". In: *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*. 2003, pp. 306–312.

[KL02] Sham Kakade and John Langford. "Approximately optimal approximate reinforcement learning". In: *ICML*. 2002.

[KL06] Sven Koenig and Maxim Likhachev. "Real-time adaptive A*". In: *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*. 2006, pp. 281–288. DOI: 10.1145/1160633.1160682.

[KLC98] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and Acting in Partially Observable Stochastic Domains". In: *Artif. Intell.* 101.1-2 (1998), pp. 99–134. DOI: 10.1016/S0004-3702(98)00023-X. URL: https://doi.org/10.1016/S0004-3702(98)00023-X.

[KNL91] Tae-Yong Kuc, Kwanghee Nam, and Jin S. Lee. "An iterative learning control of robot manipulators". In: *IEEE Trans. Robotics Autom.* 7.6 (1991), pp. 835–842. DOI: 10.1109/70.105392. URL: https://doi.org/10.1109/70.105392.

[Kol10] J Zico Kolter. *Learning and control with inaccurate models*. Stanford University, 2010.

[Kor90] Richard E. Korf. "Real-Time Heuristic Search". In: *Artif. Intell.* 42.2-3 (1990), pp. 189–211. DOI: 10.1016/0004-3702(90)90054-4.

[KPC93] Michail M. Konstantinov, Petko Hr. Petkov, and Nicolai Christov. "Perturbation analysis of the discrete Riccati equation". In: *Kybernetika* 29.1 (1993), pp. 18–29. URL: http://www.kybernetika.cz/content/1993/1/18.

[KS02] Michael J. Kearns and Satinder P. Singh. "Near-Optimal Reinforcement Learning in Polynomial Time". In: *Machine Learning* 49.2-3 (2002), pp. 209–232. DOI: 10.1023/A:1017984413808.

[KS93] Sven Koenig and Reid G. Simmons. "Complexity Analysis of Real-Time Reinforcement Learning". In: *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993*. 1993, pp. 99–107.

[Lam+20] Nathan O. Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. "Objective Mismatch in Model-based Reinforcement Learning". In: *Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control, L4DC 2020, Online Event, Berkeley, CA, USA, 11-12 June 2020*. Ed. by Alexandre M. Bayen, Ali Jadbabaie, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger. Vol. 120. Proceedings of Machine Learning Research. PMLR, 2020, pp. 761–770. URL: http://proceedings.mlr.press/v120/lambert20a.html.

[Lat91] Jean-Claude Latombe. *Robot motion planning*. Vol. 124. The Kluwer international series in engineering and computer science. Kluwer, 1991. ISBN: 978-0-7923-9206-4. DOI: 10.1007/978-1-4615-4022-9. URL: https://doi.org/10.1007/978-1-4615-4022-9.

[LaV06] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. ISBN: 9780511546877. DOI: 10.1017/CBO9780511546877. URL: http://planning.cs.uiuc.edu/.

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[Lee+20a] Michelle A. Lee, Carlos Florensa, Jonathan Tremblay, Nathan D. Ratliff, Animesh Garg, Fabio Ramos, and Dieter Fox. "Guided Uncertainty-Aware Policy Optimiza-

tion: Combining Learning and Model-Based Strategies for Sample-Efficient Policy Learning". In: *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*. IEEE, 2020, pp. 7505–7512. DOI: 10.1109/ICRA40945.2020.9197125. URL: https://doi.org/10.1109/ICRA40945.2020.9197125.

[Lee+20b] Michelle A. Lee, Carlos Florensa, Jonathan Tremblay, Nathan D. Ratliff, Animesh Garg, Fabio Ramos, and Dieter Fox. "Guided Uncertainty-Aware Policy Optimization: Combining Learning and Model-Based Strategies for Sample-Efficient Policy Learning". In: *CoRR* abs/2005.10872 (2020). arXiv: 2005.10872. URL: https://arxiv.org/abs/2005.10872.

[LJ01] Steven M. LaValle and James J. Kuffner Jr. "Randomized Kinodynamic Planning". In: *I. J. Robotics Res.* 20.5 (2001), pp. 378–400. DOI: 10.1177/02783640122067453.

[Lju10] Lennart Ljung. "Perspectives on system identification". In: *Annu. Rev. Control.* 34.1 (2010), pp. 1–12. DOI: 10.1016/j.arcontrol.2009.12.001. URL: https://doi.org/10.1016/j.arcontrol.2009.12.001.

[LLK20] Alex Lagrassa, Steven Lee, and Oliver Kroemer. "Learning skills to patch plans based on inaccurate models". In: *2020 IEEE International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[LT04] Weiwei Li and Emanuel Todorov. "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems". In: *ICINCO 2004, Proceedings of the First International Conference on Informatics in Control, Automation and Robotics, Setúbal, Portugal, August 25-28, 2004*. Ed. by Helder Araújo, Alves Vieira, José Braz, Bruno Encarnação, and Marina Carvalho. INSTICC Press, 2004, pp. 222–229.

[McC+20] Dale McConachie, Thomas Power, Peter Mitrano, and Dmitry Berenson. "Learning When to Trust a Dynamics Model for Planning in Reduced State Spaces". In: *IEEE Robotics Autom. Lett.* 5.2 (2020), pp. 3540–3547. DOI: 10.1109/LRA.2020.2972858.

[MDB92] Kevin L. Moore, Mohammed Dahleh, and S. P. Bhattacharyya. "Iterative learning control: A survey and new results". In: *J. Field Robotics* 9.5 (1992), pp. 563–594. DOI: 10.1002/rob.4620090502. URL: https://doi.org/10.1002/rob.4620090502.

[MGR18] Horia Mania, Aurelia Guy, and Benjamin Recht. "Simple random search provides a competitive approach to reinforcement learning". In: *arXiv preprint arXiv:1803.07055* (2018).

[MMB21] Peter Mitrano, Dale McConachie, and Dmitry Berenson. "Learning where to trust unreliable models in an unstructured world for deformable object manipulation". In: *Sci. Robotics* 6.54 (2021), p. 8170. DOI: 10.1126/scirobotics.abd8170. URL: https://doi.org/10.1126/scirobotics.abd8170.

[Mni+15] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[Mod+21] Nirbhay Modhe, Harish Kamath, Dhruv Batra, and Ashwin Kalyan. "Model-Advantage Optimization for Model-Based Reinforcement Learning". In: *CoRR* abs/2106.14080 (2021). arXiv: 2106.14080. URL: https://arxiv.org/abs/2106.14080.

[MRG03] Shie Mannor, Reuven Y Rubinstein, and Yohai Gat. "The cross entropy method for fast policy search". In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 512–519.

[MSD12] Fabian L. Mueller, Angela P. Schoellig, and Raffaello D'Andrea. "Iterative learning of feed-forward corrections for high-performance tracking". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*. IEEE, 2012, pp. 3276–3281. DOI: 10.1109/IROS.2012.6385647. URL: https://doi.org/10.1109/IROS.2012.6385647.

[MTR19] Horia Mania, Stephen Tu, and Benjamin Recht. "Certainty Equivalent Control of LQR is Efficient". In: *CoRR* abs/1902.07826 (2019). arXiv: 1902.07826. URL: http://arxiv.org/abs/1902.07826.

[MVL21] Nader Maray, Anirudh Vemula, and Maxim Likhachev. "Improved Soft Duplicate Detection in Search-Based Motion Planning". In: *CoRR* abs/2109.12427 (2021). arXiv: 2109.12427. URL: https://arxiv.org/abs/2109.12427.

[Nag+18] 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 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 7559–7566. DOI: 10.1109/ICRA.2018.8463189. URL: https://doi.org/10.1109/ICRA.2018.8463189.

[Nik+21] Evgenii Nikishin, Romina Abachi, Rishabh Agarwal, and Pierre-Luc Bacon. "Control-Oriented Model-Based Reinforcement Learning with Implicit Differentiation". In: *CoRR* abs/2106.03273 (2021). arXiv: 2106.03273. URL: https://arxiv.org/abs/2106.03273.

[NL08] Ali Nouri and Michael L. Littman. "Multi-resolution Exploration in Continuous Spaces". In: *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*. 2008, pp. 1209–1216.

[NS17] Yurii Nesterov and Vladimir Spokoiny. "Random gradient-free minimization of convex functions". In: *Foundations of Computational Mathematics* 17.2 (2017), pp. 527–566.

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[PB21] Thomas Power and Dmitry Berenson. "Keep It Simple: Data-Efficient Learning for Controlling Complex Systems With Simple Models". In: *IEEE Robotics Autom. Lett.* 6.2 (2021), pp. 1184–1191. DOI: 10.1109/LRA.2021.3056368. URL: https://doi.org/10.1109/LRA.2021.3056368.

[PKK09] Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. "Differentially constrained mobile robot motion planning in state lattices". In: *J. Field Robotics* 26.3 (2009), pp. 308–333. DOI: 10.1002/rob.20285. URL: https://doi.org/10.1002/rob.20285.

[PS08] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural networks* 21.4 (2008), pp. 682–697.

[Raj+17] Aravind Rajeswaran, Kendall Lowrey, Emanuel V Todorov, and Sham M Kakade. "Towards generalization and simplicity in continuous control". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6550–6561.

[RB12] Stéphane Ross and Drew Bagnell. "Agnostic System Identification for Model-Based Reinforcement Learning". In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012. URL: http://icml.cc/2012/papers/833.pdf.

[RKK18] Divyam Rastogi, Ivan Koryakovskiy, and Jens Kober. "Sample-efficient reinforcement learning via difference models". In: *Machine Learning in Planning and Control of Robot Motion Workshop at ICRA*. 2018.

[RS90] J. A. Reeds and L. A. Shepp. "Optimal paths for a car that goes both forwards and backwards." In: *Pacific J. Math.* 145.2 (1990), pp. 367–393. URL: https://projecteuclid.org:443/euclid.pjm/1102645450.

[Sal+17] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (2017).

[Sav+17] Matteo Saveriano, Yuchao Yin, Pietro Falco, and Dongheui Lee. "Data-efficient control policy search using residual dynamics learning". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*. 2017, pp. 4709–4715. DOI: 10.1109/IROS.2017.8206343.

[SB98] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. Vol. 135. MIT Press Cambridge, 1998.

[SBA16] Wen Sun, Jur van den Berg, and Ron Alterovitz. "Stochastic Extended LQR for Optimization-Based Motion Planning Under Uncertainty". In: *IEEE Trans Autom. Sci. Eng.* 13.2 (2016), pp. 437–447. DOI: 10.1109/TASE.2016.2517124. URL: https://doi.org/10.1109/TASE.2016.2517124.

[Sch+15] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. "Trust Region Policy Optimization." In: *ICML*. 2015, pp. 1889–1897.

[Sch+19] Julian Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: *CoRR* abs/1911.08265 (2019). arXiv: 1911.08265. URL: http://arxiv.org/abs/1911.08265.

[Seh+10] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. "Parameter-exploring policy gradients". In: *Neural Networks* 23.4 (2010), pp. 551–559.

[SF20] Max Simchowitz and Dylan J. Foster. "Naive Exploration is Optimal for Online LQR". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 8937–8948. URL: http://proceedings.mlr.press/v119/simchowitz20a.html.

[Sha13] Ohad Shamir. "On the complexity of bandit and derivative-free stochastic convex optimization". In: *Conference on Learning Theory*. 2013, pp. 3–24.

[Sha17] Ohad Shamir. "An optimal algorithm for bandit and zero-order convex optimization with two-point feedback". In: *Journal of Machine Learning Research* 18.52 (2017), pp. 1–11.

[Sil+14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic policy gradient algorithms". In: *ICML*. 2014.

[Sil+17] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nat.* 550.7676 (2017), pp. 354–359. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270.

[SL06] István Szita and András Lörincz. "Learning Tetris using the noisy cross-entropy method". In: *Neural computation* 18.12 (2006), pp. 2936–2941.

[SMD12] Angela P. Schoellig, Fabian L. Mueller, and Raffaello D'Andrea. "Optimization-based iterative learning for precise quadrocopter trajectory tracking". In: *Auton. Robots* 33.1-2 (2012), pp. 103–127. DOI: 10.1007/s10514-012-9283-2. URL: https://doi.org/10.1007/s10514-012-9283-2.

[Sun+19a] Wen Sun, Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, and John Langford. "Model-based RL in Contextual Decision Processes: PAC bounds and Exponential Improvements over Model-free Approaches". In: *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*. Ed. by Alina Beygelzimer and Daniel Hsu. Vol. 99. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2898–2933. URL: http://proceedings.mlr.press/v99/sun19a.html.

[Sun+19b] Wen Sun, Anirudh Vemula, Byron Boots, and Drew Bagnell. "Provably Efficient Imitation Learning from Observation Alone". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6036–6045. URL: http://proceedings.mlr.press/v97/sun19b.html.

[Sut91] Richard S. Sutton. "Dyna, an Integrated Architecture for Learning, Planning, and Reacting". In: *SIGART Bulletin* 2.4 (1991), pp. 160–163. DOI: 10.1145/122344.122377.

[Tal+19] Abhijeet Tallavajhula, Adrian Schoisengeier, Sung-Kyun Kim, Anirudh Vemula, Levi Lister, and Oren Salzman. "Task-Informed Fidelity Management for Speeding Up Robotics Simulation". In: *CoRR* abs/1910.12284 (2019). arXiv: 1910.12284. URL: http://arxiv.org/abs/1910.12284.

[TET12] 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, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[Thr92] Sebastian Thrun. *Efficient Exploration In Reinforcement Learning*. Tech. rep. CMU-CS-92-102. Pittsburgh, PA: Carnegie Mellon University, Jan. 1992.

[TR18] Stephen Tu and Benjamin Recht. "The Gap Between Model-Based and Model-Free Methods on the Linear Quadratic Regulator: An Asymptotic Viewpoint". In: *arXiv preprint arXiv:1812.03565* (2018).

[TSC11] Matthew Tesch, Jeff Schneider, and Howie Choset. "Using response surfaces and expected improvement to optimize snake robot gait parameters". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, pp. 1069–1074.

[VB20] Anirudh Vemula and J. Andrew Bagnell. "Tron: A Fast Solver for Trajectory Optimization with Non-Smooth Cost Functions". In: *59th IEEE Conference on Decision and Control, CDC 2020, Jeju Island, South Korea, December 14-18, 2020*. IEEE, 2020, pp. 4157–4163. DOI: 10.1109/CDC42340.2020.9303915. URL: https://doi.org/10.1109/CDC42340.2020.9303915.

[VBL20] Anirudh Vemula, J. Andrew Bagnell, and Maxim Likhachev. "CMAX++ : Leveraging Experience in Planning and Execution using Inaccurate Models". In: *CoRR* abs/2009.09942 (2020). arXiv: 2009.09942. URL: https://arxiv.org/abs/2009.09942.

[Vem+20] Anirudh Vemula, Yash Oza, J. Bagnell, and Maxim Likhachev. "Planning and Execution using Inaccurate Models with Provable Guarantees". In: *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA, July 2020. DOI: 10.15607/RSS.2020.XVI.001.

[Vem+21] Anirudh Vemula, Wen Sun, Maxim Likhachev, and J. Andrew Bagnell. "On the Effectiveness of Iterative Learning Control". In: *CoRR* abs/2111.09434 (2021). arXiv: 2111.09434. URL: https://arxiv.org/abs/2111.09434.

[VJY21] Cameron Voloshin, Nan Jiang, and Yisong Yue. "Minimax Model Learning". In: *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*. Ed. by Arindam Banerjee and Kenji Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 2021, pp. 1612–1620. URL: http://proceedings.mlr.press/v130/voloshin21a.html.

[VMO17] Anirudh Vemula, Katharina Mülling, and Jean Oh. "Modeling cooperative navigation in dense human crowds". In: *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*. 2017, pp. 1685–1692. DOI: 10.1109/ICRA.2017.7989199.

[VS00] Sethu Vijayakumar and Stefan Schaal. "Locally Weighted Projection Regression: Incremental Real Time Learning in High Dimensional Space". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*. 2000, pp. 1079–1086.

[VSB19] Anirudh Vemula, Wen Sun, and J. Andrew Bagnell. "Contrasting Exploration in Parameter and Action Space: A Zeroth-Order Optimization Perspective". In: *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2926–2935. URL: http://proceedings.mlr.press/v89/vemula19a.html.

[Wan+19] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. "Benchmarking Model-Based Reinforcement Learning". In: *CoRR* abs/1907.02057 (2019). arXiv: 1907.02057. URL: http://arxiv.org/abs/1907.02057.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. "Technical Note Q-Learning". In: *Mach. Learn.* 8 (1992), pp. 279–292. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

[Wil92] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* (1992).

[Yu+20] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y. Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. "MOPO: Model-based Offline Policy Optimization". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.

neurips.cc/paper/2020/hash/a322852ce0df73e204b7e67cbbef0d0a-Abstract.html.

[Zel92] Alexander Zelinsky. "A mobile robot exploration algorithm". In: *IEEE Trans. Robotics Autom.* 8.6 (1992), pp. 707–717. DOI: 10.1109/70.182671. URL: https://doi.org/10.1109/70.182671.

[Zha+11] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. "Analysis and improvement of policy gradient estimation". In: *Advances in Neural Information Processing Systems*. 2011, pp. 262–270.

[Zuc+11] Matthew Zucker, Nathan D. Ratliff, Martin Stolle, Joel E. Chestnutt, J. Andrew Bagnell, Christopher G. Atkeson, and James Kuffner. "Optimization and learning for rough terrain legged locomotion". In: *Int. J. Robotics Res.* 30.2 (2011), pp. 175–191. DOI: 10.1177/0278364910392608. URL: https://doi.org/10.1177/0278364910392608.