APPRENTICESHIP LEARNING AND REINFORCEMENT LEARNING WITH APPLICATION TO ROBOTIC CONTROL

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Pieter Abbeel
August 2008

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Andrew Y. Ng) Principal Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Daphne Koller)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Stephen M. Rock)

Approved for the University Committee on Graduate Studies.

Abstract

Many problems in robotics have unknown, stochastic, high-dimensional, and highly non-linear dynamics, and offer significant challenges to both traditional control methods and reinforcement learning algorithms. Some of the key difficulties that arise in these problems are: (i) It is often difficult to write down, in closed form, a formal specification of the control task. For example, what is the objective function for "flying well"? (ii) It is often difficult to build a good dynamics model because of both data collection and data modeling challenges (similar to the "exploration problem" in reinforcement learning). (iii) It is often computationally expensive to find closed-loop controllers for high dimensional, stochastic domains.

We describe learning algorithms with formal performance guarantees which show that these problems can be efficiently addressed in the apprenticeship learning setting—the setting when expert demonstrations of the task are available. Our algorithms are guaranteed to return a control policy with performance comparable to the expert's. We evaluate performance on the same task and in the same (typically stochastic, high-dimensional and non-linear) environment as the expert.

Besides having theoretical guarantees, our algorithms have also enabled us to solve some previously unsolved real-world control problems: They have enabled a quadruped robot to traverse challenging, previously unseen terrain. They have significantly extended the state-of-the-art in autonomous helicopter flight. Our helicopter has performed by far the most challenging aerobatic maneuvers performed by any autonomous helicopter to date, including maneuvers such as continuous in-place flips, rolls and tic-tocs, which only exceptional expert human pilots can fly. Our aerobatic flight performance is comparable to that of the best human pilots.

To my parents—Miel Abbeel and Lutgart Vandermeerschen.

Acknowledgments

This dissertation is the results of a very close collaboration with my advisor, Andrew Ng. Andrew has been an amazing mentor throughout my Ph.D. student career. I have learned a great many things from him. Just to name a few that stand out: the way he goes about choosing and solving research problems, how he interacts with his advisees, how he teaches, and how he communicates research through papers and presentations. I have also over and over benefited from his genuine enthusiasm about our research and his ever willingness to make time to meet with me.

I am very grateful to Daphne Koller. While I was a masters student at Stanford, Daphne's lectures sparked my interest in artificial intelligence. She taught me all my foundations on probabilistic graphical models. I started my research efforts in artificial intelligence in her group and throughout my Ph.D., we have continued to have very exciting and fruitful discussions.

I would like to thank Steve Rock for the many thought-provoking questions during my defense and the dissertation writing process.

Ben Van Roy's class on approximate dynamic programming during the first year of my Ph.D. greatly sharpened my thinking about reinforcement learning. I am also thankful to Ben for chairing my defense.

I have greatly benefited from Claire Tomlin's class on nonlinear control. I am also thankful to Claire for being on my defense committee.

I am very grateful to Stephen Boyd: his classes, besides arguably being some of the most entertaining classes I have taken, greatly shaped my thinking about optimization and control. His book and lecture notes are among the references I have consulted the most throughout my Ph.D.

During the last three years of my Ph.D., I have collaborated very closely with Adam Coates. Our interactions have included long flight-test days out on the Sand Hill fields; deep discussions on learning and control in Gates 113; late night phone conversations about the latest research ideas on our mind. Adam has been simply amazing to chat with about research. My Ph.D. experience would not have been remotely the same without him as a fellow student and friend.

Our helicopter pilot, Garett Oku, has always been very supportive of our autonomous flight research efforts. He is the "expert" providing us with the demonstrations for our apprenticeship learning algorithms described in this dissertation. It has been a great pleasure to get to collaborate with him.

I met Vivek Farias in Ben Van Roy's approximate programming class during my first year in the Ph.D. program. Ever since, Vivek has been a great friend. I can't count the number of lunches and coffees we have had together, and invariably, he was always very interested in and supportive of my research work.

I have been very lucky to have been part of Andrew's research group, and to have had the chance to interact with his group of students, Adam Coates, Chuong Do (Tom), J. Zico Kolter, Quoc Le, Honglak Lee, Morgan Quigley, Rajat Raina, Olga Russakovsky, Ashutosh Saxena, Yirong Shen, Rion Snow. I am very thankful for the many intellectually inspiring interactions as well as the many friendships that have developed.

I would like to thank Morgan Quigley, Zico Kolter, Timothy Hunter and Adam Coates for the many entertaining lunches over the past couple of years. Morgan has also patiently explained to me uncountably many things about robotic hardware. Zico has invariably been excited about discussing novel algorithmic ideas.

I am especially grateful to Ben Taskar. When I worked in Daphne's group as a masters student, we worked very closely together. Since then, we have continued to have many great research (and otherwise) discussions. He has been the more senior student mentoring me through the various stages of my research career as well as a great friend.

I have greatly benefited from my many interactions with Gabe Hoffmann about a large variety of topics ranging from control theoretic discussions through hardware

specifics for our helicopter/his quadrotor.

It has been my pleasure to get to interact with my fellow students in the Stanford AI lab, in particular my office mates Haidong Wang and SuIn Lee, with whom I spent many late nights at Gates. I would also like to single out Gal Elidan, who has given me invaluable feedback on various presentations.

I would like to thank my collaborators, for the inspiring intellectual interactions, their hard work and our great times together, Gal Chechik, Adam Coates, David De Lorenzo, Dmitri Dolgov, Ethan Dreyfuss, Quan Gan, Varun Ganapathi, Timothy Hunter, Daphne Koller, Zico Kolter, Jessa Lee, SuIn Lee, Mike Montemerlo, Andrew Ng, Garett Oku, Morgan Quigley, Brian Sa, Jeffrey Spehar, Ben Taskar, Sebastian Thrun, Ben Tse, Mark Woodward, Tim Worley.

There are a number of close friends at Stanford who are not directly related to my research, but their company made this experience so much more enjoyable. I am particularly grateful to Alenka Zeman, who was my closest companion for most of my Ph.D. I am also particularly grateful to my five-year-long roommate and close friend David Sears, who, amongst others, was always interested to listen to my stories and who got me hooked on tennis. Merci beaucoup to Sriram Viji, who has been a great friend since my very first days at Stanford. I am also very grateful for my friendships with Ciamac Moallemi and Emre Oto.

Many thanks to my many friends from Belgium, who had nothing to do with any of the work in this thesis, but who always warmly welcomed me back whenever I was visiting.

To my sisters, Tine, Annelies, Karlien, Sandrien, thanks for being such loving, caring and fun companions throughout my life.

Last and foremost, I am thanking my parents, Miel and Lutgart, who have given me unconditional love, support and encouragement throughout my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In model-based reinforcement learning (RL) and optimal control we assume there is (i) a reward function, which describes the task, and (ii) a dynamics model, which describes how the system, and often also the environment, will evolve over time as a (stochastic) function of the current state and actions.

Given a reward function and a dynamics model, model-based RL and optimal control algorithms then attempt to find an optimal control policy, which maximizes the expected sum of rewards accumulated over time.[1]

In practice, for many control problems it can be very challenging to (i) Write down, in closed form, a formal specification of the control task. For example, what is the objective function for "flying well"? (ii) Provide a sufficiently accurate dynamics model because of both data collection and data modeling challenges (similar to the "exploration problem" in reinforcement learning). (iii) Find a (near-)optimal control policy, even when the reward function and the dynamics model are given.

In this dissertation, we describe learning algorithms with formal performance guarantees which show that each of these problems can be efficiently addressed in the apprenticeship learning setting—the setting when expert demonstrations of the task are available.

[1]Model-based RL and optimal control consider very similar settings: they attempt to find an optimal control policy for a given reward (cost) function and dynamics model. In this dissertation we do not attempt to distinguish between the two terms and consider them interchangeable.

1.1 Reward function

In most control problems, our final goal is to find a good control policy—i.e., a prescription that tells us a good action to take for every possible state of the system. The reinforcement learning and optimal control formalisms are useful for many problems because it is often easier to specify a reward function than to directly specify the optimal policy.

However, even the reward function is frequently difficult to specify manually.[2] In fact, even for experienced control engineers, it can be very challenging to articulate a formal specification for many practical control tasks. For example, what would be the right objective functions for having a helicopter fly a high-performance aerobatic airshow; for having a quadruped climb across a challenging, rocky terrain; or for having a car navigate in highway traffic?

In practice, this means that the reward function is often manually tweaked (cf. reward shaping, Ng et al., 1999) until the desired behavior is obtained. From conversations with engineers in industry and our own experience in applying reinforcement learning algorithms to several robots (including a quadruped robot and a helicopter), we believe that, for many problems, the difficulty of manually specifying a reward function represents a significant barrier to the broader applicability of reinforcement learning and optimal control algorithms.

While often it is hard, even for experts, to articulate the task in the form of a reward function, experts can often much more easily demonstrate good performance on the task. For example, rather than describing to us exactly what a good aerobatic helicopter flip entails, it is much easier for our expert helicopter pilot to just fly our helicopter and demonstrate a few aerobatic flips.

In this dissertation we present apprenticeship learning algorithms which use the expert demonstrations to extract a definition of the task in the form of a reward

[2] Classical control and robust control methods offer similar challenges. Classical control methods often require a task specification in the form of frequency domain properties or poles and zeros. Robust control methods often require a task specification in the form of, not only a reward function, but also model uncertainty and/or perturbation magnitude estimates.

function—a representation that can be readily used as a task description in reinforcement learning and optimal control algorithms. Our algorithms enable us to learn two important classes of reward functions:

(i) A reward function that linearly trades off between a known set of features. If the set of features is sufficiently rich, this assumption is fairly unrestrictive. Features can be discrete valued, for example, whether the car is on the road, whether the car is in a collision etc. Features can also be continuous valued, for example, we could have features corresponding to the squared distance to the target position or orientation.

(ii) A reward function for trajectory following tasks—here the key is to find a trajectory in state-space that corresponds to the task to be executed. Hand-specifying such a trajectory is often extremely time-consuming, or even impossible when the system dynamics is very complex. A natural alternative is to use a set of trajectories from expert demonstrations. As in most practical situations perfect expert demonstrations are very hard to obtain, merely using a single trajectory demonstrated by the expert is insufficient to capture their intent. We propose an algorithm that automatically recovers the most likely *intended* expert trajectory from a large set of sub-optimal demonstrations. Most likely here is with respect to a probabilistic model that encodes the robot's dynamics and models how the expert tends to deviate from their intended trajectory.

1.2 Dynamics model

To apply model-based RL and optimal control algorithms when the dynamics are not known in advance, the dynamics often need to be learned from observations of the system. A key problem in learning the system dynamics is that of *exploration*: How can we ensure that all relevant parts of the state space are visited sufficiently often that we manage to build up a sufficiently accurate dynamics model for control?

A state-of-the-art answer to this problem in the reinforcement learning literature is the E^3-algorithm (Kearns and Singh, 2002) (and its variants/extensions: Kearns

and Koller, 1999; Kakade, Kearns and Langford, 2003; Brafman and Tennenholtz, 2002). These algorithms guarantee that near-optimal performance will be obtained in time polynomial in the number of states of the system. To achieve its performance guarantees, the E^3-family of algorithms demand that we run exploration policies on the unknown system until we have an accurate model for the entire state space (or at least for the "reachable" parts of it). Such exhaustive exploration can take an undesirably long time for complex systems. Moreover, the strong bias towards exploration makes the policies generated by the E^3-family often unacceptable for running on a real system.

In this dissertation, we show that it is sufficient to learn a dynamics model from expert demonstrations (and possibly, in addition, a few autonomous trials which try to perform as well as possible, rather than explore) to attain strong performance guarantees. Our theoretical results guarantee that the control policies we find perform comparably to the expert. Here we evaluate performance on the same task and environment as the expert—technically, we assume the expert and our autonomous controller are acting in the same Markov decision process.

The system identification literature (see, e.g., Ljung, 1986) also considers the problem of learning a dynamics model from data. This literature typically suggests to ensure all modes of the system are being excited sufficiently during data collection. Interestingly, our results suggest that such exhaustive data collection can be unnecessary, as it is sufficient to merely collect expert demonstrations of the task at hand to learn to perform as well as the expert. In exchange for the additional data collection efforts, traditional system identification methods can have additional guarantees: one obtains a dynamics model of sufficient detail to then enable control algorithms to be accompanied by performance guarantees beyond the operating/environmental regime of the expert demonstrations. Examples of such guarantees include robustness to certain perturbations, and global asymptotic stability.

1.3 Model-based reinforcement learning / Optimal control

Even when provided with a formal specification, e.g., in the form of a reward function, and a good dynamics model, it is often computationally expensive to find closed-loop controllers for high dimensional, stochastic domains. In fact, a very large fraction of the optimal control and reinforcement learning literature focuses on exactly this problem.

In the work described in this dissertation, we build upon existing algorithms from the reinforcement learning and optimal control literature to find the optimal control policy once we have a reward function and a dynamics model. We defer the specifics of the algorithms we used to the corresponding experimental sections.

While we typically adapt the optimal control algorithms to our specific problem setting, the key enablers for our results have been the apprenticeship learning algorithms for learning a reward function and a dynamics model. For example, for the challenging problem of autonomous helicopter aerobatics, we repeatedly tried a variety of optimal control algorithms in combination with hand-specified reward functions. However, only once we started using the apprenticeship learning techniques presented in this dissertation in conjunction with optimal control algorithms, did we succeed at having our helicopter fly high-performance aerobatics.

1.4 Overview of experimental results

Our algorithms do not only come with theoretical guarantees, but have also enabled us to solve some previously unsolved real-world control problems: They have enabled a quadruped robot to traverse challenging, previously unseen terrain. They have significantly extended the state-of-the-art in autonomous helicopter flight. Our helicopter has performed by far the most challenging aerobatic maneuvers performed by any autonomous helicopter to date, including maneuvers such as continuous in-place flips, rolls and tic-tocs, which only exceptional expert human pilots can fly. Our aerobatic flight performance is comparable to that of the best human pilots.

We have also applied our algorithms to a variety of other problems, including: Learning to drive according to different driving styles in a highway simulation; Learning to navigate a parking lot in a variety of styles; Learning to fly a trajectory with a fixed-wing in a flight-simulator; Having a remotely controlled (RC) car learn to drive trajectories at high-accuracy.

1.5 Contributions per chapter

We present the following contributions in the respective chapters of this dissertation:

- **Chapter 2**

  For many control tasks, it is challenging to articulate the reward function corresponding to the task at hand. In Chapter 2 we describe how one can leverage expert demonstrations to efficiently address this challenge. In particular, we consider learning in a Markov decision process where we are not explicitly given a reward function, but where instead we can observe an expert demonstrating the task that we want to learn to perform.

  We think of the expert as trying to maximize a reward function that is expressible as a linear combination of known features, and give an algorithm for learning the task demonstrated by the expert. Our algorithm is based on using "inverse reinforcement learning" to try to recover the unknown reward function.

  We show that our algorithm terminates in a small number of iterations, and that even though we may never recover the expert's reward function, the policy output by the algorithm will attain performance close to that of the expert, where here performance is measured with respect to the expert's *unknown* reward function.

  Besides providing theoretical guarantees, we also experimentally evaluated our algorithm on a variety of control tasks. We applied our algorithm to a gridworld simulation, to a highway driving simulation, to a parking lot navigation task, and to a quadruped locomotion problem.

- **Chapter 3**

  For many systems the dynamics is complex and one needs to collect data from the to build a good dynamics model.

  In Chapter 3 we show that, in the apprenticeship learning setting, explicit exploration is not necessary to attain performance guarantees. We show that it is sufficient to learn a dynamics model from expert demonstrations and possibly, in addition, a few autonomous trials which try to perform as well as possible—rather than explicitly explore the state space. Our theoretical results guarantee that the control policies we find perform comparably to the expert. Here we evaluate performance on the same task and environment as the expert—technically, we assume the expert and our autonomous controller are acting in the same Markov decision process.

- **Chapter 4**

  Markov decision processes use a first-order Markov model to capture the dynamics of the system to be controlled. Even when the true system is not first-order Markov with respect to the chosen state space, it is common to approximate the system dynamics using a first-order Markov model. When a first-order Markov model's parameters are estimated from data, the standard maximum likelihood estimator considers only the first-order (single-step) transitions. If the first-order conditional independence assumptions are not satisfied, the higher order transition probabilities may be poorly approximated. Motivated by the problem of learning an MDP's parameters for control, we propose an algorithm for learning a first-order Markov model that explicitly takes into account higher order interactions during training. Our algorithm uses an optimization criterion different from maximum likelihood, and allows us to learn models that capture longer range effects, but without giving up the benefits of using first-order Markov models. Our experimental results also show the new algorithm outperforming conventional maximum likelihood estimation in a number of control problems where the MDP's parameters are estimated from data.

- **Chapter 5**

  In this chapter we describe the parameterization of our helicopter models. We also present an efficient algorithm for (approximately) optimizing the lagged criterion presented in Chapter 4 for continuous state-action space models and apply it to helicopter modeling.

  Our helicopter models are to great extent directly estimated from data, yet, we incorporate well-known effects from physics (such as gravity and intertia) by explicitly encoding them into an acceleration based model. Hence we avoid these effects have to be estimated from data—as was the case in some prior work on helicopter modeling in the machine learning community (e.g., Bagnell & Schneider, 2001; Ng et al., 2004b). We compare our non-linear acceleration based models to the linear models such as learned by CIFER (the industry standard in linear modeling for helicopters), and show that the linear parameterization is often insatisfactory as it makes certain properties of dynamical systems, such as inertia, fundamentally difficult to capture.

- **Chapter 6**

  While a more accurate dynamics model will typically only improve the controllers found when using model-based reinforcement learning (RL) algorithms, a fully accurate dynamics model is not always necessary to find good controllers. E.g., when a human learns to drive a car, they can learn to drive the car reliably without necessarily building up a very detailed dynamics model. Instead, we can learn to drive a car based upon a crude model *and* a small number of real-life trials.

  In Chapter 6, we present an algorithm that requires only an approximate model (of a possibly complex system), and only a small number of real-life trials. The key idea is that a real-life trial together with an approximate model can be sufficient to obtain reasonable policy gradient estimates.

  Our theoretical results show that, under certain assumptions, this algorithm achieves near-optimal performance in the real system, even when the model is

only approximate. Empirical results also demonstrate that—when given only a crude model and a small number of real-life trials—our algorithm can obtain near-optimal performance in the real system.

- **Chapter 7**

  While trajectory following is a fairly common task in robotics, it is often very challenging to specify the desired trajectory. Expert demonstrations of the trajectory are a natural alternative to hand-coding the desired trajectory. However, for challenging tasks, even for experts it can be difficult to demonstrate the desired trajectories nicely. On the other hand, it is often easier to obtain many sub-optimal demonstrations.

  As repeated sub-optimal demonstrations tend to differ in their suboptimalities, together they often encode the intended trajectory. In Chapter 7 we consider the problem of learning an intended trajectory from a small number of demonstrations from a sub-optimal expert. We present an algorithm that (i) extracts the—initially unknown—intended trajectory from the sub-optimal expert's demonstrations and (ii) learns a local model suitable for control along the learned trajectory.

  We apply our algorithm to learning trajectories for helicopter aerobatics.

- **Chapter 8**

  In Chapter 8 we describe the application of the apprenticeship learning methods presented in preceding chapters to the problem of autonomous helicopter flight. They have enabled us to significantly extend the state of the art in autonomous helicopter aerobatics. Our results include the first autonomous in-place flips, in-place rolls, tic-tocs, loops and hurricane, and a complete airshow, which requires autonomous transitions between these and various other maneuvers. Our controllers perform as well, and often even better, than our expert pilot.

  We posted movies of our autonomous helicopter flight results at:

  `http://heli.stanford.edu`.

- **Chapter 9**

  We summarize and discuss our results and then expand on some directions for future work.

1.6 First published appearances of the described contributions

Most contributions described in this thesis have first appeared as various publications. Below are the publications (crudely) corresponding to each of the chapters:

- Chapter 2: Abbeel and Ng (2004); Kolter, Abbeel and Ng (2008a); Abbeel, Dolgov, Ng and Thrun (2008).
- Chapter 3: Abbeel and Ng (2005a).
- Chapter 4: Abbeel and Ng (2005b).
- Chapter 5: Abbeel, Ganapathi and Ng (2006a).
- Chapter 6: Abbeel, Quigley and Ng (2006b).
- Chapter 7: Coates, Abbeel and Ng (2008).
- Chapter 8: Abbeel, Coates, Quigley and Ng (2007); Coates, Abbeel and Ng (2008).

Chapter 2

Apprenticeship Learning via Inverse Reinforcement Learning

One of the key challenges when applying traditional control or reinforcement learning methods is to provide a formal specification of the control task. The optimal control and reinforcement learning formalisms require the specification of a reward (or cost) function defining "goodness" of each possible state. This reward function is often hard to specify. For example, what would be the correct reward function for "driving well"?

In this chapter, we describe how one can leverage expert demonstrations to efficiently address this challenge. In particular, we consider learning in a Markov decision process where we are not explicitly given a reward function, but where instead we can observe an expert demonstrating the task that we want to learn to perform. We think of the expert as trying to maximize a reward function that is expressible as a linear combination of known features, and give an algorithm for learning the task demonstrated by the expert. Our algorithm is based on using "inverse reinforcement learning" to try to recover the unknown reward function.

We show that our algorithm terminates in a small number of iterations, and that even though we may never recover the expert's reward function, the policy output by the algorithm will attain performance close to that of the expert, where here performance is measured with respect to the expert's *unknown* reward function.

Besides providing theoretical guarantees, we also experimentally evaluated our algorithm on a variety of control tasks. We applied our algorithm to a gridworld simulation, to a highway driving simulation, to a parking lot navigation task, and to a quadruped locomotion problem.

2.1 Introduction

Given a sequential decision making problem posed in the Markov decision process (MDP) formalism, a number of standard algorithms exist for finding an optimal or near-optimal policy. In the MDP setting, we typically assume that a reward function is given. Given a reward function and the MDP's state transition probabilities, the value function and optimal policy are exactly determined.

The MDP formalism is useful for many problems because it is often easier to specify the reward function than to directly specify the value function (and/or optimal policy). However, we believe that even the reward function is frequently difficult to specify manually. Consider, for example, the task of highway driving. When driving, we typically trade off many different desiderata, such as maintaining a safe following distance, keeping away from the curb, staying far from any pedestrians, maintaining a reasonable speed, perhaps a slight preference for driving in the middle lane, not changing lanes too often, etc. To specify a reward function for the driving task, we would have to assign a set of weights stating exactly how we would like to trade off these different factors. Despite being able to drive competently, the authors do not believe they can confidently specify a specific reward function for the task of "driving well."[1]

In practice, this means that the reward function is often manually tweaked (cf. reward shaping, Ng et al., 1999) until the desired behavior is obtained. From conversations with engineers in industry and our own experience in applying reinforcement

[1]We note that this is true even though the reward function may often be easy to state in English. For instance, the "true" reward function that we are trying to maximize when driving is, perhaps, our "personal happiness." The practical problem however is how to model this (i.e., our happiness) explicitly as a function of the problems' states, so that a reinforcement learning algorithm can be applied.

learning algorithms to several robots, we believe that, for many problems, the difficulty of manually specifying a reward function represents a significant barrier to the broader applicability of reinforcement learning and optimal control algorithms.

When teaching a young adult to drive, rather than telling them what the reward function is, it is much easier and more natural to demonstrate driving to them, and have them learn from the demonstration. The task of learning from an expert is called *apprenticeship learning* (also learning by watching, imitation learning, or learning from demonstration).

A number of approaches have been proposed for apprenticeship learning in various applications. Most of these methods try to directly mimic the demonstrator by applying a supervised learning algorithm to learn a direct mapping from the states to the actions. This literature is too wide to survey here, but some examples include Sammut et al. (1992); Kuniyoshi et al. (1994); Demiris and Hayes (1994); Amit and Mataric (2002); Pomerleau (1989). One notable exception is given in Atkeson and Schaal (1997). They considered the problem of having a robot arm follow a demonstrated trajectory, and used a reward function that quadratically penalizes deviation from the desired trajectory. Note however, that this method is applicable only to problems where the task is to mimic the expert's trajectory. For highway driving, blindly following the expert's trajectory would not work, because the pattern of traffic encountered is different each time.

Given that the entire field of reinforcement learning is founded on the presupposition that the reward function, rather than the policy or the value function, is the most succinct, robust, and transferable definition of the task, it seems natural to consider an approach to apprenticeship learning whereby the reward function is learned.[2]

The problem of deriving a reward function from observed behavior is referred to as inverse reinforcement learning (Ng and Russell, 2000). In this paper, we assume that the expert is trying (without necessarily succeeding) to optimize an unknown reward

[2]A related idea is also seen in the biomechanics and cognitive science literature, where researchers have pointed out that simple reward functions (usually ones constructed by hand) often suffice to explain complicated behavior (policies). Examples include the minimum jerk principle to explain limb movement in primates (Hogan, 1984), and the minimum torque-change model to explain trajectories in human multi-joint arm movement (Uno et al., 1989). Related examples are also found in economics and some other literatures. (See the discussion in Ng and Russell, 2000.)

function that can be expressed as a linear combination of known "features." Even though we cannot guarantee that our algorithms will correctly recover the expert's true reward function, we show that our algorithm will nonetheless find a policy that performs as well as the expert, where performance is measured with respect to the expert's *unknown* reward function.

Since our initial work on apprenticeship learning via inverse reinforcement learning (Abbeel and Ng, 2004) there has been a substantial body of closely related work, which exploits succinctness of the reward function to efficiently learn to perform a task from expert demonstrations. This body of work includes Kolter et al. (2008a); Ratliff et al. (2006); Ratliff et al. (2007); Neu and Szepesvari (2007); Ramachandran and Amir (2007); Syed and Schapire (2008). In Section 2.6 we discuss relationships with our work.

The work described in this chapter has first appeared in Abbeel and Ng (2004); Kolter, Abbeel and Ng (2008a); Abbeel, Dolgov, Ng and Thrun (2008).

2.2 Preliminaries

A (finite-state) Markov decision process (MDP) is a tuple $(S, \mathcal{A}, T, \gamma, D, R)$, where S denotes a set of states; $\mathcal{A}$ is a set of actions; $T = \{P_{sa}\}$ is a set of state transition probabilities (here, P_{sa} is the state transition distribution upon taking action a in state s); $\gamma \in [0, 1)$ is a discount factor; D is the initial-state distribution, from which the start state s_0 is drawn; and $R : S \mapsto \mathbb{R}$ is the reward function, which we assume to be bounded in absolute value by 1. We let MDP\R denote an MDP without a reward function, i.e., a tuple of the form $(S, \mathcal{A}, T, \gamma, D)$.

We assume that there is some vector of features $\phi : S \to [0, 1]^k$ over states, and that there is some "true" reward function $R^*(s) = w^{*\top}\phi(s)$, where $w^* \in \mathbb{R}^k$.[3] In order to ensure that the rewards are bounded by 1, we also assume $\|w^*\|_1 \leq 1$. In the driving domain, ϕ might be a vector of features indicating the different desiderata in driving that we would like to trade off, such as whether we have just collided with

[3]The case of state-action rewards $R(s, a)$ offers no additional difficulties; using features of the form $\phi : S \times \mathcal{A} \to [0, 1]^k$, and our algorithms still apply straightforwardly.

another car, whether we're driving in the middle lane, and so on. The (unknown) vector w^* specifies the relative weighting between these desiderata.

A (stationary) policy π is a mapping from states to probability distributions over actions. The value of a policy π is

$$\begin{aligned} E_{s_0\sim D}[V^\pi(s_0)] &= E[\sum_{t=0}^{\infty}\gamma^t R(s_t)|\pi] && (2.1)\\ &= E[\sum_{t=0}^{\infty}\gamma^t w^\top\phi(s_t)|\pi] && (2.2)\\ &= w^\top E[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)|\pi] && (2.3)\end{aligned}$$

Here, the expectation is taken with respect to the random state sequence $s_0, s_1, \ldots$ drawn by starting from a state $s_0 \sim D$, and picking actions according to π. We define the expected discounted accumulated feature value vector $\mu(\pi)$, or more succinctly the **feature expectations**, to be

$$\mu(\pi) = E[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)|\pi] \in \mathbb{R}^k. \quad (2.4)$$

Using this notation, the value of a policy may be written $E_{s_0\sim D}[V^\pi(s_0)] = w^\top\mu(\pi)$. Given that the reward R is expressible as a linear combination of the features ϕ, the feature expectations for a given policy π completely determine the expected sum of discounted rewards when acting according to that policy.

Let Π denote the set of stationary policies for an MDP. Given two policies $\pi_1, \pi_2 \in \Pi$, we can construct a new policy π_3 by mixing them together. Specifically, imagine that π_3 operates by flipping a coin with bias λ, and with probability λ picks and always acts according to π_1, and with probability $1-\lambda$ always acts according to π_2. From linearity of expectation, clearly we have that $\mu(\pi_3) = \lambda\mu(\pi_1) + (1-\lambda)\mu(\pi_2)$. Note that the randomization step selecting between π_1 and π_2 occurs only once at the start of a trajectory, and *not* on every step taken in the MDP. More generally, if we have found some set of policies $\pi_1, \ldots, \pi_d$, and want to find a new policy whose

feature expectations vector is a convex combination $\sum_{i=1}^{n} \lambda_i \mu(\pi_i)$ ($\lambda_i \geq 0, \sum_i \lambda_i = 1$) of these policies', then we can do so by mixing together the policies $\pi_1, \ldots, \pi_d$, where the probability of picking π_i is given by λ_i.

We assume access to demonstrations by some expert π_E. Specifically, we assume the ability to observe trajectories (state sequences) generated by the expert starting from $s_0 \sim D$ and taking actions according to π_E. It may be helpful to think of π_E as the optimal policy under the reward function $R^* = w^{*T}\phi$, though we do not require this to hold.

For our algorithm, we will require an estimate of the expert's feature expectations $\mu_E = \mu(\pi_E)$. Specifically, given a set of m trajectories $\{s_0^{(i)}, s_1^{(i)}, \ldots\}_{i=1}^{m}$ generated by the expert, we denote the empirical estimate for μ_E by[4]

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}). \tag{2.5}$$

In the sequel, we also assume access to a reinforcement learning (RL) algorithm that can be used to solve an MDP\R augmented with a reward function $R = w^T\phi$.

2.3 Algorithm

We consider the following problem: Given an MDP\R, a feature mapping ϕ and the expert's feature expectations μ_E, find a policy whose performance is close to that of the expert's, on the *unknown* reward function $R^* = w^{*T}\phi$. To accomplish this, we will find a policy $\tilde{\pi}$ such that $\|\mu(\tilde{\pi}) - \mu_E\|_2 \leq \epsilon$. For such a $\tilde{\pi}$, we would have that

[4]In practice we truncate the trajectories after a finite number H of steps. If $H = H_\epsilon = \log_\gamma(\epsilon(1-\gamma))$ is the ϵ-horizon time, then this introduces at most ϵ error into the approximation.

for any $w \in \mathbb{R}^k$ ($\|w\|_1 \leq 1$),

$$
\begin{aligned}
& |E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi_E] - E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\tilde{\pi}]| && (2.6)\\
&= |w^T \mu(\tilde{\pi}) - w^T \mu_E| \\
&\leq \|w\|_2 \|\mu(\tilde{\pi}) - \mu_E\|_2 \\
&\leq 1 \cdot \epsilon = \epsilon && (2.7)
\end{aligned}
$$

The first inequality follows from the fact that $|x^T y| \leq \|x\|_2 \|y\|_2$, and the second from $\|w\|_2 \leq \|w\|_1 \leq 1$. So the problem is reduced to finding a policy $\tilde{\pi}$ that induces feature expectations $\mu(\tilde{\pi})$ close to μ_E. Our apprenticeship learning algorithm for finding such a policy $\tilde{\pi}$ is as follows:

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute a new "guess" of the reward function by solving the following convex (quadratic) programming problem:

$$
\begin{aligned}
\min_{\lambda,\mu} \quad & \|\mu_E - \mu\|_2 \\
\text{s.t.} \quad & \sum_{j=0}^{i-1} \lambda_j \mu^{(j)} = \mu \\
& \lambda \geq 0 \\
& \sum_{j=0}^{i-1} \lambda_j = 1
\end{aligned}
$$

   Set $t^{(i)} = \|\mu_E - \mu\|_2$, $w^{(i)} = \frac{\mu_E - \mu}{\|\mu_E - \mu\|_2}$.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Upon termination, the policy $\tilde{\pi}$ obtained as a mixture of the policies $\{\pi^{(i)} : i = 0 \ldots n\}$ with weights λ_i attains feature counts μ, which are within ϵ of the expert's feature counts μ_E.

We now examine the algorithm in more detail. On iteration i, we have already found some policies $\pi^{(0)}, \ldots, \pi^{(i-1)}$. The optimization in Step 2 can be viewed as an inverse reinforcement learning step in which we try to guess the reward function being optimized by the expert. Solving the convex optimization problem gives us the feature counts closest to the expert's policy in *reward feature space* amongst feature counts achievable by using a mixture policy of previously found policies $\pi^{(0)}, \ldots, \pi^{(i-1)}$. We then set the guessed reward weights $w^{(i)}$ to be along the line connecting these closest feature counts μ and the expert's feature counts μ_E.

We can also find the same w by solving the following convex (quadratic) programming problem:[5]

$$\begin{aligned} \max_{t,w} \quad & t & (2.8)\\ \text{s.t.} \quad & w^T \mu_E \geq w^T \mu^{(j)} + t, \; j = 0, \ldots, i-1 & (2.9)\\ & \|w\|_2 \leq 1 & (2.10) \end{aligned}$$

From Eqn. (2.9), we see the algorithm is trying to find a reward function $R = w^{(i)\top}\phi$ such that $E_{s_0 \sim D}[V^{\pi_E}(s_0)] \geq E_{s_0 \sim D}[V^{\pi^{(i)}}(s_0)] + t$. I.e., a reward on which the expert does better, by a "margin" of t, than any of the i policies we had found previously. This step is similar to one used in (Ng and Russell, 2000), but unlike the algorithms given there, because of the 2-norm constraint on w it cannot be posed as a linear program (LP), but only as a quadratic program.[6]

Readers familiar with support vector machines (SVMs) will also recognize this optimization as being equivalent to finding the maximum margin hyperplane separating two sets of points. (Vapnik, 1998) The equivalence is obtained by associating

[5] A third alternative formulation, which we will not rely upon, to find $w^{(i)}$ is given by: $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.

[6] Although we previously assumed that the w^* specifying the "true" rewards satisfy $\|w^*\|_1 \leq 1$ (and our theoretical results will use this assumption), we still implement the *algorithm* using $\|w\|_2 \leq 1$, as in Eq. (2.10).

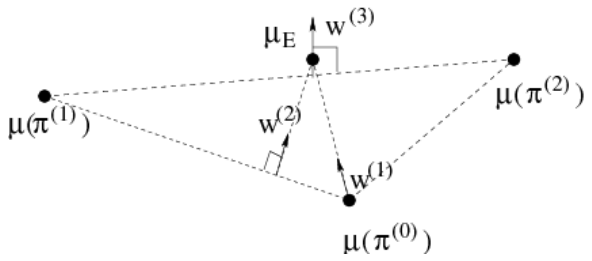


Figure 2.1: Three iterations of the max-margin version of the algorithm.

a label 1 with the expert's feature expectations μ_E, and a label -1 with the feature expectations $\{\mu(\pi^{(j)}) : j = 0..(i-1)\}$. The vector $w^{(i)}$ we want is the unit vector orthogonal to the maximum margin separating hyperplane. So, besides a generic quadratic program (QP) solver, an SVM solver can also be used to find $w^{(i)}$.

In Figure 2.1 we show an example of what the first three iterations of the algorithm could look like geometrically. Shown are several example $\mu(\pi^{(i)})$, and the $w^{(i)}$'s given by different iterations of the algorithm.

Now, suppose the algorithm terminates, with $t^{(n+1)} \leq \epsilon$. (Whether the algorithm terminates is discussed in Section 2.4.) Then directly from the optimization formulation in Step 2.

$$\forall w \text{ with } \|w\|_2 \leq 1 \; \exists i \text{ s.t. } w^T \mu^{(i)} \geq w^T \mu_E - \epsilon. \tag{2.11}$$

Since $\|w^*\|_2 \leq \|w^*\|_1 \leq 1$, this means that there is at least one policy from the set returned by the algorithm, whose performance under R^* is at least as good as the expert's performance minus ϵ. Thus, at this stage, we can ask the agent designer to manually test/examine the policies found by the algorithm, and pick one with acceptable performance. A slight extension of this method ensures that the agent designer has to examine at most $k+1$, rather than all $n+1$, different policies (see footnote 7).

Upon termination, the policy $\tilde{\pi}$ obtained as a mixture of the policies $\{\pi^{(i)} : i = 0 \ldots n\}$ with weights λ_i attains feature counts μ, which are within ϵ of the expert's feature counts μ_E.

If we do not wish to ask for human help to select a policy, alternatively we can

use the policy $\tilde{\pi}$, which is obtained as a mixture of the policies $\{\pi^{(i)} : i = 0 \ldots n\}$ with weights λ_i. It attains feature counts μ, which are within ϵ of the expert's feature counts μ_E. Following our previous discussion (Eq. 2.6-2.7), this policy attains performance near that of the expert's on the unknown reward function.[7]

Note that although we called one step of our algorithm an inverse RL step, our algorithm does not necessarily recover the underlying reward function correctly. The performance guarantees of our algorithm only depend on (approximately) matching the feature expectations, not on recovering the true underlying reward function.

2.3.1 A simpler algorithm

The algorithm described above requires access to a QP (or SVM) solver. It is also possible to change the algorithm so that no QP solver is needed. We will call the previous, QP-based, algorithm the **max-margin** method, and the new algorithm the **projection** method. Briefly, the projection method replaces step 2 of the algorithm with the following:

- Set $\tilde{\mu} = \bar{\mu}^{(i-2)} + \frac{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T (\mu_E - \bar{\mu}^{(i-2)})}{(\mu^{(i-1)} - \bar{\mu}^{(i-2)})^T (\mu^{(i-1)} - \bar{\mu}^{(i-2)})} (\mu^{(i-1)} - \bar{\mu}^{(i-2)})$ (This computes the orthogonal projection of μ_E onto the line through $\bar{\mu}^{(i-2)}$ and $\mu^{(i-1)}$.)
- Set $\bar{\mu}^{(i-1)} = \tilde{\mu}$ if $\tilde{\mu}$ lies in the convex hull of $\bar{\mu}^{(i-2)}$ and $\mu^{(i-1)}$, otherwise set $\bar{\mu}^{(i-1)}$ equal to $\text{argmin}_{x \in \{\bar{\mu}^{(i-2)}, \mu^{(i-1)}\}} \|\mu_E - x\|_2$.
- Set $w^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$
- Set $t^{(i)} = \|\mu_E - \bar{\mu}^{(i-1)}\|_2$

In the first iteration, we also set $w^{(1)} = \mu_E - \mu^{(0)}$ and $\bar{\mu}^{(0)} = \mu^{(0)}$.

n Sections 2.4 and 2.5 we give convergence results for the projection algorithm, and empirically compare it to the max-margin algorithm. An example showing three iterations of the projection method is shown in Figure 2.2.

[7] In k-dimensional space, any point that is a convex combination of a set of N points, with $N > k+1$, can be written as a convex combination of a subset of only $k+1$ points of the original N points (Caratheodory's Theorem, Rockafeller, 1970). Applying this to $\mu = \arg\min_{\mu \in Co\{\mu(\pi^{(i)})\}_{i=0}^n} \|\mu_E - \mu\|_2$ and $\{\mu(\pi^{(i)})\}_{i=0}^n$ we obtain a set of $k+1$ policies which is equally close to the expert's feature expectations and thus have same performance guarantees. (Co denotes convex hull.)

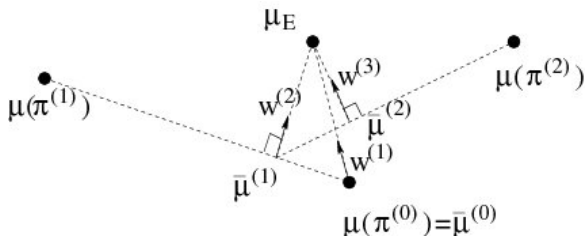


Figure 2.2: Three iterations of the projection version of the algorithm.

2.4 Theoretical results

Most of the results in the previous section were predicated on the assumption that the algorithm terminates with $t \leq \epsilon$. If the algorithm sometimes does not terminate, or if it sometimes takes a very (perhaps exponentially) large number of iterations to terminate, then it would not be useful. In this section we provide convergence guarantees for the algorithm.

The discussion so far has assumed we have the true expert feature expectations μ_E available. In practice this is typically not the case, and we use a Monte Carlo estimate $\hat{\mu}_E$ from n expert demonstrations.

Theorem 1. *Let an MDP\R, features $\phi : S \mapsto [0,1]^k$, and any $\epsilon > 0, \delta > 0$ be given. Suppose the apprenticeship learning algorithm (either max-margin or projection version) is run using an estimate $\hat{\mu}_E$ for μ_E obtained by m Monte Carlo samples. In order to ensure that with probability at least $1-\delta$ the algorithm terminates by satisfying the exit condition $t^{(i)} \leq \frac{\epsilon}{4}$ in Step 3 after at most a number of iterations n satisfying*

$$n \leq \frac{48k}{(1-\gamma)^2\epsilon^2}$$

and outputs a policy $\tilde{\pi}$ so that for any true reward $R^*(s) = w^{*T}\phi(s)$ ($\|w^*\|_1 \leq 1$) we have

$$E[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\tilde{\pi}] \geq E[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi_E] - \epsilon, \tag{2.12}$$

it suffices that

$$m \geq \frac{4k}{(1-\gamma)^2\epsilon^2} \log \frac{2k}{\delta},$$

and that we use an MDP solver that, when given the MDP\R and a reward function $R^{(i)} = \phi^\top w^{(i)}$*, returns a policy* $\pi^{(i)}$ *such that* $\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^{(i)}(s_t)|\pi^{(i)}] \geq \max_\pi \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R^{(i)}(s_t)|\pi] - \frac{\epsilon}{4}$.

We provide a complete proof of Theorem 1 in Appendix A.

In the case where the true reward function R^* does not lie exactly in the span of the basis functions ϕ, the algorithm still enjoys a graceful degradation of performance. Specifically, if $R^*(s) = w^{*\top}\phi(s) + \varepsilon(s)$ for some residual (error) term $\varepsilon(s)$, then our algorithm will have performance that is worse than the expert's by no more than $O(\|\varepsilon\|_\infty)$.

2.5 Experiments

First, we run experiments on gridworlds, illustrating convergence and sampling complexity. Second, we consider the problem of learning different styles for driving a car in a highway driving simulation. Third, we consider the problem of parking lot navigation. Last, we consider the problem of quadruped locomotion across highly challenging terrains.

2.5.1 Gridworld

In our first set of experiments, we used 128 by 128 gridworlds with multiple sparse rewards. The reward is not known to the algorithm, but we can sample trajectories from an expert's (optimal) policy. The agent has four actions to try to move in each of the four compass directions, but with 30% chance an action fails and results in a random move. The grid is divided into non-overlapping regions of 16 by 16 cells; we call these 16x16 regions "macrocells." A small number of the resulting 64 macrocells have positive rewards. For each value of $i = 1, \ldots, 64$, there is one feature $\phi_i(s)$ indicating whether that state s is in macrocell i. Thus, the rewards may be written $R^* = (w^*)^T\phi$. The weights w^* are generated randomly so as to give sparse rewards,

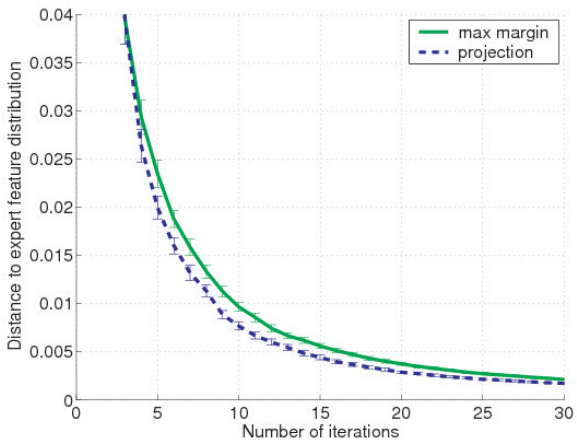


Figure 2.3: A comparison of the convergence speeds of the max-margin and projection versions of the algorithm on a 128x128 grid. Euclidean distance to the expert's feature expectations is plotted as a function of the number of iterations. We rescaled the feature expectations by $(1-\gamma)$ such that they are in $[0,1]^k$. The plot shows averages over 40 runs, with 1 s.e. errorbars.

which leads to fairly interesting/rich optimal policies.[8]

In the basic version, the algorithm is run using the 64-dimensional features ϕ. We also tried a version in which the algorithm knows exactly which macrocells have non-zero rewards (but not their values), so that the dimension of ϕ is reduced to contain only features corresponding to non-zero rewards.

In Figure 2.3, we compare the max-margin and projection versions of the algorithm, when μ_E is known exactly. We plot the margin $t^{(i)}$ (distance to expert's policy) vs. the number of iterations, using all 64 macrocells as features. The expert's policy is the optimal policy with respect to the given MDP. The two algorithms exhibited fairly similar rates of convergence, with the projection version doing slightly better.

The second set of experiments illustrates the performance of the algorithm as we vary the number m of sampled expert trajectories used to estimate μ_E. The performance measure is the value of the best policy in the set output by the algorithm. We ran the algorithm once using all 64 features, and once using only the features that

[8]Details: We used $\gamma = 0.99$, so the expected horizon is of the order of the gridsize. The true reward function was generated as follows: For each macrocell i ($i = 1, \ldots, 64$), with probability 0.9 the reward there is zero ($w_i^* = 0$), and with probability 0.1 a weight w_i^* is sampled uniformly from [0,1]. Finally, w^* is renormalized so that $\|w^*\|_1 = 1$. Instances with fewer than two non-zero entries in w^* are non-interesting and were discarded. The initial state distribution is uniform over all states.

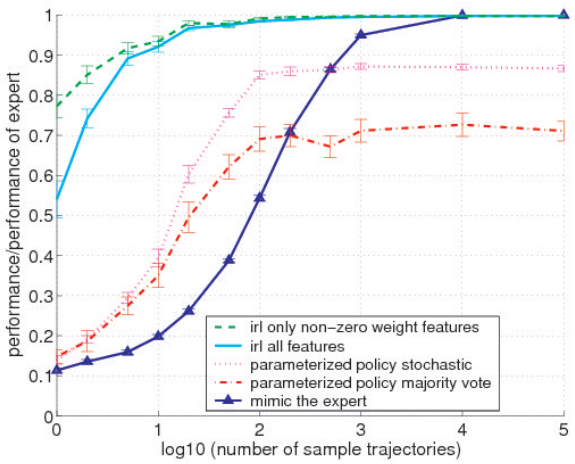


Figure 2.4: Plot of performance vs. number of sampled trajectories from the expert. Averages over 20 instances are plotted, with 1 s.e. errorbars. Note the base-10 logarithm scale on the x-axis. (Best viewed in color.)

truly correspond to non-zero rewards.[9] We also report on the performance of three other simple algorithms. The "mimic the expert" algorithm picks whatever action the expert had taken if it finds itself in a state in which it had previously observed the expert, and picks an action randomly otherwise. The "parameterized policy stochastic" uses a stochastic policy, with the probability of each action constant over each macrocell and set to the empirical frequency observed for the expert in the macrocell. The "parameterized policy majority vote" algorithm takes deterministically the most frequently observed action in the macrocell. Results are shown in Figure 2.4. Using our algorithm, only a few sampled expert trajectories—far fewer than for the other methods—are needed to attain performance approaching that of the expert. (Note log scale on x-axis.)[10] Thus, by learning a compact representation of the reward function, our algorithm significantly outperforms the other methods. We also observe that when the algorithm is told in advance which features have non-zero weight in the true

[9]Note that, as in the text, our apprenticeship learning algorithm assumes the ability to call a reinforcement learning subroutine (in this case, an exact MDP solver using value iteration). In these experiments, we are interested mainly in the question of how many times an expert must demonstrate a task before we learn to perform the same task. In particular, we do not rely on the expert's demonstrations to learn the state transition probabilities.

[10]The parameterized policies never reach the expert's performance, because their policy class is not rich enough. Their restricted policy class is what makes them do better than the "mimic the expert" algorithm initially.

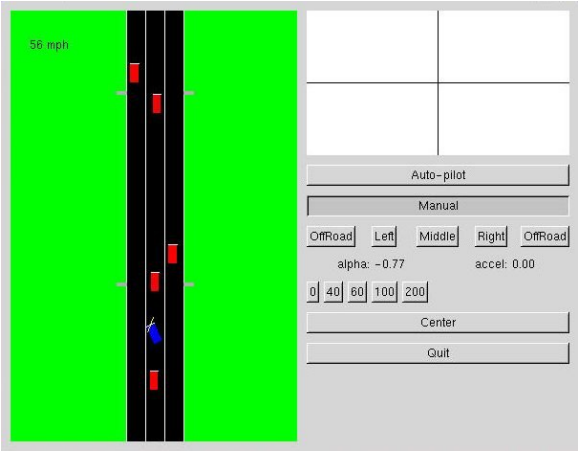


Figure 2.5: Screenshot of driving simulator.

reward function, it is able to learn using fewer expert trajectories.

2.5.2 Car driving simulation

For our second experiment, we implemented a car-driving simulation, and applied apprenticeship learning to try to learn different "driving styles." (See also Abbeel & Ng, 2004.) A screenshot of our simulator is shown in Figure 2.5. We are driving on a highway at 25m/s (56mph), which is faster than all the other cars. The MDP has five different actions, three of which cause the car to steer smoothly to one of the lanes, and two of which cause us to drive off (but parallel to) the road, on either the left or the right side. Because our speed is fixed, if we want to avoid hitting other cars it is sometimes necessary to drive off the road.

The simulation runs at 10Hz, and in the experiments that follow, the expert's features were estimated from a single trajectory of 1200 samples (corresponding to 2 minutes of driving time). There were features indicating what lane the car is currently in (including offroad-left and offroad-right, for a total of five features), and the distance of the closest car in the current lane.[11] Note that a distance of 0 from the nearest car implies a collision. When running the apprenticeship learning algorithm,

[11]More precisely, we used the distance to the single closest car in its current lane, discretized to the nearest car length between -7 to +2, for a total of 10 features.

the step in which reinforcement learning was required was implemented by solving a discretized version of the problem. In all of our experiments, the algorithm was run for 30 iterations, and a policy was selected by inspection (per the discussion in Section 2.3).

We wanted to demonstrate a variety of different driving styles (some corresponding to highly unsafe driving) to see if the algorithm can mimic the same "style" in every instance. We considered five styles:

1. Nice: The highest priority is to avoid collisions with other cars; we also prefer the right lane over the middle lane over the left lane, over driving off-road.
2. Nasty: Hit as many other cars as possible.
3. Right lane nice: Drive in the right lane, but go off-road to avoid hitting cars in the right lane.
4. Right lane nasty: Drive off-road on the right, but get back onto the road to hit cars in the right lane.
5. Middle lane: Drive in the middle lane, ignoring all other cars (thus crashing into all other cars in the middle lane).

After each style was demonstrated to the algorithm (by driving in the simulator for 2 minutes), apprenticeship learning was used to try to find a policy that mimics the demonstrated style. Videos of the demonstrations and of the resulting learned policies are available at

`http://www.cs.stanford.edu/~pabbeel/irl/`.

In every instance, the algorithm was qualitatively able to mimic the demonstrated driving style. Since no "true" reward was ever specified or used in the experiments, we cannot report on the results of the algorithm according to R^*. However, Table 2.1 shows, for each of the five driving styles, the feature expectations of the expert (as estimated from the 2 minute demonstration), and the feature expectations of the learned controller for the more interesting features. Also shown are the weights w used to generate the policy shown. While our theory makes no guarantees about

Table 2.1: Feature expectations of teacher $\hat{\mu}_E$ and of selected/learned policy $\mu(\tilde{\pi})$ (as estimated by Monte Carlo). and weights w corresponding to the reward function that had been used to generate the policy shown. (The table shows 6 of the more interesting features, out of a total of 15 features.)

		Collision	Offroad Left	Left Lane	Middle Lane	Right Lane	Offroad Right
1	$\hat{\mu}_E$	0.0000	0.0000	0.1325	0.2033	0.5983	0.0658
	$\mu(\tilde{\pi})$	0.0001	0.0004	0.0904	0.2287	0.6041	0.0764
	$\tilde{w}$	-0.0767	-0.0439	0.0077	0.0078	0.0318	-0.0035
2	$\hat{\mu}_E$	0.1167	0.0000	0.0633	0.4667	0.4700	0.0000
	$\mu(\tilde{\pi})$	0.1332	0.0000	0.1045	0.3196	0.5759	0.0000
	$\tilde{w}$	0.2340	-0.1098	0.0092	0.0487	0.0576	-0.0056
3	$\hat{\mu}_E$	0.0000	0.0000	0.0000	0.0033	0.7058	0.2908
	$\mu(\tilde{\pi})$	0.0000	0.0000	0.0000	0.0000	0.7447	0.2554
	$\tilde{w}$	-0.1056	-0.0051	-0.0573	-0.0386	0.0929	0.0081
4	$\hat{\mu}_E$	0.0600	0.0000	0.0000	0.0033	0.2908	0.7058
	$\mu(\tilde{\pi})$	0.0569	0.0000	0.0000	0.0000	0.2666	0.7334
	$\tilde{w}$	0.1079	-0.0001	-0.0487	-0.0666	0.0590	0.0564
5	$\hat{\mu}_E$	0.0600	0.0000	0.0000	1.0000	0.0000	0.0000
	$\mu(\tilde{\pi})$	0.0542	0.0000	0.0000	1.0000	0.0000	0.0000
	$\tilde{w}$	0.0094	-0.0108	-0.2765	0.8126	-0.5099	-0.0154

any set of weights w found, we note that the values generally make intuitive sense. For instance, in the first driving style, we see negative rewards for collisions and for driving offroad, and larger positive rewards for driving in the right lane than for the other lanes.

2.5.3 Parking lot navigation

In these experiments, we consider the problem of navigating a robotic car in a parking lot. In this setting, a policy corresponds to a path. Our path-planning algorithm is based on the algorithm used by the Stanford racing team in the DARPA Urban Challenge (Montemerlo et al., 2008).[12] We have extended the existing algorithm by

[12]Our apprenticeship learning algorithms were *not* used for the parking lot navigation during the Urban Challenge race. We conducted all our experiments *after* the Urban Challenge race.

introducing several new cost potentials, which allow us to model a wide range of natural driving styles.

We define the state of the vehicle as $\langle \mathbf{x}, \theta, d \rangle$, where $\mathbf{x} = \langle x, y \rangle$ is the position of the vehicle, θ is its orientation, and $\delta = \{0, 1\}$ determines the direction of motion: forward ($\delta = 0$) or backwards ($\delta = 1$). Further, assume that the network of road lanes is given as a directed graph $\mathcal{G} = \langle V, E \rangle$, and let α_E be the angle of edge E. We define a distance between a point $\mathbf{x}$ and the graph $\mathcal{G}$:

$$\mathcal{D}(\mathbf{x}, \mathcal{G}) = \min_E \mathcal{D}(E, \mathbf{x}),$$

where $\mathcal{D}(E, \mathbf{x})$ is the 2D Euclidean distance between a point and a line segment. Also, let us define a distance between an *oriented* point $\langle \mathbf{x}, \theta \rangle$ and the graph $\mathcal{G}$:

$$\mathcal{D}(\mathbf{x}, \theta, \mathcal{G}) = \min_{\{E : |\alpha_E - \theta| < \alpha_{min}\}} \mathcal{D}(E, \mathbf{x}),$$

i.e., the distance to the nearest edge whose angle is close—within α_{min}—to the heading of the car θ. Further, define an indicator function $R(s)$, where $R(s) = 1$ if the car is on the road, i.e., the 2D euclidean distance between $\mathbf{x}_i$ and $\mathcal{G}$ is below a given threshold: $R(s) = 1 \iff \mathcal{D}(\mathbf{x}, \mathcal{G}) < \mathcal{D}_{road}$. Finally, let $\alpha_i = \alpha\big(\arg\min_E(\mathcal{D}(E, \mathbf{x}_i))\big)$ be the angle of the edge E nearest to the trajectory point $\mathbf{x}_i$.

The objective of the path planner is to minimize the following potential defined

over a trajectory $\mathbf{s} = \{\langle \mathbf{x}_i, \theta_i, \delta_i \rangle\}$:[13]

$$\begin{aligned} & w_{fwd} \sum_{i:i>1,\delta_i=0} \|\mathbf{x}_i - \mathbf{x}_{i-1}\| + \\ & w_{rev} \sum_{i:i>1,\delta_i=1} \|\mathbf{x}_i - \mathbf{x}_{i-1}\| + w_{sw} \sum_{i:\delta_i \neq \delta_{i-1}} 1 + \\ & w_{road} \sum_{i:R(s_i)=0} \|\mathbf{x}_i - \mathbf{x}_{i-1}\| + w_{lane} \sum_i \mathcal{D}(\mathbf{x}_i, \theta_i, \mathcal{G}) + \\ & w_{dir} \sum_i \sin^2\left(2(\theta_i - \alpha_i)\right) + w_{curv} \sum_{i>1,i<|\mathbf{s}|} (\Delta\mathbf{x}_{i+1} - \Delta\mathbf{x}_i)^2, \end{aligned} \tag{2.13}$$

where $\Delta\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_{i-1}$. The terms above respectively correspond to: (1) length of trajectory driven forward, (2) length of trajectory drive in reverse, (3) number of times the direction of motion switches, (4) length of trajectory driven off-road, (5) an aggregate distance of the trajectory to the road-lane graph, (6) a measure of misalignment of trajectory and the principal directions of the parking lot, and (7) a measure of smoothness of the trajectory.

The path-planning problem defined above is a complex continuous-coordinate optimization program with multiple local minima. For computational reasons, we therefore follow a two-phase approach. The first phase performs an approximate discrete global search that finds a solution in a neighborhood of the global optimum; the second phase then fine-tunes the solution in continuous coordinates.

Our first phase uses a variant of A* search with a discrete set of control actions, applied to the 4-dimensional kinematic state of the vehicle defined above. As this phase uses a highly discretized set of control actions, it cannot cleanly accommodate the potential terms in Eqn. 2.13 that correspond to local properties of the trajectory (smoothness, alignment). As such, the first phase focuses on a subset of features that affect global behavior. The local features are used in the subsequent second phase of

[13] In practice, the potential will also contain terms corresponding to hard constraints such as collision avoidance, minimum turning radius of the vehicle, etc. Since these constraints must be satisfied regardless of driving style, we fix their weights at large values (orders of magnitude higher than other terms) and do not include them in learning. See a more thorough description of the core of our path-planning algorithm (Dolgov et al., 2008) for details on modeling and implementing such constraints in optimization.

the optimization algorithm.

The main components that define the behavior of A* are the cost of a partial solution and the cost-to-go heuristics. The heuristics are described in (Dolgov et al., 2008). The cost function is defined by the terms of the potential in Eqn. 2.13 that correspond to the global features with the following weights: $\langle w_{fwd}, w_{rev}, w_{sw}, w_{road}, w_{lane} \rangle$.

Due to coarse discretization used in our global search and for computational reasons, we replace the continuous-coordinate version of the lane-attraction potential with a discrete version similar to the on-road potential. Let us define an indicator function $L(s) = 1$ if the car is in the correct lane, i.e., the distance between the (oriented) car and the lane graph $\mathcal{G}$ is below a given threshold: $L(s) = 1 \iff \mathcal{D}(\mathbf{x}, \theta, \mathcal{G}) < \mathcal{D}_{lane}$.

The lane-keeping potential is approximated as follows:

$$w'_{lane} \sum_{i:i>1, L(s_i)=0} \|\mathbf{x}_i - \mathbf{x}_{i-1}\|,$$

i.e., this term computes a (weighted) length of the path driven out of lane.

For computational reasons, the global A* uses a highly discretized set of control actions, leading to paths that are suboptimal. The second phase of our algorithm improves the quality of the solution by using conjugate-gradient, a very efficient numerical continuous-coordinate optimization technique.

The input to the smoother is the trajectory produced by A* as defined in the previous section. Since the global behavior of the trajectory is already determined, the global features of the potential in Eqn. 2.13 are meaningless in the second phase of our algorithm, since it only performs local adjustment.

The second phase therefore uses the potential terms corresponding to the weights $\langle w_{dir}, w_{curv}, w_{lane} \rangle$, and the optimization is performed using conjugate-gradient descent. The implementation of conjugate-gradient requires a gradient of the objective function, which can be computed analytically for all of these terms.

For our experimental evaluation, we used the Stanford Racing Team's robotic vehicle, Junior (Figure 2.6), which is equipped with several sensors modalities (RADAR, LIDAR, cameras) and a high-precision GPS+IMU system. (See Montemerlo et al, 2008,

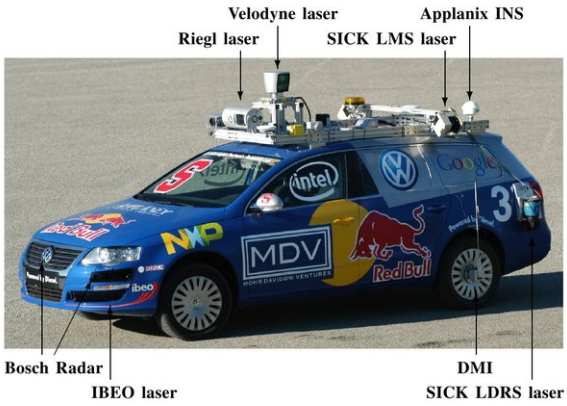


Figure 2.6: All data used in this work was gathered using the Stanford Racing Team's robotic vehicle, Junior. Junior is equipped with several LIDAR and RADAR units, and a high-accuracy inertial measurement system. The training data for the algorithm described in this paper was obtained by manually driving Junior and recording it sensor readings.

for a detailed description of Junior.) For the purposes of our experiments, the autonomous-driving capabilities of the vehicle were not used, rather the car was driven by a human to collect training data. During such data-collection runs, we logged the messages from the pose-estimation GPS+IMU system as well as the messages from the car's 3D LIDAR, which allowed us to later reproduce the exact obstacle maps of the environment as well as the precise trajectories that were driven.

We asked a human driver to navigate a parking lot using three very distinct driving styles:

- "Nice": we tell the driver to drive in the right lane whenever safely possible.
- "Sloppy": we tell the driver it is okay to deviate from the standard lanes. We also tell the driver to only use forward driving.
- "Backward": we allow the driver to drive backward, but only when it makes for a shorter path to the goal.

For each driving style, we collected five demonstrations and ran our learning algorithm five times: every time we learn from four of the demonstrations, and then evaluate performance on the left-out fifth demonstration.

We used the following variation on the max-margin version of the algorithm for "guessing" the cost function in each iteration:

$$\begin{aligned} \min_{w,x} \quad & \|w\|_2^2 \\ \text{s.t.} \quad & \mu = \sum_i x_i \mu^{(i)}; \quad x \geq 0; \\ & \sum_i x_i = 1; \quad w \geq 0 \\ & w \geq \mu - \mu_E; \quad w \in \mathcal{W} \end{aligned}$$

By contrast, in the formulation in Section 2.3 we did not have the last three constraints, but just had a single constraint instead $w = \mu - \mu_E$. The constraints $w \geq 0, w \geq \mu - \mu_E$ encode the fact that we know the weights are positive, and the contributions of the various potential terms to the distance are non-zero only when the expert is outperforming the current best path $\mu^{(i)}$. Similar to Syed and Schapire (2008), this allows us to learn to outperform the expert. By contrast in our original formulation (Section 2.3) the sign of each potential term is considered unknown, hence the algorithm learns to "match" the expert's behaviour, rather than learning to perform equally well or better. The constraint $w \in \mathcal{W}$ allows us to encode additional prior information. In our experiments we encode the prior information that the weight for the potential term corresponding to backward driving has to be at least as high as the one for forward driving.

When the algorithm exits, we have that $\|\mu - \mu_E\| \leq \|w\| \leq \epsilon$. Hence, when stochastically choosing (according to x) between the paths found throughout the iterations of the algorithm, we can perform as well as the expert up to some accuracy ϵ. To generalize to a new setting, we can correspondingly stochastically choose between the weight vectors $\{w^{(0)}, \ldots, w^{(i)}\}$ according to x, and then solve the resulting path-planning problem. In practice, we inspected the training set and we chose the weight vector that resulted in the path "closest" to the expert path as measured in cost (reward) feature space.

Figures 2.7, 2.8, 2.9 show the nice expert demonstrations and the nice autonomous navigation results, the sloppy expert demonstrations and the sloppy autonomous

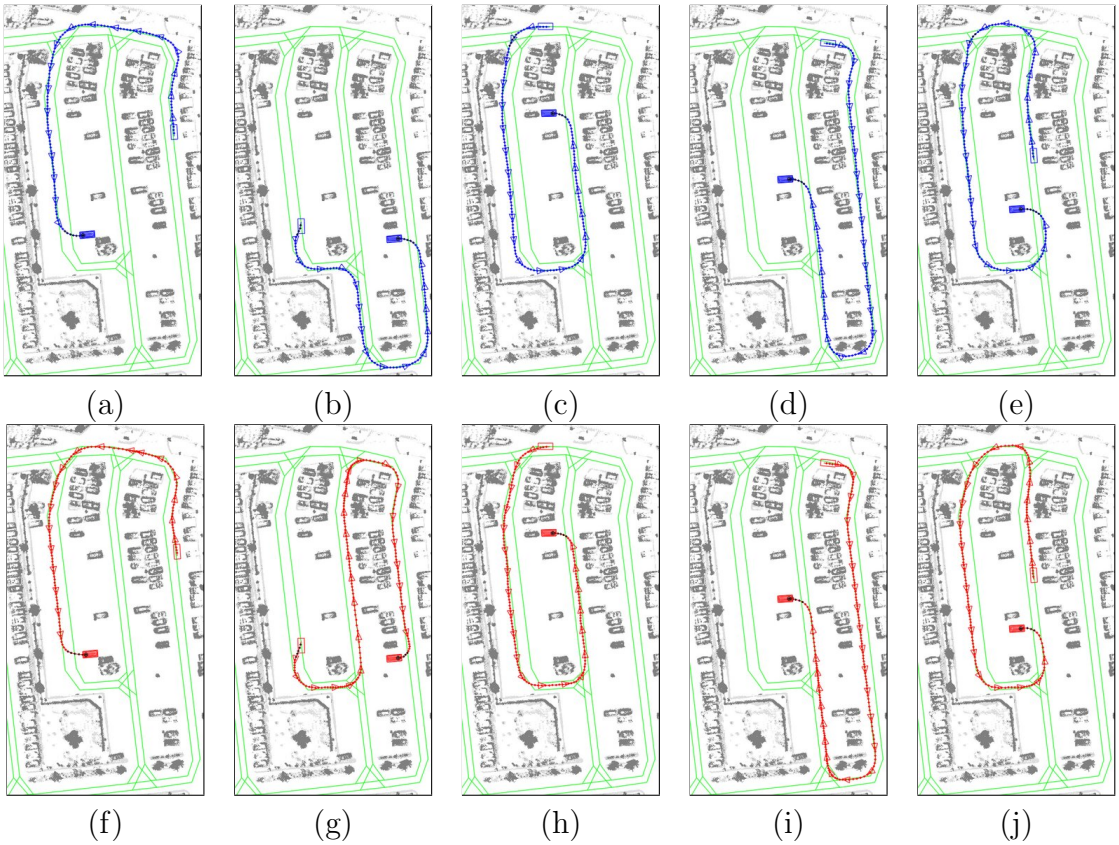


Figure 2.7: "Nice" parking lot navigation driving: expert demonstrations (a-e); trajectories found by learning on four demonstrations, and testing on the fifth (f-j). (Best viewed in color. See text for details.)

navigation results, and the backward expert demonstrations and the backward autonomous navigations results respectively.

The initial state is shown as a car outline, and the goal state is a shaded rectangle. Gray objects are obstacles; the initial state of the car is denoted by a single rectangle; the goal state is denoted by several concentric rectangles; the path's x-y coordinates are shown by a colored line with dots according to the planner's time granularity. Regularly spaced (in time), we overlay a triangle over the path to show the car's heading. The colored dots are replaced by black dots whenever the car is not in its lane ($L(\mathbf{s}) = 0$). When the car goes "off-road" ($R(\mathbf{s}) = 0$), we use larger-sized black circles. The green lines show the graph of driving lanes $\mathcal{G}$. Expert demonstrations are shown in blue, while trajectories produced by our path planner are shown in red.

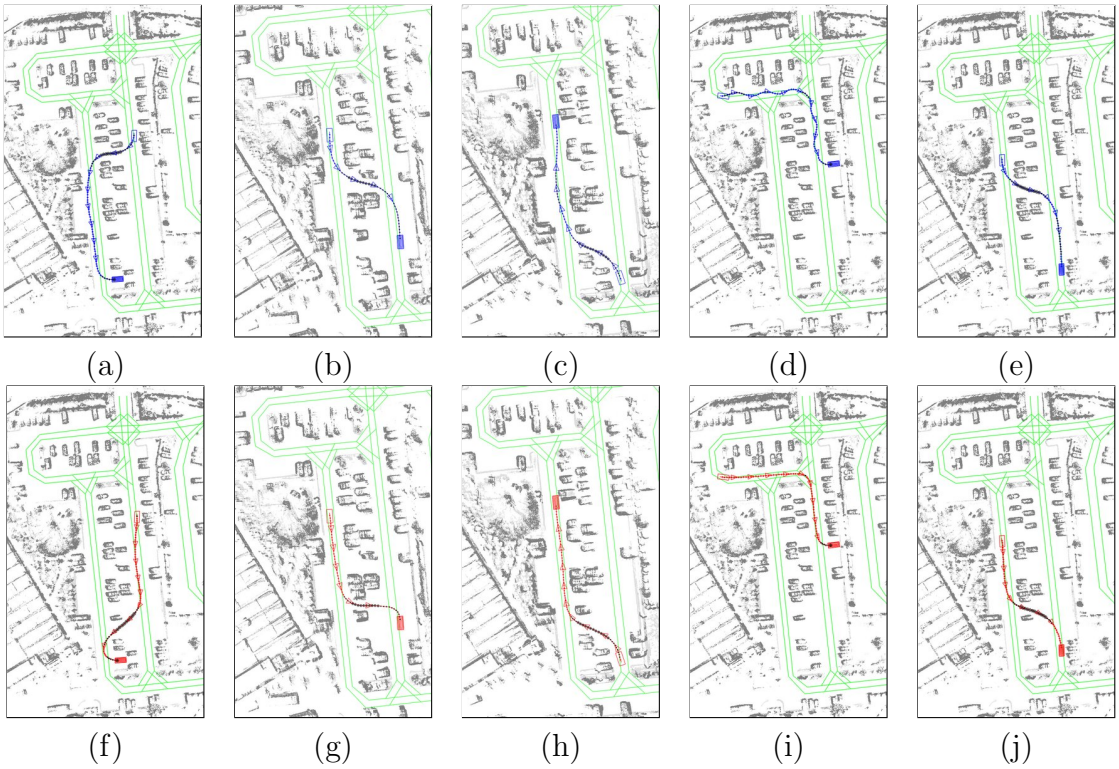


Figure 2.8: "Sloppy" parking lot navigation driving: expert demonstrations (a-e); trajectories found by learning on four demonstrations, and testing on the fifth (f-j). (Best viewed in color. See text for details.)

Inspecting the figures, we note that the learned navigation styles are very similar to the expert's styles. For example, whenever learning from a subset of four nice demonstrations, it learns to keep the right-lane whenever possible. Whenever learning from a subset of four backward demonstrations, it learns that backward driving is allowed to make a shortcut, and successfully executes a shortcut on the left-out fifth navigation task. When learning from the sloppy driver, it successfully learns to make a shortcut through parking space whenever applicable. Interestingly, we learn similarity at the level of the cost terms. E.g., when learning to cut across, it might cut across at a different geographical location than the expert, since the geographical location of the shortcut does not contribute to the cost function.

Table 2.4 gives a quantitative evaluation of our experiments. For each of the 15 learning/testing experiments, we report the empirical feature counts of the expert and

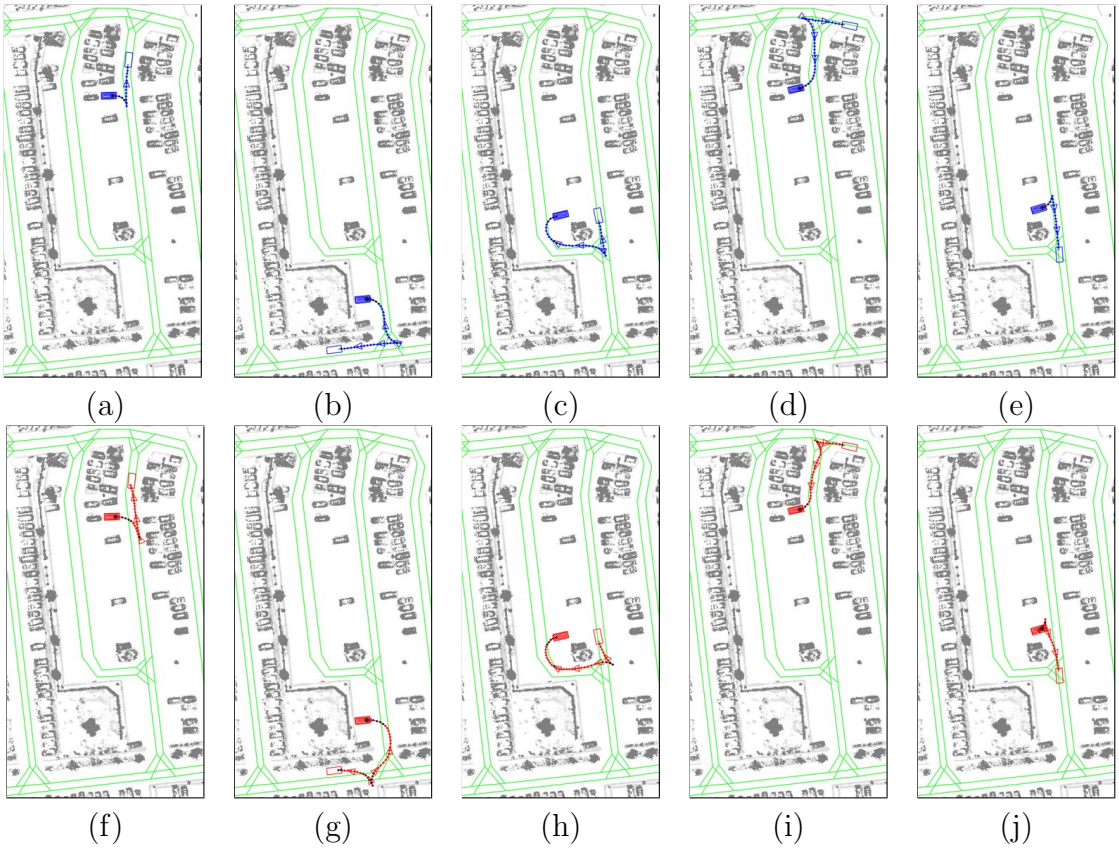


Figure 2.9: "Backward" parking lot navigation driving: expert demonstrations (a-e); trajectories found by learning on four demonstrations, and testing on the fifth (f-j). (Best viewed in color. See text for details.)

the learned planner on the training data, the feature counts of the learned planner on the test data and the learned weight vector. Inspecting the table, we note that indeed, our algorithm finds a set of weights such that both at training and test time the cumulative values of the cost functions are close to those obtained by the expert. Looking more closely at the weight functions learned, we observe the relative weightings for different driving qualitatively matches our intuition about these styles. For the nice driver the penalty for going backward, off-lane, or off-road is much higher than for the other two styles. The backward driving styles has a cost for going backwards that is as low as the cost for going forward. (Consistent with our constrains on the weights: $w \in \mathcal{W}$, which enforces that backward driving is at least as expensive as forward driving. This captures the fact we only want to learn about

Table 2.2: Detailed experimental results for the *nice* navigation style. (See text for details.)

		fwd	rev	sw	$road$	$lane'$	$curv$	dir	$lane$
Nice 1	μ_E (train)	160.82	0	0	1.25	9.0000	1.33	51.69	25.48
	μ (train)	155.32	0	0	1.25	9.25	1.35	49.85	15.37
	μ_E (test)	140.78	0	0	1.00	7.00	0.83	40.28	18.43
	μ (test)	136.95	0	0	1.00	6.00	1.20	38.70	4.13
	w	1.00	96.85	2.44	4.40	24.40	100.00	1.36	2.00
Nice 2	μ_E (train)	165.48	0	0	1.25	8.25	1.16	47.46	24.25
	μ (train)	160.84	0	0	1.25	6.75	1.15	43.89	11.51
	μ_E (test)	122.15	0	0	1.00	10.00	1.49	57.20	23.38
	μ (test)	183.94	0	0	1.00	10.00	1.72	56.13	13.80
	w	1.00	94.20	2.37	10.44	31.80	100.00	2.00	1.67
Nice 3	μ_E (train)	155.86	0	0	1.25	8.25	1.2611	50.53	21.60
	μ (train)	150.65	0	0	1.25	9.50	0.87	41.93	19.76
	μ_E (test)	160.62	0	0	1.00	10.00	1.10	44.95	33.97
	μ (test)	155.72	0	0	1.00	6.00	0.82	43.69	22.15
	w	1.00	83.16	2.40	17.37	52.69	100.00	2.00	0.36
Nice 4	μ_E (train)	150.71	0	0	1.00	9.25	1.18	52.91	24.74
	μ (train)	142.64	0	0	1.25	9.25	1.41	49.37	12.00
	μ_E (test)	181.23	0	0	2.00	6.00	1.41	35.40	21.38
	μ (test)	185.35	0	0	2.00	5.00	1.30	34.13	10.47
	w	1.0000	83.94	1.95	19.50	50.70	100.00	2.00	1.98
Nice 5	μ_E (train)	151.20	0	0	1.25	8.25	1.21	44.45	24.29
	μ (train)	147.06	0	0	1.25	9.75	1.47	41.91	13.07
	μ_E (test)	179.30	0	0	1.00	10.00	1.31	69.23	23.20
	μ (test)	175.43	0	0	1.00	5.00	1.43	55.88	10.89
	w	1.00	57.45	11.57	1.00	81.00	100.00	2.00	1.74

backward driving as a way to make a shortcut, not as a default driving style.) Similar observations hold for the weights for other features. We also note the weight vector entries are fairly consistent over different training runs.

2.5.4 Quadruped locomotion

In this section we consider the task of navigating a quadruped robot (shown in Figure 2.10(a)) over challenging, irregular terrains as the one shown in Figure 2.10(b,c). The quadruped robot is designed and built by Boston Dynamics, Inc. The robot consists of 12 independently actuated servo motors, three on each leg, with two at the hip and one at the knee. It is equipped with an internal IMU and foot force sensors.

Table 2.3: Detailed experimental results for the *sloppy* navigation style. (See text for details.)

		fwd	rev	sw	$road$	$lane'$	$curv$	dir	$lane$
Sloppy 1	μ_E (train)	61.79	0	0	8.50	29.50	0.61	41.05	32.08
	μ (train)	61.88	0	0	8.50	19.00	0.25	33.87	15.48
	μ_E (test)	78.77	0	0	10.00	28.00	0.77	35.56	25.14
	μ (test)	72.00	0	0	11.00	27.00	0.34	54.26	15.76
	w	1.00	1.00	0.03	2.75	0.03	100.00	0.10	0.10
Sloppy 2	μ_E (train)	68.82	0	0	8.50	28.25	0.74	40.12	31.77
	μ (train)	66.36	0	0	9.50	19.75	0.38	22.49	19.66
	μ_E (test)	50.64	0	0	10.00	33.00	0.28	39.29	26.36
	μ (test)	53.83	0	0	10.00	25.00	0.36	22.34	16.61
	w	1.00	2.00	2.00	2.00	0.00	100.00	2.00	0.02
Sloppy 3	μ_E (train)	65.18	0	0	8.00	28.25	0.75	39.13	31.54
	μ (train)	63.50	0	0	8.50	19.50	0.29	37.56	14.59
	μ_E (test)	65.20	0	0	12.00	33.00	0.23	43.25	27.29
	μ (test)	65.23	0	0	13.00	26.00	0.13	41.88	16.11
	w	1.00	2.00	2.00	2.00	0.00	100.00	0.10	0.10
Sloppy 4	μ_E (train)	63.26	0	0	10.75	31.75	0.3912	39.39	26.80
	μ (train)	62.25	0	0	11.50	24.50	0.38	23.28	16.17
	μ_E (test)	72.91	0	0	1.00	19.00	1.66	42.19	46.26
	μ (test)	70.43	0	0	1.00	7.00	0.35	19.16	30.58
	w	1.00	2.00	2.00	2.00	0.00	100.00	2.00	0.02
Sloppy 5	μ_E (train)	66.88	0	0	8.25	28.25	0.73	40.07	31.26
	μ (train)	65.24	0	0	8.75	21.00	0.29	36.97	16.63
	μ_E (test)	58.42	0	0	11.00	33.00	0.29	39.47	28.40
	μ (test)	58.28	0	0	12.00	20.00	0.14	44.21	7.97
	w	1.00	2.00	2.00	2.00	0.00	100.00	0.10	0.10

Table 2.4: Detailed experimental results for the *backward* navigation style. (See text for details.)

		fwd	rev	sw	$road$	$lane'$	$curv$	dir	$lane$
Backward 1	μ_E (train)	20.00	157.70	1.50	1.00	5.75	4.81	16.26	17.07
	μ (train)	26.63	124.04	2.75	1.00	8.50	12.37	49.90	38.59
	μ_E (test)	6.19	134.95	2.00	1.00	5.00	4.57	7.64	12.02
	μ (test)	12.88	187.65	4.00	1.00	7.00	15.46	63.71	32.38
	w	1.00	1.00	0.11	0.64	2.04	100.00	2.00	1.31
Backward 2	μ_E (train)	16.57	141.54	1.50	0.75	5.25	4.76	14.41	13.75
	μ (train)	23.28	84.54	2.25	0.75	12.00	10.92	36.40	28.20
	μ_E (test)	19.88	199.55	2.00	2.00	7.00	4.78	15.04	25.33
	μ (test)	28.88	139.88	4.00	2.00	19.00	18.40	100.67	68.68
	w	1.00	1.00	0.49	0.36	1.35	100.00	2.00	1.45
Backward 3	μ_E (train)	14.40	162.32	1.75	1.25	5.50	4.64	12.24	16.81
	μ (train)	26.73	74.76	5.00	1.25	16.00	13.30	89.27	87.54
	μ_E (test)	28.55	116.44	1.00	0	6.00	5.24	23.70	13.06
	μ (test)	31.16	89.07	1.00	0	11.00	4.70	28.06	16.95
	w	1.00	1.00	0.04	0.15	0.95	100.00	2.00	1.81
Backward 4	μ_E (train)	14.96	157.20	1.50	1.00	5.50	4.76	13.13	14.19
	μ (train)	23.51	119.31	3.75	1.00	13.00	10.94	64.23	39.85
	μ_E (test)	26.32	136.93	2.00	1.00	6.00	4.78	20.16	23.53
	μ (test)	24.95	99.22	2.00	1.00	6.00	8.89	35.61	29.10
	w	1.00	1.00	0.07	0.61	1.70	100.00	2.00	1.12
Backward 5	μ_E (train)	20.24	146.97	1.75	1.00	6.00	4.84	16.64	18.48
	μ (train)	21.85	87.20	2.00	3.50	16.50	5.45	28.05	21.35
	μ_E (test)	5.21	177.86	1.00	1.00	4.00	4.45	6.13	6.38
	μ (test)	4.24	186.81	3.00	3.00	5.00	15.78	16.51	30.04
	w	1.00	1.00	0.15	1.55	6.50	100.00	2.00	0.12

We estimate the robot's state using a motion capture system that tracks reflective markers on the robot's body. We perform all computation on a desktop computer, and send commands to the robot via a wireless connection.

(a) (b) (c)

Figure 2.10: (a) LittleDog robot, designed and built by Boston Dynamics, Inc. (b) Typical terrain. (c) Height map of the depicted terrain. (Black = 0cm altitude, white = 12cm altitude.)

In a naive approach, the dimensionality of the state space is prohibitively large: the robot has 12 independently actuated joints, and the state must also specify the current three-dimensional position and orientation of the robot, leading to an 18-dimensional state space that is well beyond the capabilities of standard RL algorithms. Fortunately, this control task succumbs very naturally to a hierarchical decomposition. Our high-level controller is a *body path planner*, that plans an approximate trajectory for the robot's center of mass over the terrain; our low-level controller is a *footstep planner* that, given a path for the robot's center, plans a set of footsteps that follow this path. Finally, there is a joint level controller which plans the joint movements to achieve these footsteps. A more complete description of the control system is outside the scope of this thesis and can be found in Kolter et al. (2008b). Here we focus on finding a reward function both for the high-level controller and for the low-level controller that results in good performance on the task of traversing previously unseen, irregular terrains.

The footstep planner uses a reward function that specifies the relative trade-off between several different features of the robot's state, including (i) several features capturing the roughness and slope of the terrain at several different spatial scales

around the robot's feet, (ii) distance of the foot location from the robot's desired center, (iii) the area and inradius of the support triangle formed by the three stationary feet, and other similar features. Kinematic feasibility is required for all candidate foot locations and collision of the legs with obstacles is forbidden. To form the high-level cost, we aggregate features from the footstep planner. In particular, for each foot we consider all the footstep features within a 3 cm radius of the foot's "home" position (the desired position of the foot relative to the center of mass in the absence of all other discriminating features), and aggregate these features to form the features for the body path planner. While this is an approximation, we found that it performed very well in practice, possibly due to its ability to account for stochasticity of the domain. After forming the cost function for both levels, we used value iteration to find the optimal policy for the body path planner, and a five-step lookahead receding horizon search to find a good set of footsteps for the footstep planner.

As we use a hierarchical reinforcement learning algorithm, we adapted our apprenticeship learning algorithms to match this hierarchical structure. In particular, we adapted them to take demonstrations at both levels of the hierarchy—a body path, and a preference between footsteps. The full details fall outside the scope of this thesis and can be found in Kolter et al. (2008a).

All experiments were carried out on two terrains: a relatively easy terrain for training, and a significantly more challenging terrain for testing. To give advice at the high level, we specified complete body trajectories for the robot's center of mass, as shown in Figure 2.11(a). To give advice for the low level we looked for situations in which the robot stepped in a suboptimal location, and then indicated the correct greedy foot placement, as shown in Figure 2.11(b). The entire training set consisted of a single high-level path demonstration across the training terrain, and 20 low-level footstep demonstrations on this terrain; it took about 10 minutes to collect the data.

Even from this small amount of training data, the learned system achieved excellent performance, not only on the training board, but also on the much more difficult testing board. Figure 2.12 shows snapshots of the quadruped crossing the testing board. Figure 2.13 shows the resulting footsteps taken for each of the different types of constraints, which shows a very large qualitative difference between the footsteps

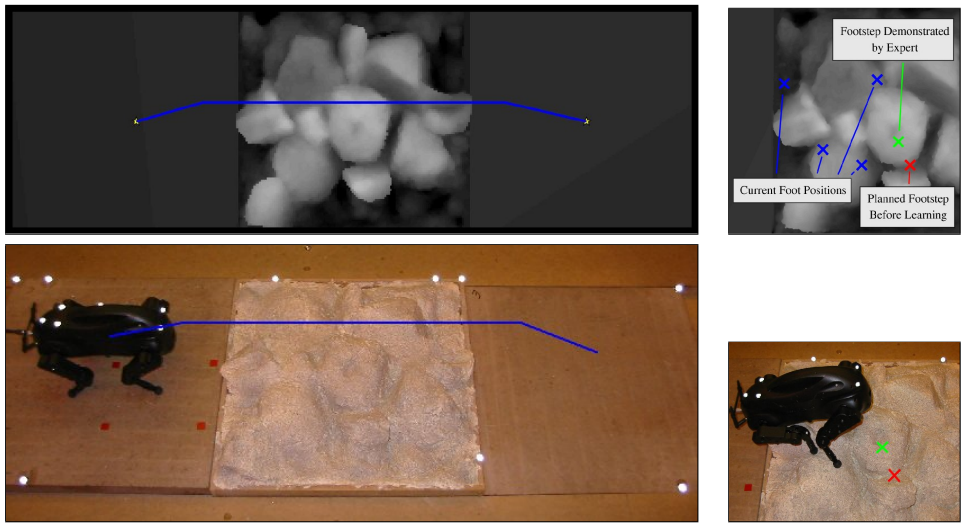


Figure 2.11: (a) High-level (path) expert demonstration. (b) Low-level (footstep) expert demonstration.

Figure 2.12: Snapshots of quadruped while traversing the testing terrain.

chosen before and after training. Table 2.5 shows the crossing times when taking into account different subsets of the demonstrations. As shown, he HAL algorithm outperforms all the intermediate methods. Using only footstep demonstrations does quite well on the training board, but on the testing board the lack of high-level training leads the robot to take a very roundabout route, and it performs much worse. The quadruped fails at crossing the testing terrain when learning from the path-level demonstration only or when not learning at all.

Videos of the results in this section are available at

`http://www.cs.stanford.edu/~pabbeel/quadruped-thesis`.

Prior to undertaking our apprenticeship learning work, we invested several weeks

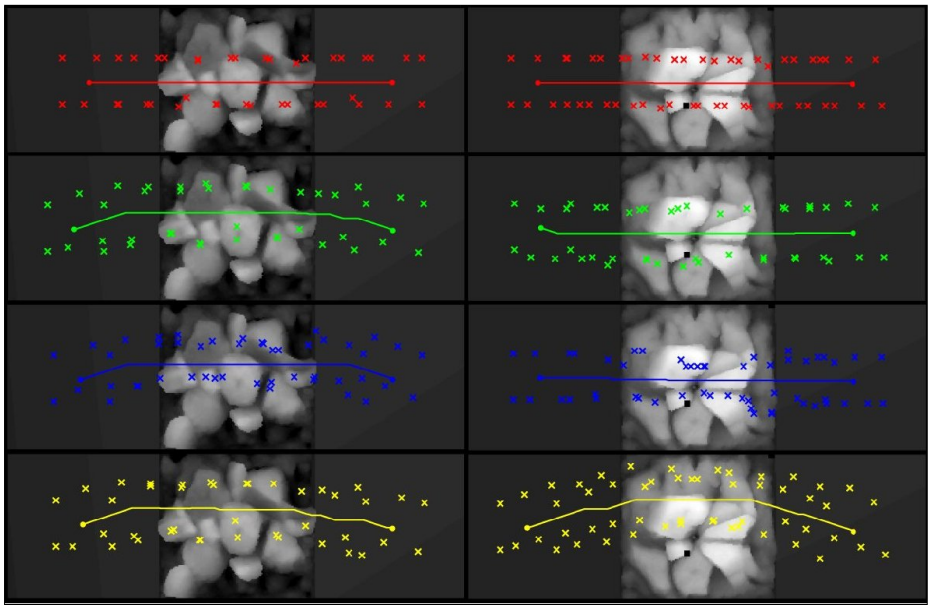


Figure 2.13: Body and footstep plans for different constraints on the training (left) and testing (right) terrains: (Red) No Learning, (Green) HAL, (Blue) Path Only, (Yellow) Footstep Only. (Best viewed in color.)

		HAL	Feet Only	Path Only	No Learning
Training	Time (sec)	31.03	33.46	—	40.25
Testing	Time (sec)	35.25	45.70	—	—

Table 2.5: Execution times for different constraints on training and testing terrains. Dashes indicate that the robot fell over and did not reach the goal.

attempting to hand-tune controller capable of picking good footsteps across challenging terrain. However, none of our previous efforts could significantly outperform the controller presented here, learned from about 10 minutes worth of data, and many of our previous efforts performed substantially worse.

2.6 Related work

Both the max-margin and the projection algorithm iteratively solve the following optimization problem:

$$\min_{\tilde{\pi}\in\tilde{\Pi}} \|\mu(\tilde{\pi}) - \mu_E\|_2^2. \tag{2.14}$$

Here $\tilde{\Pi}$ is the set of mixtures of stationary policies of the MDP. Recall, as described in Section 2.2, when executing a mixture of the policies $\pi_1, \ldots, \pi_d$ with mixture weights $p_1, \ldots, p_d$, we do so by randomly choosing a single policy (according to the mixture weights p) and then adhering to that policy from then on.

Syed and Schapire (2008) consider a very similar setting to ours. Namely, they also assume an MDP\R, a set of features $\phi(\cdot)$ and expert demonstrations are given. Additionally they assume all the entries of the weight vector w are positive. They then propose the following optimization problem to find a weight vector w and a policy $\tilde{\pi}$ that capture the expert demonstration:

$$\max_{\tilde{\pi}\in\tilde{\Pi}} \min_{w:\|w\|=1, w\geq 0} w^\top \mu(\tilde{\pi}) - w^\top \mu_E. \tag{2.15}$$

To see the relationship between our formulation, as given in Eqn. (2.14), and the formulation by Syed and Schapire in Eqn. (2.15), we take the Lagrangian dual of the minimization over w and obtain the following optimization problem:

$$\begin{aligned} \max_{\xi\geq 0, \tilde{\pi}\in\tilde{\Pi}} \quad & \xi \\ \text{s.t.} \quad & \mu(\tilde{\pi}) \geq \mu_E + \xi \mathbf{1}. \end{aligned} \tag{2.16}$$

Hence, there are two differences with our formulation: (i) Rather than finding the policy that is closest (in feature space) to the expert's demonstration, Syed and Schapire's formulation suggests to compute a policy $\tilde{\pi}$ that is better than the expert. This is possible as they assume knowledge of the fact that all the reward weights are positive. (ii) They use the infinity norm to measure distance from the expert rather than the 2-norm.

Ratliff et al. (2006) and their later extension (Ratliff et al., 2007) also consider the same apprenticeship learning setting as ours. I.e., they assume an MDP\R, a set of features $\phi(\cdot)$, and expert demonstrations are given. They suggest the following max-margin optimization problem to find a weight vector that explains the expert

demonstrations well:[14]

$$\min_{w,\zeta\geq 0,v} \quad \tfrac{1}{2}\|w\|^2 + \beta\zeta \tag{2.17}$$

$$\text{s.t.} \quad w^\top F\lambda_E + \zeta \geq d^\top v \tag{2.18}$$

$$v(s) \geq (w^\top F + l)(s,a) + \textstyle\sum_{s'} P(s'|s,a)v(s'). \tag{2.19}$$

Here λ_E denotes the state action visitation rates of the expert, i.e., each entry $\lambda_E(s,a) = \mathrm{E}[\sum_{t=0}^{\infty}\gamma^t \mathbf{1}\{$ expert in state s and takes action a at time $t\}]$. The matrix F is defined such that $\mu = F\lambda$, hence F is a matrix tabulation of the features $\phi(\cdot)$. The vector d is the initial state distribution. The loss function $l(s,a)$ represents prior information on how bad it is to visit the state (s,a). This is similar to loss functions in supervised learning structured prediction. For example, for a path planning problem $l(s,a)$ could be equal to the distance between s and the nearest state visited by the expert demonstration. (See Ratliff et al., 2006 for details.) Their constraints enforce (i) The expert performance ($w^t F\lambda_E$) is higher than the performance of the value function v in the MDP, up to some slack ζ; (ii) The value function v has to satisfy the Bellman equation, where the reward function is equal to the learned reward function $w^\top F$ plus a loss function l. Their objective (Eqn. 2.17) regularizes the weight vector w and penalizes for a non-zero slack variable ζ. To illustrate the close relationship with our formulation, we take the Lagrangian dual of the optimization problem in Eqn. (2.17, 2.18, 2.19), which gives us:

$$\min_{\kappa\geq 0,\lambda\geq 0} \quad \tfrac{1}{2}\|\kappa F\lambda_E - F\lambda\|_2^2 - \textstyle\sum_{s,a}\lambda(s,a)l(s,a) \tag{2.20}$$

$$\text{s.t.} \quad \kappa \leq \beta \tag{2.21}$$

$$\textstyle\sum_a \lambda(s,a) = \kappa d(s) + \sum_{s',a} P(s|s',a)\lambda(s',a). \tag{2.22}$$

In case we replace the second constraint by $\lambda = 1$, and have a constant loss function l, then the formulation of Ratliff et al. (2006) becomes very similar to our formulation. In general, the optimization problem is very similar to ours (as given in Eqn. 2.14), yet differs in (i) The objective is augmented with a loss function; (ii) Mixture policies are

[14]The formulation in their paper also considers the case of multiple MDPs.

not considered; (iii) The regularization allows one to scale down κ, which effectively scales down w. From the solution of the optimization problem in Eqn. (2.22), we can find the weight vector as $w = \kappa F \lambda_E - F\lambda$.

Neu and Szepesvari (2007) and Ramachandran and Amir (2007) also consider the same setting. They use a loss function which operates directly in policy space rather than reward feature space. Concretely, Neu and Szepesvari (2007) formulate an optimization problem that finds the reward function such that the policy obtained in the MDP when using that reward function takes similar actions as the expert (as measured with some loss function). Ramachandran and Amir (2007) consider the Q-function induced by a choice of reward function. Their approach maintains a probability distribution over reward functions. The likelihood of a reward function is the product of the likelihood of each of the expert's actions. The likelihood of an action a in state s is given by $\frac{\exp(Q(s,a))}{\sum_{b \in \mathcal{A}} \exp(Q(s,b))}$. We refer the reader to their papers for more details.

2.7 Discussion and Conclusions

When applying reinforcement learning algorithms, often one of the key challenges is to provide a reward function that properly articulates the task we want the system to perform. Rather than articulating the reward function, it can often be more convenient to have an expert demonstrate the desired task. In this chapter we presented an algorithm that takes advantage of expert demonstrations to alleviate the need to explicitly specify a reward function.

We assumed the expert is trying to maximize a reward function expressible as a linear combination of known features. We presented an apprenticeship learning algorithm based on inverse reinforcement learning. The presented algorithm terminates in a small number of iterations, and guarantees that the policy found will have performance comparable to or better than that of the expert, on the expert's unknown reward function.

Our algorithm assumed the reward function is expressible as a linear function of known features. If the set of features is sufficiently rich, this assumption is fairly

unrestrictive. (In the extreme case where there is a separate feature for each state-action pair, fully general reward functions can be learned.) However, it remains an important problem to develop methods for learning reward functions that may be non-linear functions of the features, and to incorporate automatic feature construction and feature selection ideas into our algorithms. See in Ratliff et al. (2007) for some recent work in this direction.

It might be possible to derive an alternative apprenticeship learning algorithm using the dual to the LP that is used to solve Bellman's equations. (Manne, 1960) Specifically, in this LP the variables are the state/action visitation rates, and it is possible to place constraints on the learned policy's stationary distribution directly. While there are few algorithms for approximating this dual (as opposed to primal) LP for large MDPs and exact solutions would be feasible only for small MDPs, we consider this an interesting direction for future work.

Chapter 3

Exploration and Apprenticeship Learning in Reinforcement Learning

Two of the key challenges when applying traditional control and reinforcement learning methods are that they require (i) an appropriate reward function, and (ii) a dynamics model that is sufficiently accurate. In the previous chapter we described how we can leverage expert demonstrations to learn the reward function, which alleviates the need for the (often very time-consuming) process of hand-engineering an appropriate reward function. In this chapter we describe how we can leverage expert demonstrations to obtain a good dynamics model.

3.1 Introduction

The reinforcement learning and optimal control literatures provide a powerful set of tools for modeling and solving control problems, and many algorithms exist for finding (near) optimal solutions for a given Markov decision process (MDP). (See, e.g., Bertsekas & Tsitsiklis, 1996; Sutton & Barto, 1998; Anderson & Moore, 1989; Zhou et al., 1995.) To apply these algorithms to control problems in which the dynamics are not known in advance, the parameters of the MDP typically need to be

learned from observations of the system.

A key problem in learning an MDP's parameters is that of *exploration*: How can we ensure that all relevant parts of the MDP are visited sufficiently often that we manage to collect accurate statistics for their state transition probabilities? A state-of-the-art answer to this problem in the reinforcement learning literature is the E^3-algorithm (Kearns and Singh, 2002) (and variants/extensions: Kearns and Koller, 1999; Kakade, Kearns and Langford, 2003; Brafman and Tennenholtz, 2002). These algorithms guarantee that near-optimal performance will be obtained in time polynomial in the number of states of the system. The basic idea of E^3 is that it will repeatedly apply an "exploration policy," i.e., one that tries to visit state-action pairs whose transition dynamics are still inaccurately modeled. After a polynomial number of iterations, it will deem itself to have modeled enough of the MDP accurately. Then, it will apply an "exploitation policy," which (given the current MDP model) tries to maximize the sum of rewards obtained over time. In the original E^3 work (Kearns and Singh, 2002), the algorithm would explicitly use an exploration policy until the model was considered accurate enough, after which it switched to an exploitation policy. In later variants such as the R-Max algorithm by Brafman and Tennenholtz (2002) this choice of exploration vs. exploitation policy was made less explicitly, but through a reward scheme reminiscent of "optimism in the face of uncertainty," (e.g., Kaelbling, Littman and Moore, 1996). However, the algorithm still tends to end up generating (and using) exploration policies in its initial stage.

To achieve its performance guarantees, the E^3-family of algorithms demand that we run exploration policies on the unknown system until we have an accurate model for the entire MDP (or at least for the "reachable" parts of it). The strong bias towards exploration makes the policies generated by the E^3-family often unacceptable for running on a real system. Consider for example running E^3 on an autonomous helicopter. This would require executing policies that aggressively explore different parts of the state-space, including parts of it that would lead to crashing the helicopter.[1] As a second example, if the system to be controlled is a chemical plant,

[1]Indeed, in our work on autonomous helicopter flight, our first crash occurred during (manual flight) exploration, when a human pilot was over-aggressive in exploring the boundaries of the flight envelope (moving the control sticks through their extreme ranges), which placed excessive strain on

E^3-generated policies may well cause an explosion in the plant through its aggressive exploration of the entire state space. Despite the strong theoretical results, for many robotics and other applications, we do not believe that E^3 is a practical algorithm.

In this chapter, we consider the apprenticeship learning setting, in which we have available an initial teacher demonstration of the task to be learned. For example, we may have a human pilot give us an initial demonstration of helicopter flight. Given this initial training data with which to learn the dynamics, we show that it suffices to only execute exploitation policies (ones that try to do as well as possible, given the current model of the MDP). More specifically, we propose the following algorithm:

1. Have a teacher demonstrate the task to be learned, and record the state-action trajectories of the teacher's demonstration.
2. Use all state-action trajectories seen so far to learn a dynamics model for the system. For this model, find a (near) optimal policy using any reinforcement learning (RL) algorithm.
3. Test that policy by running it on the real system. If the performance is as good as the teacher's performance, stop. Otherwise, *add the state-action trajectories from the (unsuccessful) test to the training set*, and go back to step 2.

Note that the algorithm we described uses a greedy policy with respect to the current estimated model at every iteration. So there is never an explicit exploration step.

The algorithm we propose uses a greedy policy with respect to the current estimated model at all times. So there is never an explicit exploration step. For systems with complex dynamics, merely requiring expert demonstrations (rather than explicit exploration through exhaustive data collection to capture all aspects of the systems dynamics) can save a substantial amount of time. Beyond that, in practice, exploitation policies often tend to be more benign, and thus can be significantly more palatable to deploy. For example, unlike E^3, this is a procedure that can much more safely and confidently be tried on an autonomous helicopter.

We note that the algorithm proposed above also parallels a reasonably common practice in applied control, in which some initial policy is used to collect data and

the rotor head assembly and caused it to disintegrate in mid-air.

build a model for a simulator. Then, if subsequently a controller is found that works in simulation but not in real-life, the designer tries (usually manually) to adjust the simulator to make it correctly predict the failure of this policy. If machine learning is used to build the simulator, then a natural way to modify the simulator after observing an unsuccessful policy is to add the data obtained from the unsuccessful policy to the training set. Thus, our work can also be viewed as formally analyzing, and thereby attempting to cast light on, the conditions under which a procedure like this can be expected to lead to a good policy.

In fact, a large fraction of the traditional control literature considers the problem of modeling system dynamics from data—typically referred to as system identification. (See, e.g., Ljung, 1986.) Amongst others, the system identification literature studies which "probing" signals can give us accuracy guarantees about the resulting models. Interestingly, our algorithm can be seen as choosing a very specific set of probing signals, namely expert demonstrations of the task at hand. Our results show that, if our autonomous control policy acts on the same system and in the same conditions as the expert (technically, in the same Markov decision process), then it is sufficient to merely collect expert demonstrations of the task at hand rather than to exhaustively excite all dynamical modes of the system. The system identification literature typically suggests to ensure all modes of the system are being excited sufficiently during data collection. In exchange for this more exhaustive data collection procedure—which could take substantially longer for complex systems—the traditional system identification algorithms can attain stronger guarantees (typically under certain model structure assumptions). For example, assuming we follow their prescribed data collection procedures, the system identification algorithms ensure us that we obtain dynamics models of sufficient detail to enable our control algorithms to be accompanied by performance guarantees, such as robustness to certain perturbations, and global asymptotic stability in the presence of bounded external perturbations. More exhaustive data collection (probing) efforts can be worthwhile when the expert is demonstrating in a different environment or operating conditions than the conditions in which the autonomous system will be tested. This could happen, e.g., when the testing environment is not available at demonstration time.

Previous work has shown the effectiveness of using teacher or expert demonstrations (called apprenticeship learning, also imitation learning, and learning by watching) in various ways for control. Schaal and Atkeson (1994) and Smart and Kaelbling (2000) both give examples where learning is significantly faster when bootstrapping from a teacher. Their methods are somewhat related in spirit, but different in detail from ours (e.g., Smart and Kaelbling, 2000, uses model-free Q-learning, and does not learn the MDP parameters), and had no formal guarantees.

Other examples include Sammut et al. (1992); Kuniyoshi, Inaba and Inoue (1994); Demiris and Hayes (1994); Amit and Mataric (2002); and Pomerleau (1989), which apply supervised learning to learn a parameterized policy from the demonstrations. In these examples, neither the reward function nor the system dynamics need to be specified since a policy is learned directly as a mapping from the states to the actions. This approach has been applied successfully in a variety of applications, but may require careful selection of an appropriate policy class parameterization, and generally lacks strong performance guarantees. We showed in Chapter 2 how to use the demonstrations to remove the need for explicitly specifying a reward function; there, the system dynamics were assumed to be known.

In what follows, we prove that, with high probability, our algorithm given above terminates with a policy whose performance is comparable to (or better than) the teacher. In the case of discrete state MDPs, the algorithm scales at most polynomially in the number of states. In the case of linearly parameterized dynamical systems, we use a martingale over relative losses to show that the algorithm scales at most polynomially in the dimension of the state space.

The work described in this chapter has first appeared in Abbeel and Ng (2005a).

3.2 Preliminaries

A Markov decision process (MDP) is a tuple $(S, \mathcal{A}, T, H, D, R)$, where S is a set of states; $\mathcal{A}$ is a set of actions/inputs; $T = \{P(\cdot|s,a)\}_{s,a}$ is a set of state transition probabilities (here, $P(\cdot|s,a)$ is the state transition distribution upon taking action a in

state s); H is the horizon time of the MDP, so that the MDP terminates after H steps;[2] D is a distribution over states from which the initial state s_0 is drawn; and $R : S \mapsto \mathbb{R}$ is the reward function, which we assume to be non-negative and bounded by $R_{\max}$. A policy π is a mapping from states S to a probability distribution over the set of actions $\mathcal{A}$. The utility of a policy π in an MDP M is given by $U_M(\pi) = \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi, M]$. Here the expectation is over all possible state trajectories in the MDP M.

Specifying an MDP therefore requires specifying each item of the tuple (S, $\mathcal{A}$, T, H, D, R). In practice, the state transitions probabilities T are usually the most difficult element of this tuple to specify, and must often be learned from data. More precisely, the state space S and action space $\mathcal{A}$ are physical properties of the system being controlled, and thus easily specified. R (and H) is typically given by the task specification (or otherwise can be learned from a teacher demonstration, as explained in Chapter 2). Finally, D is usually either known or can straightforwardly be estimated from data. Thus, in the sequel, we will assume that $S, \mathcal{A}, H, D$ and R are given, and focus exclusively on the problem of learning the state transition dynamics T of the MDP.

Consider an MDP $M = (S, \mathcal{A}, T, H, D, R)$, and suppose we have some approximation $\hat{T}$ of the transition probabilities. Thus, $\hat{M} = (S, \mathcal{A}, \hat{T}, H, D, R)$ is our approximation to M. The Simulation Lemma (stated below) shows that so long as $\hat{T}$ is close to T on states that are visited with high probability by a policy π, then the utility of π in $\hat{M}$ is close to the utility of π in M. The Simulation Lemma shows certain robustness guarantees for policies we might find using model-based reinforcement learning, in that, it can be sufficient to evaluate policies in an approximation of the true MDP. (Related results are also given in Kearns and Singh, 2002; Kearns and Koller, 1999; Kakade, Kearns and Langford, 2003; Brafman and Tennenholtz, 2002.)

Lemma 2 (Simulation Lemma)**.** *Let any $\epsilon, \eta \geq 0$ be given. Let an MDP $M = (S, \mathcal{A}, T, H, D, R)$ be given. Let $\hat{M} = (S, \mathcal{A}, \hat{T}, H, D, R)$ be another MDP which only differs from M in its transition probabilities. Let π be a policy over the state-action sets $S, \mathcal{A}$, so that π can be applied to both M and $\hat{M}$. Assume there exists a set of*

[2] Any infinite horizon MDP with discounted rewards can be ϵ-approximated by a finite horizon MDP, using a horizon $H_\epsilon = \lceil \log_\gamma(\epsilon(1-\gamma)/R_{\max}) \rceil$.

state-action pairs $\overline{SA}_\eta \subseteq S \times \mathcal{A}$ *such that the following holds*

$$\begin{aligned} &\text{(i)} \quad \forall (s,a) \in \overline{SA}_\eta, \quad d_{\text{var}}(P(\cdot|s,a), \hat{P}(\cdot|s,a)) \leq \epsilon, \\ &\text{(ii)} \quad P(\{(s_t,a_t)\}_{t=0}^H \subseteq \overline{SA}_\eta | \pi, M) \geq 1-\eta. \end{aligned}$$

(Above, d_{var} *denotes variational distance.*[3]*) Then we have*

$$|U_M(\pi) - U_{\hat{M}}(\pi)| \leq H^2 \epsilon R_{\max} + \eta H R_{\max}.$$

Consider the special case where every state-action pair $(s,a) \in S \times \mathcal{A}$ satisfies condition (i), in other words, $\overline{SA}_\eta = S \times \mathcal{A}$ and thus condition (ii) is satisfied for $\eta = 0$. Then the Simulation Lemma tells us that accurate transition probabilities are sufficient for accurate policy evaluation. The Simulation Lemma also shows that not necessarily all state-action pairs' transition probabilities need to be accurately modeled: it is sufficient to accurately model a subset of state-action pairs $\overline{SA}_\eta$ such that the probability of leaving this set $\overline{SA}_\eta$ under the policy π is sufficiently small.

Let there be some event that has probability bounded away from zero. Suppose we would like to observe that event some minimum number of times in a set of IID experiments. The following lemma allows us to prove bounds on how often we need to repeat the experiment to see that event at least the desired number of times (with high probability).

Lemma 3. *Let any* $\delta > 0$ *and* $a > 0$ *be given. Let* $\{X_i\}_{i=1}^m$ *be IID Bernoulli(*ϕ*) random variables. Then for* $\sum_{i=1}^m X_i \geq a$ *to hold with probability at least* $1-\delta$*, it suffices that* $m \geq \frac{2}{\phi}(a + \log \frac{1}{\delta})$.

3.3 Problem description

The problems we are concerned with in this chapter are control tasks that can be described by an MDP $M = (S, \mathcal{A}, T, H, D, R)$. However the system dynamics T are

[3]Let $P(\cdot), Q(\cdot)$ be two probability distributions over a set $\mathcal{X}$, then the variational distance $d_{\text{var}}(P,Q)$ is defined as follows: $d_{\text{var}}(P,Q) = \frac{1}{2}\int_{x \in \mathcal{X}} |P(x) - Q(x)| dx$.

unknown. Everything else in the specification of the MDP is assumed to be known. We consider two specific classes of state-action spaces and transition probabilities, which we will refer to as discrete dynamics and linearly parameterized dynamics respectively.

- Discrete dynamics: The sets S and $\mathcal{A}$ are finite sets. The system dynamics T can be described by a set of transition probabilities $P(s'|s,a)$, which denote the probability of the next-state being s' given the current state is s and the current action is a. More specifically we have a multinomial distribution $P(\cdot|s,a)$ over the set of states S for all state-action pairs $(s,a) \in S \times \mathcal{A}$.
- Linearly parameterized dynamics: The sets $S = \mathbb{R}^{n_S}$ and $\mathcal{A} = \mathbb{R}^{n_\mathcal{A}}$ are now continuous. We assume the system obeys the following dynamics:[4]

$$x_{t+1} = A\phi(x_t) + Bu_t + w_t, \tag{3.1}$$

  where $\phi(\cdot) : \mathbb{R}^{n_S} \to \mathbb{R}^{n_S}$. Thus, the next-state is a linear function of some (possibly non-linear) features of the current state (plus noise). This generalizes the familiar LQR model from classical control (Anderson and Moore, 1989) to non-linear settings. For example, the (body-coordinates) helicopter model we describe in Chapter 5 is of this form, with a particular choice of non-linear ϕ, and the unknown parameters A and B are estimated from data. The process noise $\{w_t\}_t$ is IID with $w_t \sim \mathcal{N}(0, \sigma^2 I_{n_S})$. Here σ^2 is a fixed, known, constant. We also assume that $\|\phi(s)\|_2 \leq 1$ for all s, and that the inputs u_t satisfy $\|u_t\|_2 \leq 1$.[5]

3.4 Algorithm

Let π_T be the policy of a teacher. Although it is natural to think of π_T as a good policy for the MDP, we do not assume this to be the case. Let any $\alpha > 0$ be given.

[4] We chose to adhere to the most commonly used notation for continuous systems. I.e., states are represented by x, inputs by u and the system matrices by A and B. We use script $\mathcal{A}$ for the set of actions and standard font A for the system matrix.

[5] The generalizations to unknown σ^2, to non-diagonal noise covariances, and to non-linear features over the inputs ($B\psi(u_t)$ replacing Bu_t) offer no special difficulties.

Our algorithm (with parameters N_T and k_1) is as follows:

1. Run N_T trials under the teacher's policy π_T. Save the state-action trajectories encountered during these trials. Compute $\hat{U}_M(\pi_T)$—an estimate of the utility of the teacher's policy π_T for the real system M—by averaging the sum of rewards accumulated in each of the N_T trials. Initialize $i = 1$.
2. Using all state-action trajectories saved so far, estimate the system dynamics T using maximum likelihood estimation for the discrete dynamics case, and regularized linear regression for the linearly parameterized dynamics case (as described below). Call the estimated dynamics $\hat{T}^{(i)}$.
3. Find a $\alpha/8$ optimal policy[6] for the MDP $\hat{M}^{(i)} = (S, \mathcal{A}, \hat{T}^{(i)}, H, D, R)$. Call this policy $\pi^{(i)}$.
4. Evaluate the utility of the policy $\pi^{(i)}$ on the real system M. More specifically, run the policy $\pi^{(i)}$ for k_1 trials on the system M. Let $\hat{U}_M(\pi^{(i)})$ be the average sum of rewards accumulated in the k_1 trials. Save the state-action trajectories encountered during these trials.
5. If $\hat{U}_M(\pi^{(i)}) \geq \hat{U}_M(\pi_T) - \alpha/2$, return $\pi^{(i)}$ and exit. Otherwise set $i = i + 1$ and go back to step 2.

In the i^{th} iteration of the algorithm, a policy is found using an estimate $\hat{T}^{(i)}$ of the true system dynamics T. For the discrete dynamics, the estimate used in the algorithm is the maximum likelihood estimates for each of the multinomial distributions $P(\cdot|s, a)$. For the linearly parameterized dynamics, the model parameters A, B are estimated via regularized linear regression. In particular the k^{th} rows of A and B are estimated by[7] $\arg\min_{A_{k,:},B_{k,:}} \sum_j (x^{(j)}_{\text{next}} - (A_{k,:}\phi(x^{(j)}_{\text{curr}}) + B_{k,:}u^{(j)}_{\text{curr}}))^2 + \frac{1}{\kappa^2}(\|A_{k,:}\|_2^2 + \|B_{k,:}\|_2^2)$, where j indexes over all state-action-state triples $\{(x^{(j)}_{\text{curr}}, u^{(j)}_{\text{curr}}, x^{(j)}_{\text{next}})\}_j$ occurring after each other in the trajectories observed for the system.

[6] A policy π_1 is an ϵ-optimal policy for an MDP M if $U_M(\pi_1) \geq \max_\pi U_M(\pi) - \epsilon$.

[7] We use matlab-like notation. $A_{k,:}$ denotes the k^{th} row of A.

3.5 Main theorem

The following theorem gives performance and running time guarantees for the algorithm described in Section 3.4.[8]

Theorem 4. *Let an MDP $M = (S, \mathcal{A}, T, H, D, R)$ be given, except for its transition probabilities T. Let the system either be a discrete dynamics system or a linearly parameterized dynamical system as defined in Section 3.3. Let any $\alpha > 0, \delta > 0$ be given. Let π_T be the teacher's policy, and let π be the policy returned by the algorithm defined above. Let N denote the number of iterations of the main loop of the algorithm until the exit condition is met. Let $\mathcal{T} = (H, R_{\max}, |S|, |\mathcal{A}|)$ for the discrete case, and let $\mathcal{T} = (H, R_{\max}, n_S, n_{\mathcal{A}}, \|A\|_{\mathrm{F}}, \|B\|_{\mathrm{F}})$ for the linearly parameterized dynamics case. Then for*

$$U_M(\pi) \geq U_M(\pi_T) - \alpha, \tag{3.2}$$

$$N = O(\mathrm{poly}(\frac{1}{\alpha}, \frac{1}{\delta}, \mathcal{T})) \tag{3.3}$$

to hold with probability at least $1 - \delta$, it suffices that

$$N_T = \Omega(\mathrm{poly}(\frac{1}{\alpha}, \frac{1}{\delta}, \mathcal{T})), \tag{3.4}$$

$$k_1 = \Omega(\mathrm{poly}(\frac{1}{\alpha}, \frac{1}{\delta}, \mathcal{T})). \tag{3.5}$$

Note that Eqn. (3.2) follows from the termination condition of our algorithm and assuming we choose k_1 and N_T large enough such that the utilities of the policies $\{\pi^{(i)}\}_i$ and π_T are sufficiently accurately evaluated in M.

The proof of this theorem is quite lengthy, and will make up most of the remainder of this chapter. We now give a high-level sketch of the proof ideas. Our proof is based on showing the following two facts:

[8] The performance guarantees in the theorem are stated with respect to the teacher's demonstrated performance. However, the proof requires only that the initial dynamical model be accurate for at least one good policy. Thus, for example, it is sufficient to observe a few good teacher demonstrations along with many bad demonstrations (ones generated via a highly sub-optimal policy); or even only bad demonstrations that manage to visit good parts of the state space.

1. After we have collected sufficient data from the teacher, the estimated model is accurate for evaluating the utility of the teacher's policy in every iteration of the algorithm. (Note this does not merely require that the model has to be accurate after the N_T trials under the teacher's policy, but also has to stay accurate after extra data is collected from testing the policies $\{\pi^{(i)}\}_i$.)
2. One can visit inaccurately modeled state-action pairs only a "small" number of times until all state-action pairs are accurately modeled.

We now sketch how these two facts can be proved. After we have collected sufficient data from the teacher, the state-action pairs that are visited often under the teacher's policy are modeled well. From the Simulation Lemma we know that an accurate model of the state-action pairs visited often under the teacher's policy is sufficient for accurate evaluation of the utility of the teacher's policy. This establishes (1.). Every time an inaccurate state-action pair is visited, the data collected for that state-action pair can be used to improve the model. However the model can be improved only a "small" number of times until it is accurate for all state-action pairs. This establishes (2.).

We now explain how these two facts can be used to bound the number of iterations of our algorithm. Consider the policy $\pi^{(i)}$ found in iteration i of the algorithm. This policy $\pi^{(i)}$ is the optimal policy[9] for the current model. When finding this policy $\pi^{(i)}$ in the model we could have chosen the teacher's policy. So the policy $\pi^{(i)}$ performs at least as well as the teacher's policy in the current model. Now if in the real system the utility of the policy $\pi^{(i)}$ is significantly lower than the teacher's utility (which is the case as long as the algorithm continues), then the model incorrectly predicted that $\pi^{(i)}$ was better than the teacher's policy. From (1.) we have that the model correctly evaluates the utility of the teacher's policy. Thus the model must have evaluated the policy $\pi^{(i)}$ inaccurately. Using the (contrapositive of) the Simulation Lemma, we get that the policy $\pi^{(i)}$ must be visiting (with probability bounded away from 0) state-action pairs that are not very accurately modeled. So when running the policy $\pi^{(i)}$ we can collect training data that allow us to improve the model. Now from (2.) we

[9]For simplicity of exposition in this informal discussion, we assume $\pi^{(i)}$ is optimal, rather than near-optimal. The formal results in this chapter do not use this assumption.

have that visiting inaccurately modeled state-action pairs can only happen a small number of times until the dynamics is learned, thus giving us a bound on the number of iterations of the algorithm.

The theorem will be proved for the discrete dynamics case in Section 3.6 and for the linearly parameterized dynamics case in Section 3.7.

3.6 Discrete state space systems

In this section we prove Theorem 4 for the case of discrete dynamics.

The Hoeffding inequality gives a bound on the number of samples that are sufficient to estimate the expectation of a (bounded) random variable. In our algorithm, we want to guarantee that the model is accurate (for the teacher's policy) not only when we have seen the samples from the teacher, but also any time after additional samples are collected. The following lemma, which is a direct consequence of Hoeffding's inequality (as shown in Appendix B), gives such a bound.

Lemma 5. *Let any $\epsilon > 0, \delta > 0$ be given. Let X_i be IID k-valued multinomial random variables, with distribution denoted by P. Let $\hat{P}_n$ denote the n sample estimate of P. Then for $\max_{n \geq N} d_{\mathrm{var}}(P(\cdot), \hat{P}_n(\cdot)) \leq \epsilon$ to hold with probability $1 - \delta$, it suffices that $N \geq \frac{k^2}{4\epsilon^2} \log \frac{k^2}{\delta\epsilon}$.*

Lemma 5 will serve two important purposes. In the proof of Lemma 6 it is used to bound the number of trajectories needed under the teacher's policy to guarantee that frequently visited state-action pairs are accurately modeled in all models $\{\hat{M}^{(i)}\}_i$. This corresponds to establishing Fact (1.) of the proof outline in Section 3.5. In the proof of Lemma 7 it is used to bound the total number of times a state-action pair can be visited that is not accurately modeled. This latter fact corresponds exactly to establishing Fact (2.) of the proof outline in Section 3.5.[10]

[10]Fact (2.) follows completely straightforwardly from Lemma 5, so rather than stating it as a separate lemma, we will instead derive it within the proof of Lemma 7.

Lemma 6. *Let any $\alpha > 0, \delta > 0$ be given. Assume we use the algorithm as described in Section 3.4. Let N_T satisfy the following condition*

$$N_T \geq \frac{4096|S|^3|\mathcal{A}|H^5R_{\max}^3}{\alpha^3}\log\frac{32H^2R_{\max}|S|^3|\mathcal{A}|}{\delta\alpha}.$$

Then with probability $1-\delta$ we have that

$$\forall i \geq N_T \quad |U_{\hat{M}^{(i)}}(\pi_T) - U_M(\pi_T)| \leq \alpha/8.$$

Proof (sketch). Let $\epsilon > 0, \eta > 0$. Let $SA_\xi \subseteq S \times A$ be the set of state-action pairs such that the probability of seeing any specific state-action pair $(s,a) \in SA_\xi$ under the policy π_T in a single trial of duration H is at least $\frac{\eta}{|S||\mathcal{A}|}$. From Lemma 5 and Lemma 3 we have that for any $(s,a) \in SA_\xi$ for

$$\forall i \geq N_T \quad d_{\text{var}}(P(\cdot|s,a), \hat{P}^{(i)}(\cdot|s,a)) \leq \epsilon \tag{3.6}$$

to hold with probability $1-\delta'-\delta''$, it is sufficient to have

$$N_T \geq \frac{2|S||\mathcal{A}|}{\eta}\left(\frac{|S|^2}{4\epsilon^2}\log\frac{|S|^2}{\delta'\epsilon} + \log\frac{1}{\delta''}\right). \tag{3.7}$$

Taking a union bound over all state-action pairs $(s,a) \in SA_\xi$ (note $|SA_\xi| \leq |S||\mathcal{A}|$) gives that for Eqn. (3.6) to hold for all $(s,a) \in SA_\xi$ with probability $1 - |S||\mathcal{A}|\delta' - |S||\mathcal{A}|\delta''$, it suffices that Eqn. (3.7) is satisfied. We also have from the definition of SA_ξ that $\mathrm{P}(\{(s_t,a_t)\}_{t=0}^H \subseteq \overline{SA}_\xi|\pi_T) \geq 1-\eta$. Thus the Simulation Lemma gives us that

$$\forall i \quad |U_{\hat{M}^{(i)}}(\pi_T) - U_M(\pi_T)| \leq H^2\epsilon R_{\max} + \eta H R_{\max}.$$

Now choose $\epsilon = \frac{1}{2}\frac{\alpha/8}{H^2R_{\max}}$, $\eta = \frac{1}{2}\frac{\alpha/8}{HR_{\max}}$ and $\delta' = \delta'' = \frac{\delta}{2|S||\mathcal{A}|}$ to get the lemma. □

Lemma 6 shows that, after having seen the teacher sufficiently often, the learned model will be accurate for evaluating the utility of the teacher's policy. Moreover, no later data collection (no matter under which policy the data is collected) can make the model inaccurate for evaluation of the utility of the teacher's policy. I.e., $U_{\hat{M}^{(i)}}(\pi_T)$

will be close to $U_M(\pi_T)$ for all i.

Lemma 7. *Let any $\alpha > 0, \delta > 0$ be given. Let*

$$N_{\text{ubound}} = \frac{32HR_{\max}}{\alpha}(\log\frac{4}{\delta} + \frac{16^2H^4R_{\max}^2|S|^3|\mathcal{A}|}{4\alpha^2}\log\frac{64H^2R_{\max}|S|^3|\mathcal{A}|}{\alpha\delta}). \tag{3.8}$$

Assume in the algorithm described in Section 3.4 we use

$$k_1 \geq \frac{16^2H^2R_{\max}^2}{2\alpha^2}\log\frac{8N_{\text{ubound}}}{\delta}, \tag{3.9}$$

$$N_T \geq \frac{4096|S|^3|\mathcal{A}|H^5R_{\max}^3}{\alpha^3}\log\frac{256H^2R_{\max}|S|^3|\mathcal{A}|}{\delta\alpha}. \tag{3.10}$$

Let N denote the number of iterations of the algorithm until it terminates. Then we have that with probability $1-\delta$ the following hold

$$\text{(i)} \quad N \leq N_{ubound}, \tag{3.11}$$

$$\text{(ii)} \quad \forall i = 1:N \quad |U_{\hat{M}^{(i)}}(\pi_T) - U_M(\pi_T)| \leq \frac{\alpha}{8}, \tag{3.12}$$

$$\text{(iii)} \quad \forall i = 1:N \quad |\hat{U}_M(\pi^{(i)}) - U_M(\pi^{(i)})| \leq \frac{\alpha}{16}, \tag{3.13}$$

$$\text{(iv)} \quad |\hat{U}_M(\pi_T) - U_M(\pi_T)| \leq \frac{\alpha}{16}. \tag{3.14}$$

Proof (sketch). From Lemma 6 and from the Hoeffding inequality we have that for Eqn. (3.12), (3.13) and (3.14) to hold (for all $i \leq N_{\text{ubound}}$) with probability $1-\frac{\delta}{2}$, it suffices that Eqn. (3.10) and Eqn. (3.9) are satisfied.

Now since the algorithm only exits in iteration N, we must have for all $i = 1:N-1$ that

$$\hat{U}_M(\pi^{(i)}) < \hat{U}_M(\pi_T) - \alpha/2. \tag{3.15}$$

Combining Eqn. (3.15), (3.12), (3.13) and (3.14) and the fact that $\pi^{(i)}$ is $\alpha/8$-optimal for $\hat{M}^{(i)}$ we get

$$\forall i = 1:N-1 \quad U_{\hat{M}^{(i)}}(\pi^{(i)}) \geq U_M(\pi^{(i)}) + \alpha/8. \tag{3.16}$$

In words: when the algorithm continues (in iterations $i = 1:N-1$), the model

overestimated the utility of $\pi^{(i)}$. Using the contrapositive of the Simulation Lemma with $\epsilon = \frac{1}{2}\frac{\alpha/8}{H^2 R_{\max}}$ we get that for all $i = 1 : N-1$ the policy $\pi^{(i)}$ must be visiting a state-action pair (s, a) that satisfies

$$d_{\text{var}}(P(\cdot|s,a), \hat{P}^{(i)}(\cdot|s,a)) > \frac{\alpha}{16H^2 R_{\max}} \tag{3.17}$$

with probability at least $\frac{\alpha}{16HR_{\max}}$. From Lemma 3 and Lemma 5 we get that if the algorithm had run for a number of iterations N_{ubound} then with probability $1 - \frac{\delta}{2}$ all state-actions pairs would satisfy

$$d_{\text{var}}(P(\cdot|s,a), \tilde{P}^{(N)}(\cdot|s,a)) \leq \frac{\alpha}{16H^2 R_{\max}}. \tag{3.18}$$

On the other hand we showed above that if the algorithm does not exit in iteration i, there must be a state-action pair satisfying Eqn. (3.17), which contradicts Eqn. (3.18). Thus N_{ubound} gives an upper bound on the number of iterations of the algorithm. □

The proof of Theorem 4 for the case of discrete dynamics is a straightforward consequence of Lemma 7.

Proof of Theorem 4 for discrete dynamics. First note that the conditions on N_T and k_1 of Lemma 7 are satisfied in Theorem 4. So Lemma 7 proves the bound on the number of iterations as stated in Eqn. (3.3). Now it only remains to prove that at termination, Eqn. (3.2) holds. We have from the termination condition that $\hat{U}(\pi) \geq \hat{U}(\pi_T) - \alpha/2$. Now using Eqn. (3.13) and Eqn. (3.14) we get $U_\pi \geq U_{\pi_T} - \frac{5}{8}\alpha$, which implies Eqn. (3.2). □

3.7 Linearly parameterized dynamical systems

In this section we prove Theorem 4 for the case of linearly parameterized dynamics described in Eqn. (3.1). As pointed out in Section 3.5, the performance guarantee of Eqn. (3.2) follows from the termination condition of our algorithm and assuming we choose k_1 and N_T large enough such that the utility of the policies $\{\pi^{(i)}\}_i$ and π_T are sufficiently accurately evaluated in M. This leaves us to prove the bound on the

number of iterations of the algorithm. As explained more extensively in Section 3.5, there are two main parts to this proof. In Section 3.7.2 we establish the first part: the estimated model is accurate for evaluating the utility of the teacher's policy in every iteration of the algorithm. In Section 3.7.3 we establish the second part: one can visit inaccurately modeled states only a "small" number of times (since every such visit improves the model). In Section 3.7.4 we combine these two results to prove Theorem 4 for the case of linearly parameterized dynamical systems.

3.7.1 Preliminaries

The following proposition will allow us to relate accuracy of the expected value of the next-state to variational distance for the next-state distribution. This will be important for using the Simulation Lemma, which is stated in terms of variational distance.

Proposition 8. *We have*

$$d_{\text{var}}(\mathcal{N}(\mu_1, \sigma^2 I_n), \mathcal{N}(\mu_2, \sigma^2 I_n)) \leq \frac{1}{\sqrt{2\pi}\sigma} \|\mu_1 - \mu_2\|_2.$$

3.7.2 Accuracy of the model for the teacher's policy

Given a set of state-action trajectories, the system matrices A, B are estimated by solving n_S separate regularized linear regression problems, one corresponding to each row of A and B. After appropriately relabeling variables and data, each of these regularized linear regression problems is of the form

$$\min_{\theta} \sum_{i} (y^{(i)} - \theta^\top z^{(i)})^2 + \frac{\|\theta\|_2^2}{\kappa^2}. \tag{3.19}$$

Here $\theta \in \mathbb{R}^{n_S + n_A}$ corresponds to a row in A and B, and the norm bounds on u and $\phi(x)$ result in $\|z\|_2 \leq \sqrt{2}$. The relabeled data points are kept in the same order as they were collected. The training data collected from the teacher's demonstration is indexed from 1 to $m = N_T H$. The additional training data collected when testing the policies $\{\pi^{(j)}\}_{j=1}^{N}$ is indexed from $m+1$ to $\tilde{m} = N_T H + k_1 N H$. The data is

generated according to a true model M as described in Section 3.4. In the notation of Eqn. (3.19), this means there is some θ^* such that

$$\forall i \quad y^{(i)} = \theta^{*\top} z^{(i)} + w^{(i)}, \tag{3.20}$$

where the $\{w^{(i)}\}_i$ are IID, with $w^{(i)} \sim \mathcal{N}(0, \sigma^2)$. The data generation process that we just described will be referred to as "data generated according to Eqn. (3.20)" from here on. Note that the training data $\{z^{(i)}\}_i$ in this setup are *non-IID*. The teacher's policy π_T induces a distribution over states x_t and inputs u_t at all times t. However these distributions need not be the same for different times t, making the data non-IID. Moreover, the training data indexed from $m+1$ to $\tilde{m}$ is obtained from various policies and the resulting data generation process is very difficult to model. As a consequence, our analysis will consider the worst-case scenario where an adversary can choose the additional training data indexed from $m+1$ to $\tilde{m}$.

For $1 \leq k \leq N_T H + k_1 NH$ let the following equations define $\hat{\theta}^{(k)}$ and $\text{loss}^{(k)}(\theta)$:

$$\begin{aligned} \text{loss}^{(k)}(\theta) &= \sum_{i=1}^{k} (y^{(i)} - \theta^\top z^{(i)})^2 + \frac{1}{\kappa^2} \|\theta\|_2^2, \\ \hat{\theta}^{(k)} &= \arg\min_{\theta} \quad \text{loss}^{(k)}(\theta). \end{aligned} \tag{3.21}$$

The following lemma establishes that a "small" number of samples from the teacher's policy π_T is sufficient to guarantee an accurate model $\hat{\theta}^{(k)}$ for all time steps $k = N_T H$ to $N_T H + k_1 NH$.

Lemma 9. *Let any $\delta > 0, \epsilon > 0, \eta > 0$ be given. Consider data $\{y^{(i)}, z^{(i)}\}_{i=1}^{N_T H + k_1 NH}$ generated as described in Eqn. (3.20). Let $\{\hat{\theta}^{(k)}\}_k$ be defined as in Eqn. (3.21). Let $\{\tilde{y}^{(t)}, \tilde{z}^{(t)}\}_{t=1}^{H}$ be data generated from one trial under π_T (and appropriately relabeled as described in paragraph above). Then for*

$$P(\max_{t \in 1:H} |\theta^\top \tilde{z}^{(t)} - \theta^{*\top} \tilde{z}^{(t)}| > \epsilon) \leq \eta \tag{3.22}$$

to hold with probability $1-\delta$ *for all* $\theta \in \{\hat{\theta}^{(k)}\}_{k=N_T H}^{N_T H+k_1 NH}$*, it suffices that*

$$N_T = \Omega\left(\text{poly}(\frac{1}{\epsilon}, \frac{1}{\eta}, \frac{1}{\delta}, H, \|\theta^*\|_2, n_S, n_{\mathcal{A}}, k_1, N)\right).$$

If θ satisfies Eqn. (3.22) then it is accurate for data generated under the teacher's policy and we refer to it as accurate in the discussion below; otherwise it is referred to as inaccurate. We now sketch the key ideas in the proof of Lemma 9. A full proof is provided in the appendix. The proof proceeds in four steps.

Step 1. For any inaccurate parameter θ we establish that with high probability the following holds

$$\text{loss}^{(N_T H)}(\theta) > \text{loss}^{(N_T H)}(\theta^*) + \Omega(N_T). \tag{3.23}$$

I.e., the true parameter θ^* outperforms an inaccurate parameter θ by a margin of $\Omega(N_T)$ after seeing N_T trajectories from the teacher. The key idea is that the expected value of the loss difference $\text{loss}^{(N_T H)}(\theta) - \text{loss}^{(N_T H)}(\theta^*)$ is of order N_T for inaccurate θ. Our proof establishes the concentration result for this non-IID setting by looking at a martingale over the differences in loss at every step and uses Azuma's inequality to prove the sum of these differences is close to its expected value with high probability.

Step 2. Let $\text{loss}_{\text{adv}}^{(k)}(\theta) = \sum_{i=N_T H+1}^{k} (y^{(i)} - \theta^\top z^{(i)})^2$ be the additional loss incurred over the additional data points $\{z^{(i)}\}_{i=N_T H+1}^{k}$. We establish that for any $-a < 0$,

$$P(\exists k > N_T H : \text{loss}_{\text{adv}}^{(k)}(\theta) < \text{loss}_{\text{adv}}^{(k)}(\theta^*) - a) \leq \exp(-\frac{a}{\sigma^2}).$$

In words, the probability of θ ever outperforming θ^* by a margin a on the additional data is exponentially small in a. The proof considers the random walk $\{Z_k\}_k$

$$Z_k = \text{loss}_{\text{adv}}^{(k)}(\theta) - \text{loss}_{\text{adv}}^{(k)}(\theta^*).$$

Crudely speaking we exploit the fact that no matter how an adversary chooses each additional data point $z^{(i)}$ as a function of the history up to time $i-1$, the random walk $\{Z_k\}_k$ has a positive bias. More precisely, we use the Optional Stopping Theorem on

the martingale $Y_k = \exp(\frac{-1}{2\sigma^2} Z_k)$.[11]

Step 3. Let θ be an inaccurate parameter. From Step 1 we have that the optimal θ^* outperforms θ by a margin $\Omega(N_T)$ after having seen the initial data points $\{z^{(i)}, y^{(i)}\}_{i=1}^{N_T H}$. Step 2 says that the probability for θ to ever make up for this margin $\Omega(N_T)$ is exponentially small in N_T. Our proof combines these two results to show that a "small" number of samples N_T from the teacher is sufficient to guarantee (with high probability) that θ^* has a smaller loss than θ in every iteration, and thus $\theta \notin \{\hat{\theta}^{(k)}\}_{k=N_T H}^{N_T H + k_1 N H}$.

Step 4. Our proof uses a covering argument to extend the result that $\theta \notin \{\hat{\theta}^{(k)}\}_{k=N_T H}^{N_T H + k_1 N H}$ for one specific inaccurate θ from Step 3 to hold for all inaccurate parameters θ simultaneously. As a consequence, the estimated parameters $\hat{\theta}^{(k)}$ throughout all iterations k ($N_T H \leq k \leq N_T H + k_1 N H$) must be accurate. Which establishes Lemma 9.

Theorem 10. *Let any $\delta > 0, \alpha > 0$ be given. Let $\{\hat{M}^{(i)}\}_{i=1}^{N}$ be the models estimated throughout N iterations of the algorithm for the linearly parameterized dynamics case, as described in Section 3.4. Then for $|U_{\hat{M}^{(i)}}(\pi_T) - U_M(\pi_T)| \leq \alpha$ to hold for all $i \in 1:N$ with probability $1-\delta$, it suffices that*

$$N_T = \Omega\left(\text{poly}(\frac{1}{\alpha}, \frac{1}{\delta}, H, R_{\max}, \|A\|_{\text{F}}, \|B\|_{\text{F}}, n_S, n_{\mathcal{A}}, k_1, N)\right).$$

Proof (idea). From Prop. 8 and Lemma 9 we conclude that the estimated models $\{\hat{M}^{(i)}\}_{i=1}^{N}$ are close to the true model in variational distance with high probability for states visited under the teacher's policy. Using the Simulation Lemma gives the resulting accuracy of utility evaluation. □

Theorem 10 shows that a "small" number of samples from the teacher's policy π_T is sufficient to guarantee accurate models $\hat{M}_i^{(i)}$ throughout all iterations of the

[11] **Definition**(Martingale.) Let $(\Omega, \mathcal{F}, P)$ be a probability space with a a filtration $\mathcal{F}_0, \mathcal{F}_1, \cdots$. Suppose that $X_0, X_1, \cdots$ are random variables such that for all $i \geq 0$, X_i is $\mathcal{F}_i$-measurable. The sequence $X_0, X_1, \cdots$ is a martingale provided, for all $i \geq 0$, we have that $\text{E}[X_{i+1}|\mathcal{F}_i] = X_i$. Due to space constraints we can not expand on these concepts here. We refer the reader to, e.g., Durrett (1995); Billingsley (1995); Williams (1991), for more details on martingales and stopping times.

algorithm. An accurate model here means that the utility of the teacher's policy π_T is accurately evaluated in that model, i.e., $U_{\hat{M}^{(i)}}(\pi_T)$ is close to $U_M(\pi_T)$.

3.7.3 Bound on the number of inaccurate states visits

Based on the online learning results for regularized linear regression in Kakade and Ng (2005), we can show the following result.

Lemma 11. *Let any $\mu > 0, \delta > 0$ be given. For the algorithm described in Section 3.4 we have with probability $1 - \delta$ that the number of times a state-action pair (x, u) is encountered such that $\|(A\phi(x) + Bu) - (\hat{A}^{(i)}\phi(x) + \hat{B}^{(i)}u)\|_2 > \mu$ is bounded by $N_\mu = O(k_1\sqrt{k_1 N}(\log k_1 N)^3 \mathrm{poly}(\|A\|_F, \|B\|_F, n_S, n_{\mathcal{A}}, \log \frac{1}{\delta},$
$H, \frac{1}{\mu}))$.*

Lemma 11 is proved in the appendix. Lemma 11 is key to proving the bound on the number of iterations in the algorithm.

3.7.4 Proof of Theorem 4 for linearly parameterized dynamical systems

Proof (rough sketch). The conditions in Eqn. (3.4), (3.5) ensure that $\hat{U}_M(\pi_T)$ and $\{\hat{U}_M(\pi^{(i)}\}_i$ are accurately evaluated with high probability (by the Hoeffding inequality) and Eqn. (3.4) also ensures that $\{U_{\hat{M}^{(i)}}(\pi_T)\}_i$ are accurate estimates of $U_M(\pi_T)$ (by Theorem 10). Using the Simulation Lemma and the same reasoning as in the proof of Lemma 7 gives us that if the algorithm does not terminate in step 4 of the algorithm, then the policy $\pi^{(i)}$ must be visiting a state-action pair (x, u) that satisfies

$$d_{\mathrm{var}}(P(\cdot|x, u), \hat{P}^{(i)}(\cdot|x, u)) > \frac{\alpha}{16H^2 R_{\max}} \tag{3.24}$$

with probability at least $\frac{\alpha}{16HR_{\max}}$. If (x, u) satisfies Eqn. (3.24) then we must have (using Prop. 8) that

$$\|(A\phi(x) + Bu) - (\hat{A}^{(i)}\phi(x) + \hat{B}^{(i)}u)\|_2 > \frac{\sqrt{2\pi}\sigma\alpha}{16H^2 R_{\max}}.$$

From Lemma 11 this can happen only

$$O(k_1\sqrt{k_1 N}(\log k_1 N)^3 \text{poly}(\|A\|_F, \|B\|_F, n_S, n_{\mathcal{A}}, \log\frac{1}{\delta}, H, R_{\max}, \frac{1}{\alpha})) \tag{3.25}$$

times in N iterations of the algorithm. On the other hand, if the algorithm continues, we have from above that such an error must be encountered (with high probability)

$$\Omega(\frac{\alpha}{HR_{\max}}N) \tag{3.26}$$

times. Note that the lower bound on the number of state-action pairs encountered with large error in Eqn. (3.26) grows faster in N than the upper bound in Eqn. (3.25).[12] Once the lower bound is larger than the upper bound we have a contradiction. Thus from Eqn. (3.26) and (3.25) we can conclude that after a number of iterations as given by Eqn. (3.3) the algorithm must have terminated with high probability. Also, since we chose k_1, N_T such that $\{\hat{U}_M(\pi^{(i)}\}_i$ and $\hat{U}_M(\pi_T)$ are accurately evaluated, Eqn. (3.2) holds when the algorithm terminates. □

3.8 Discussion

For many applications, especially in robotics, explicit exploration can be dangerous for the system. In this chapter we presented an algorithm that leverages expert demonstrations to learn a dynamics model without ever requiring explicit exploration. We also provided theoretical guarantees. Concretely, we showed that our algorithm is guaranteed to perform comparably to the expert on the task at hand while requiring only a small number of expert demonstrations and a small number of autonomous trials. For each of the autonomous trials, our algorithm generates a controller which tries to perform as well as possible according to the current dynamics model.

Our results assume the teacher and our controller act in the same Markov decision

[12] In this proof sketch we ignore a dependence of k_1 on N. See Appendix B for a formal proof.

process, where the Markov decision process includes both the robot and its environment. More explicit exploration methods from the system identification literature (see, e.g., Ljung, 1986) try to explicitly excite all modes of the robot, even if the "training" environment might not naturally do so. This can result in performance guarantees, which extend to "testing" environments different from the training environment. A future direction could be to extend our apprenticeship learning approach to also distinguish between (robotic) system and environment. A modified algorithm could ask the teacher to demonstrate good performance on the task in a variety of environments. A clever choice of environments could directly influence the number of demonstrations required.

Chapter 4

Learning First-Order Markov Models for Control

Two of the key challenges when applying traditional control and reinforcement learning methods are the need for (i) an appropriate reward function, and (ii) a dynamics model that is sufficiently accurate. In Chapter 2 we described how we can leverage expert demonstrations to learn the reward function, which alleviates the need for the (often very time-consuming) process of hand-engineering an appropriate reward function. In Chapter 3 we described how we can leverage expert demonstrations to obtain a good dynamics model. While our treatment there focused on the data collection procedure, in this chapter we focus on the estimation procedure: how to estimate the dynamics model from a *given* data set already collected from the system. In Chapter 5, we will describe the parameterization of our helicopter models and present an extension of the algorithms presented in this chapter for that particular setting.

4.1 Introduction

First-order Markov models have enjoyed numerous successes in many sequence modeling and in many control tasks, and are now a workhorse of machine learning.[1]

[1]To simplify the exposition, in this paper we will consider only first-order Markov models. However, the problems we describe in this paper also arise with higher order models and with more

First-order Markov models assume that the next state depends (stochastically) on the current state and current inputs only. More generally, n'th order Markov models assume the next state depends (stochastically) on the state in the last n timesteps and the inputs only.

Even in control problems in which the system is suspected to have hidden state and thus be non-Markov, a fully observed Markov decision process (MDP) model is often favored over partially observable Markov decision process (POMDP) models, since it is significantly easier to solve MDPs than POMDPs to obtain a controller (Kaelbling et al., 1998).

When the parameters of a Markov model are not known a priori, they are often estimated from data using maximum likelihood (ML) (and perhaps smoothing). However, in many applications the dynamics are not truly first-order Markov, and the ML criterion may lead to poor modeling performance. In particular, we will show that the ML model fitting criterion explicitly considers only the first-order (one-step) transitions. If the dynamics are truly governed by a first-order system, then the longer-range interactions would also be well modeled. But if the system is not first-order, then interactions on longer time scales are often poorly approximated by a model fit using maximum likelihood. In reinforcement learning and control tasks where the goal is to maximize our long-term expected rewards, the predictive accuracy of a model on long time scales can have a significant impact on the attained performance.

As a specific motivating example, consider a system whose dynamics are governed by a random walk on the integers. Letting S_t denote the state at time t, we initialize the system to $S_0 = 0$, and let $S_t = S_{t-1} + \varepsilon_t$, where the increments $\varepsilon_t \in \{-1, +1\}$ are equally likely to be -1 or $+1$. Writing S_t in terms of only the ε_t's, we have $S_t = \varepsilon_1 + \cdots + \varepsilon_t$. Thus, if the increments are independent, we have $\mathrm{Var}(S_T) = T$. However if the increments are perfectly correlated (so $\varepsilon_1 = \varepsilon_2 = \cdots$ with probability 1), then $\mathrm{Var}(S_T) = T^2$. So, depending on the correlation between the increments, the expected value $\mathrm{E}[|S_T|]$ can be either $O(\sqrt{T})$ or $O(T)$. Further, regardless of the

structured models such as dynamic Bayesian networks (Ghahramani, 1998; Pearl, 1988) and mixed memory Markov models (Ney et al., 1994; Saul and Jordan, 1999), and it is straightforward to extend our methods and algorithms to these models.

true correlation in the data, using maximum likelihood (ML) to estimate the model parameters from training data would return the same model with $\mathrm{E}[|S_T|] = O(\sqrt{T})$.

To see how these effects can lead to poor performance on a control task, consider learning to control a vehicle (such as a car or a helicopter) under disturbances ε_t due to very strong winds. The influence of the disturbances on the vehicle's position over one time step may be small, but if the disturbances ε_t are highly correlated, their cumulative effect over time can be substantial. If our model completely ignores these correlations, we may overestimate our ability to control the vehicle (thinking our variance in position is $O(T)$ rather than $O(T^2)$), and try to follow overly narrow/dangerous paths.

Our motivation also has parallels in the debate on using discriminative vs. generative algorithms for supervised learning. There, the consensus (assuming there is ample training data) seems to be that it is usually better to directly minimize the loss with respect to the ultimate performance measure, rather than an intermediate loss function such as the likelihood of the training data. (See, e.g., Vapnik, 1998; Ng and Jordan, 2002.) This is because the model (no matter how complicated) is almost always not completely "correct" for the problem data. By analogy, when modeling a dynamical system for a control task, we are interested in having a model that accurately predicts the performance of different control policies—so that it can be used to select a good policy—and not in maximizing the likelihood of the observed sequence data.

In related work, robust control offers an alternative family of methods for accounting for model inaccuracies, specifically by finding controllers that work well for a large class of models. Examples from the literature include Satia and Lave (1973); White and Eldeib (1994); Gahinet et al. (1995). Also, in applied control, some practitioners manually adjust their model's parameters (particularly the model's noise variance parameters) to obtain a model which captures the variability of the system's dynamics. Our work can be viewed as proposing an algorithm that gives a more structured approach to estimating the "right" variance parameters. The issue of time scales has also been addressed in hierarchical reinforcement learning (e.g., Dietterich,

2000; Sutton, 1995; Precup et al., 1998), but most of this work has focused on speeding up exploration and planning rather than on accurately modeling non-Markovian dynamics.

The rest of this chapter is organized as follows. We define our notation in Section 4.2, then formulate the model learning problem ignoring actions in Section 4.3, and propose a learning algorithm in Section 4.4. In Section 4.5, we extend our algorithm to incorporate actions. Section 4.6 presents experimental results. Section 4.7 concludes the chapter with a discussion.

In the next chapter, we consider the application of this chapter's ideas to modeling vehicular dynamics, with an emphasis on helicopter modeling.

The work described in this chapter has first appeared in Abbeel and Ng (2005b).

4.2 Preliminaries

If $x \in \mathbb{R}^n$, then x_i denotes the i-th element of x. Also, let $j\colon k = [j \;\; j+1 \;\; j+2 \;\; \ldots k-1 \;\; k]^T$. For any k-dimensional vector of indices $I \in \mathbf{N}^k$, we denote by x_I the k-dimensional vector with the subset of x's entries whose indices are in I. For example, if $x = [0.0 \;\; 0.1 \;\; 0.2 \;\; 0.3 \;\; 0.4 \;\; 0.5]^T$, then $x_{0:2} = [0.0 \;\; 0.1 \;\; 0.2]^T$.

A finite-state decision process (DP) is a tuple (S, A, T, γ, D, R), where S is a finite set of states; A is a finite set of actions; $T = \{P(S_{t+1} = s' | S_{0:t} = s_{0:t}, A_{0:t} = a_{0:t})\}$ is a set of state transition probabilities (here, $P(S_{t+1} = s' | S_{0:t} = s_{0:t}, A_{0:t} = a_{0:t})$ is the probability of being in a state $s' \in S$ at time $t+1$ after having taken actions $a_{0:t} \in A^{t+1}$ in states $s_{0:t} \in S^{t+1}$ at times $0:t$); $\gamma \in [0, 1)$ is a discount factor; D is the initial state distribution, from which the initial state s_0 is drawn; and $R : S \mapsto \mathbb{R}$ is the reward function. We assume all rewards are bounded in absolute value by $R_{\max}$. A DP is not necessarily Markov.

A policy π is a mapping from states to probability distributions over actions. Let $V^\pi(s) = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s]$ be the usual value function for π. Then the utility of π is

$$U(\pi) = \mathrm{E}_{s_0 \sim D}[V^\pi(s_0)] = \mathrm{E}[\textstyle\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi] = \sum_{t=0}^{\infty} \gamma^t \sum_{s_t} P(S_t = s_t | \pi) R(s_t).$$

The second expectation above is with respect to the random state sequence $s_0, s_1, \ldots$ drawn by starting from $s_0 \sim D$, picking actions according to π and transitioning according to P.

Throughout this paper, $P_{\hat{\theta}}$ will denote some estimate of the transition probabilities. We denote by $\hat{U}(\pi)$ the utility of the policy π in an MDP whose first-order transition probabilities are given by $P_{\hat{\theta}}$ (and similarly $\hat{V}^{\pi}$ the value function in the same MDP). Thus, we have[2]

$$\hat{U}(\pi) = \hat{\mathrm{E}}_{s_0 \sim D}[\hat{V}^{\pi}(s_0)] = \hat{\mathrm{E}}[\textstyle\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] = \sum_{t=0}^{\infty} \gamma^t \sum_{s_t} P_{\hat{\theta}}(S_t = s_t|\pi) R(s_t).$$

Note that if $|U(\pi) - \hat{U}(\pi)| \leq \varepsilon$ for all π, then finding the optimal policy in the estimated MDP that uses parameters $P_{\hat{\theta}}$ (using value iteration or any other algorithm) will give a policy whose utility is within 2ε of the optimal utility. (Kearns et al., 1999)

For stochastic processes without decisions/actions, we will use the same notation but drop the conditioning on π. Often we will also abbreviate $P(S_t = s_t)$ by $P(s_t)$.

4.3 Problem Formulation

To simplify our exposition, we will begin by considering stochastic processes that do not have decisions/actions. Section 4.5 will discuss how actions can be incorporated into the model.

We first consider how well $\hat{V}(s_0)$ approximates $V(s_0)$. We have

$$\begin{aligned} |\hat{V}(s_0) - V(s_0)| &= \left| \sum_{t=0}^{\infty} \gamma^t \sum_{s_t} P_{\hat{\theta}}(s_t|s_0) R(s_t) - \sum_{t=0}^{\infty} \gamma^t \sum_{s_t} P(s_t|s_0) R(s_t) \right| \\ &\leq R_{\max} \sum_{t=0}^{\infty} \gamma^t \sum_{s_t} \left| P_{\hat{\theta}}(s_t|s_0) - P(s_t|s_0) \right|. \end{aligned} \tag{4.1}$$

So, to ensure that $\hat{V}(s_0)$ is an accurate estimate of $V(s_0)$, we would like the parameters

[2]Since $P_{\hat{\theta}}$ is a first-order model, it explicitly parameterizes only $P_{\hat{\theta}}(S_{t+1} = s_{t+1}|S_t = s_t, A_t = a_t)$. We use $P_{\hat{\theta}}(S_t = s_t|\pi)$ to denote the probability that $S_t = s_t$ in an MDP with one-step transition probabilities $P_{\hat{\theta}}(S_{t+1} = s_{t+1}|S_t = s_t, A_t = a_t)$ and initial state distribution D when acting according to the policy π.

$\hat{\theta}$ of the model to minimize the right hand side of (4.1). The term $\sum_{s_t} |P_{\hat{\theta}}(s_t|s_0) - P(s_t|s_0)|$ is exactly (twice) the variational distance between the two conditional distributions $P_{\hat{\theta}}(\cdot|s_0)$ and $P(\cdot|s_0)$. Unfortunately P is not known when learning from data. We only get to observe state sequences sampled according to P. This makes Eqn. (4.1) a difficult criterion to optimize. However, it is well known that the variational distance is upper bounded by a function of the KL-divergence. (See, e.g., Cover and Thomas, 1991.) The KL-divergence between P and $P_{\hat{\theta}}$ can be estimated (up to a constant) as the log-likelihood of a sample. So, given a training sequence $s_{0:T}$ sampled from P, we propose to estimate the transition probabilities $P_{\hat{\theta}}$ by

$$\hat{\theta} = \arg\max_\theta \sum_{t=0}^{T-1} \sum_{k=1}^{T-t} \gamma^k \log P_\theta(s_{t+k}|s_t). \tag{4.2}$$

Note the difference between this and the standard maximum likelihood (ML) estimate. Since we are using a model that is parameterized as a first-order Markov model, the probability of the data under the model is given by $P_\theta(s_0, \ldots, s_T) = P_\theta(s_T|s_{T-1})P_\theta(s_{T-1}|s_{T-2}) \ldots P_\theta(s_1|s_0)D(s_0)$ (where D is the initial state distribution). By definition, maximum likelihood (ML) chooses the parameters θ that maximize the probability of the observed data. Taking logs of the probability above, (and ignoring $D(s_0)$, which is usually parameterized separately), we find that the ML estimate is given by

$$\hat{\theta} = \arg\max_\theta \sum_{t=0}^{T-1} \log P_\theta(s_{t+1}|s_t). \tag{4.3}$$

All the terms above are of the form $P_\theta(s_{t+1}|s_t)$. Thus, the ML estimator explicitly considers, and tries to model well, *only the observed one-step transitions*. In Figure 4.1 we use Bayesian network notation to illustrate the difference between the two objectives for a training sequence of length four. Figure 4.1(a) shows the training sequence, which can have arbitrary dependencies. Maximum likelihood (ML) estimation maximizes $f_{ML}(\theta) = \log P_\theta(s_1|s_0) + \log P_\theta(s_2|s_1) + \log P_\theta(s_3|s_2)$. Figure 4.1(b) illustrates the interactions modeled by ML. Ignoring γ for now, for this example our objective (Eqn. 4.2) is $f_{ML}(\theta) + \log P_\theta(s_2|s_0) + \log P_\theta(s_3|s_1) + \log P_\theta(s_3|s_0)$. Thus, it takes into account both the interactions in Figure 4.1(b) as well as the longer-range

ones in Figure 4.1(c).

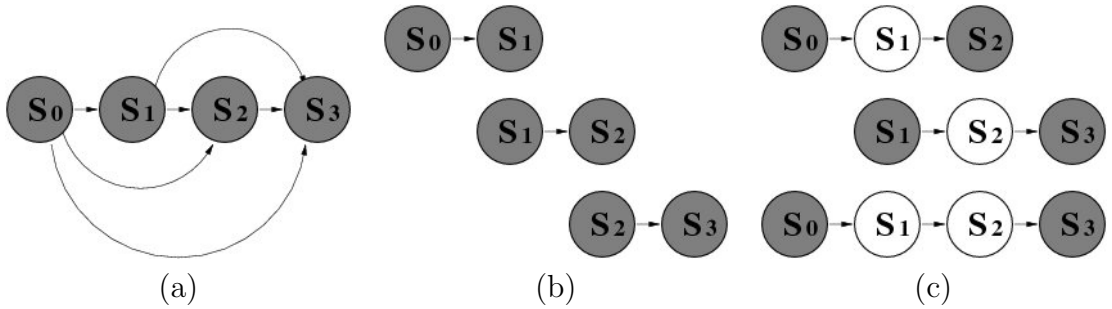


Figure 4.1: (a) A length four training sequence. (b) ML estimation for a first-order Markov model optimizes the likelihood of the second node given the first node in each of the length two subsequences. (c) Our objective (Eqn. 4.2) also includes the likelihood of the last node given the first node in each of these three longer subsequences of the data. (White nodes represent unobserved variables, shaded nodes represent observed variables.)

4.4 Algorithm

We now present an EM algorithm for optimizing the objective in Eqn. (4.2) for a first-order Markov model.[3] Our algorithm is derived using the method of Neal and Hinton (1999). (See Appendix C for details.) The algorithm iterates between the following two steps:

- *E-step: Compute expected counts*
  - $\forall i, j \in S$, set $stats(j, i) = 0$
  - $\forall t : 0 \leq t \leq T-1, \forall k : 1 \leq k \leq T-t, \forall l : 0 \leq l \leq k-1, \forall i, j \in S$ $stats(j, i) \mathrel{+}= \gamma^k P_{\hat{\theta}}(S_{t+l+1} = j, S_{t+l} = i | S_t = s_t, S_{t+k} = s_{t+k})$
- *M-step: Re-estimate model parameters*

  Update $\hat{\theta}$ such that $\forall i, j \in S$, $P_{\hat{\theta}}(j|i) = stats(j, i) / \sum_{k \in S} stats(k, i)$

[3]Using higher order Markov models or more structured models—such as dynamic Bayesian networks (Ghahramani, 1998; Pearl, 1988) or mixed memory Markov models (Ney et al., 1994; Saul and Jordan, 1999)—presents no special difficulties. However, the notation becomes more involved and the inference (in the E-step) might become more expensive.

Prior to starting EM, the transition probabilities $P_{\hat{\theta}}$ can be initialized with the first-order transition counts (i.e., the ML estimate of the parameters), possibly with smoothing.[4]

Let us now consider more carefully the computation done in the E-step for one specific pair of values for t and k (corresponding to one term $\log P_\theta(s_{t+k}|s_t)$ in Eqn. 4.2). For $k \geq 2$, as in the forward-backward algorithm for HMMs (see, e.g., Rabiner, 1989; Pearl, 1988), the pairwise marginals can be computed by a forward propagation (computing the forward messages), a backward propagation (computing the backward messages), and then combining the forward and backward messages.[5] Forward and backward messages are computed recursively:

$$\text{for} \quad l = 1 \text{ to } k-1, \quad \forall i \in S \quad m_{\rightarrow t+l}(i) = \textstyle\sum_{j \in S} m_{\rightarrow t+l-1}(j) P_{\hat{\theta}}(i|j), \qquad (4.4)$$

$$\text{for} \quad l = k-1 \text{ down to } 1, \quad \forall i \in S \quad m_{t+l\leftarrow}(i) = \textstyle\sum_{j \in S} m_{t+l+1\leftarrow}(j) P_{\hat{\theta}}(j|i), \qquad (4.5)$$

where we initialize $m_{\rightarrow t}(i) = \mathbf{1}\{i = s_t\}$, and $m_{t+k\leftarrow}(i) = \mathbf{1}\{i = s_{t+k}\}$. The pairwise marginals can be computed by combining the forward and backward messages:

$$P_{\hat{\theta}}(S_{t+l+1} = j, S_{t+l} = i | S_t = s_t, S_{t+k} = s_{t+k}) = m_{\rightarrow t+l}(i) P_{\hat{\theta}}(j|i) m_{t+l+1\leftarrow}(j). \qquad (4.6)$$

For the term $\log P_\theta(s_{t+k}|s_t)$, we end up performing $2(k-1)$ message computations, and combining messages into pairwise marginals $k-1$ times. Doing this for all terms in the objective results in $O(T^3)$ message computations and $O(T^3)$ computations of pairwise marginals from these messages. In practice, the objective (4.2) can be approximated by considering only the terms in the summation with $k \leq H$, where H is some time horizon.[6] In this case, the computational complexity is reduced to $O(TH^2)$.

[4] A parameter $P_{\hat{\theta}}(j|i)$ initialized to zero will remain zero throughout successive iterations of EM. If this is undesirable, then smoothing could be used to eliminate zero initial values.

[5] Note that the special case $k = 1$ (and thus $l = 0$) does not require inference. In this case we simply have $P_{\hat{\theta}}(S_{t+1} = j, S_t = i | S_t = s_t, S_{t+1} = s_{t+1}) = \mathbf{1}\{i = s_t\}\mathbf{1}\{j = s_{t+1}\}$.

[6] Because of the discount term γ^k in the objective (4.2), one can safely truncate the summation over k after about $O(1/(1-\gamma))$ terms without incurring too much error.

4.4.1 Computational Savings

The following observation leads to substantial savings in the number of message computations. The forward messages computed for the term $\log P_\theta(s_{t+k}|s_t)$ depend only on the value of s_t. So the forward messages computed for the terms $\{\log P_\theta(s_{t+k}|s_t)\}_{k=1}^{H}$ are the same as the forward messages computed just for the term $\log P_\theta(s_{t+H}|s_t)$. A similar observation holds for the backward messages. As a result, we need to compute only $O(TH)$ messages (as opposed to $O(TH^2)$ in the naive algorithm).

The following observation leads to further, (even more substantial) savings. Consider two terms in the objective $\log P_\theta(s_{t_1+k}|s_{t_1})$ and $\log P_\theta(s_{t_2+k}|s_{t_2})$. If $s_{t_1} = s_{t_2}$ and $s_{t_1+k} = s_{t_2+k}$, then both terms will have exactly the same pairwise marginals and contribution to the expected counts. So expected counts have to be computed only once for every triple i, j, k for which $(S_t = i, S_{t+k} = j)$ occurs in the training data. As a consequence, the running time for each iteration (once we have made an initial pass over the data to count the number of occurrences of the triples) is only $O(|S|^2H^2)$, which is independent of the size of the training data.

4.5 Incorporating actions

In decision processes, actions influence the state transition probabilities. To generate training data, suppose we choose an exploration policy and take actions in the DP using this policy. Given the resulting training data, and generalizing Eqn. (4.2) to incorporate actions, our estimator now becomes

$$\hat{\theta} = \arg\max_\theta \sum_{t=0}^{T-1} \sum_{k=1}^{T-t} \gamma^k \log P_\theta(s_{t+k}|s_t, a_{t:t+k-1}). \tag{4.7}$$

The EM algorithm is straightforwardly extended to this setting, by conditioning on the actions during the E-step, and updating state-action transition probabilities $P_\theta(j|i, a)$ in the M-step.

As before, forward messages need to be computed only once for each value of t, and backward messages only once for each value of $t+k$. However achieving the more

substantial savings, as described in the second paragraph of Section 4.4.1, is now more difficult. In particular, now the contribution of a triple i, j, k (one for which $(S_t = i, S_{t+k} = j)$ occurs in the training data) depends on the action sequence $a_{t:t+k-1}$. The number of possible sequences of actions $a_{t:t+k-1}$ grows exponentially with k.

If, however, we use a deterministic exploration policy to generate the training data (more specifically, one in which the action taken is a deterministic function of the current state), then we can again obtain these computational advantages: Counts of the number of occurrences of the triples described previously are now again a sufficient statistic. However, a single deterministic exploration policy, by definition, cannot explore all state-action pairs. Thus, we will instead use a combination of several deterministic exploration policies, which jointly can explore all state-action pairs. In this case, the running time for the E-step becomes $O(|S|^2 H^2 |\Pi|)$, where $|\Pi|$ is the number of different deterministic exploration policies used. (See Section 6.2 for an example.)

4.6 Experiments

In this section, we empirically study the performance of model fitting using our proposed algorithm, and compare it to the performance of ordinary ML estimation.

4.6.1 Shortest vs. safest path

Consider an agent acting for 100 time steps in the grid-world in Figure 4.2(a). The initial state is marked by S, and the absorbing goal state by G. The reward is -500 for the gray squares, and -1 elsewhere. This DP has four actions that (try to) move in each of the four compass directions, and succeed with probability $1-p$. If an action is not successful, then the agent's position transitions to one of the neighboring squares. Similar to our example in Section 1, the random transitions (resulting from unsuccessful actions) may be correlated over time. In this problem, if there is no noise ($p = 0$), the optimal policy is to follow one of the shortest paths to the goal that do not pass through gray squares, such as path A. For higher noise levels, the optimal

policy is to stay as far away as possible from the gray squares, and try to follow a longer path such as B to the goal.[7] At intermediate noise levels, the optimal policy is strongly dependent on how correlated the noise is between successive time steps. The larger the correlation, the more dangerous path A becomes (for reasons similar to the random walk example in Section 1). In our experiments, we compare the behavior of our algorithm and ML estimation with different levels of noise correlation.[8]

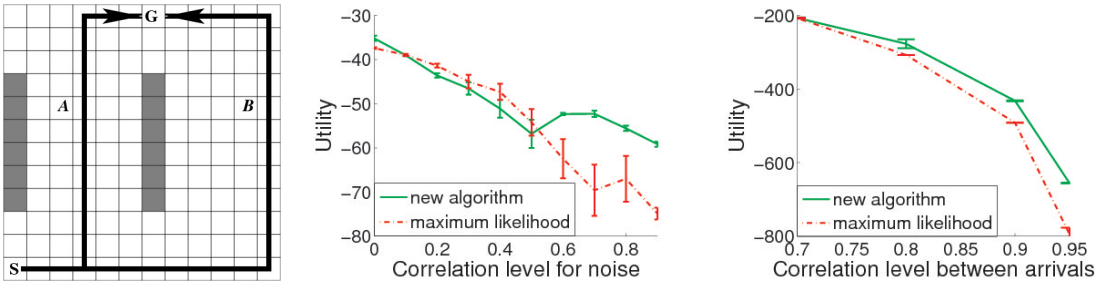
Figure 4.2(b) shows the utilities obtained by the two different models, under different degrees of correlation in the noise. The two algorithms perform comparably when the correlation is weak, but our method outperforms ML when there is strong correlation. Empirically, when the noise correlation is high, our algorithm seems to be fitting a first-order model with a larger "effective" noise level. When the resulting estimated MDP is solved, this gives more cautious policies, such as ones more inclined to choose path B over A. In contrast, the ML estimate performs poorly in this problem because it tends to underestimate how far sideways the agent tends to move due to the noise (cf. the example in Section 4.1).

4.6.2 Queue

We consider a service queue in which the average arrival rate is p. Thus, $p = P(\text{a customer arrives in one time step})$. Also, for each action i, let q_i denote the service rate under that action (thus, $q_i = P(\text{a customer is served in one time step}|\text{action} = i)$). In our problem, there are three service rates $q_0 < q_1 < q_2$ with respective rewards $0, -1, -10$. The maximum queue size is 20, and the reward for any state of the queue is 0, except when the queue becomes full, which results in a reward of -1000. The

[7] For very high noise levels (e.g. $p = 0.99$) the optimal policy is qualitatively different again.

[8] Experimental details: The noise is governed by an (unobserved) Markov chain with four states corresponding to the four compass directions. If an action at time t is not successful, the agent moves in the direction corresponding to the state of this Markov chain. On each step, the Markov chain stays in the current state with probability q, and transitions with probability $1 - q$ uniformly to any of the four states. Our experiments are carried out varying q from 0 (low noise correlation) to 0.9 (strong noise correlation). A 200,000 length state-action sequence for the grid-world, generated using a random exploration policy, was used for model fitting, and a constant noise level $p = 0.3$ was used in the experiments. Given a learned MDP model, value iteration was used to find the optimal policy for it. To reduce computation, we only included the terms of the objective (Eqn. 4.7) for which $k = 10$.



(a) (b) (c)

Figure 4.2: (a) Grid-world. (b) Grid-world experimental results, showing the utilities of policies obtained from the MDP estimated using ML (dash-dot line), and utilities of policies obtained from the MDP estimated using our objective (solid line). Results shown are means over 5 independent trials, and the error bars show one standard error for the mean. The horizontal axis (correlation level for noise) corresponds to the parameter q in the experiment description. (c) Queue experiment, showing utilities obtained using ML (dash-dot line), and using our algorithm (solid line). Results shown are means over 5 independent trials, and the error bars show one standard error for the mean. The horizontal axis (correlation level between arrivals) corresponds to the parameter b in the experiment description. (Best viewed in color.)

service rates are $q_0 = 0$, $q_1 = p$ and $q_2 = 0.75$. So the inexpensive service rate q_1 is sufficient to keep up with arrivals on average. However, even though the average arrival rate is p, the arrivals come in "bursts," and even the high service rate q_2 is insufficient to keep the queue small during the bursts of many consecutive arrivals.[9]

[9]Experimental details: The true process has two different (hidden) modes for arrivals. The first mode has a very low arrival rate, and the second mode has a very high arrival rate. We denote the steady state distribution over the two modes by (ϕ_1, ϕ_2). (I.e., the system spends a fraction ϕ_1 of the time in the low arrival rate mode, and a fraction $\phi_2 = 1 - \phi_1$ of the time in high arrival rate mode.) Given the steady state distribution, the state transition matrix $[a \;\; 1-a; 1-b \;\; b]$ has only one remaining degree of freedom, which (essentially) controls how often the system switches between the two modes. (Here, a [resp. b] is the probability, if we are in the slow [resp. fast] mode, of staying in the same mode the next time step.) More specifically, assuming $\phi_1 > \phi_2$, we have $b \in [0, 1]$, $a = 1 - (1-b)\phi_2/\phi_1$. The larger b is, the more slowly the system switches between modes. Our experiments used $\phi_1 = 0.8, \phi_2 = 0.2, P(\text{arrival}|\text{mode 1}) = 0.01, P(\text{arrival}|\text{mode 2}) = 0.99$. This means $b = 0.2$ gives independent arrival modes for consecutive time steps. In our experiments, $q_0 = 0$, and q_1 was equal to the average arrival rate $p = \phi_1 P(\text{arrival}|\text{mode 1}) + \phi_2 P(\text{arrival}|\text{mode 2})$. Note that the highest service rate $q_2 (= 0.75)$ is lower than the fast mode's arrival rate. Training data was generated using 8000 simulations of 25 time steps each, in which the queue length is initialized randomly, and the same (randomly chosen) action is taken on all 25 time steps. To reduce computational requirements, we only included the terms of the objective (Eqn. 4.7) for which $k = 20$. We used a discount factor $\gamma = .95$ and approximated utilities by truncating at a finite horizon of 100. Note that although we explain the queuing process by arrival/departure rates, the algorithm learns full transition matrices for each action, and not only the arrival/departure rates.

Experimental results on the queue are shown in Figure 4.2(c). We plot the utilities obtained using each of the two algorithms for high arrival correlations. (Both algorithms perform essentially identically at lower correlation levels.) We see that the policies obtained with our algorithm consistently outperform those obtained using maximum likelihood to fit the model parameters. As expected, the difference is more pronounced for higher correlation levels, i.e., when the true model is less well approximated by a first-order model.

For learning the model parameters, we used three deterministic exploration policies, each corresponding to always taking one of the three actions. Thus, we could use the more efficient version of the algorithm described in the second paragraph of Section 4.4.1 and at the end of Section 4.5. A single EM iteration for the experiments on the queue took 6 minutes for the original version of the algorithm, but took only 3 seconds for the more efficient version; this represents more than a 100-fold speedup.

4.7 Discussion

We proposed a method for learning a first-order Markov model that captures the system's dynamics on longer time scales than a single time step. In our experiments, this method outperformed the standard maximum likelihood model. As the policy for data collection can affect the first order model learned, it could be interesting to characterize this dependency, and explore the possible benefits of an iterative procedure, during which the current policy is in turn used for additional data collection.

Chapter 5

Learning Helicopter Models

In this chapter we describe the parameterization of our helicopter models and present experimental results on helicopter modeling. We also present an efficient algorithm for (approximately) optimizing the lagged criterion presented in Chapter 4 for continuous state-action space models and apply it to helicopter modeling.

In Chapters 7 and 8 we extend the *global* non-linear models presented in this chapter to *locally weighted* non-linear models to further improve modeling accuracy in the vicinity of trajectories from which we have flight data.

5.1 Introduction

Helicopter aerodynamics are, to date, somewhat poorly understood, and (unlike most fixed-wing aircraft) no textbook models will accurately predict the dynamics of a helicopter from only its dimensions and specifications (Leishman, 2000; Seddon, 1990). Thus, at least part of the dynamics must be learned from data.

CIFER® (Comprehensive Identification from Frequency Responses) is the industry standard for learning linear models for helicopter (and other rotorcraft) from data (Tischler and Cauffman, 1992; Mettler et al., 1999). CIFER uses frequency response methods to identify a linear model. As CIFER only handles *linear* models, the models obtained from CIFER fail to capture some important aspects of the helicopter dynamics, such as the effects of inertia. Consider a setting in which the helicopter

is flying forward, and suddenly turns sideways. Due to inertia, the helicopter will continue to travel in the same direction as before, so that it has "sideslip," meaning that its orientation is not aligned with its direction of motion. This is a non-linear effect that depends both on velocity and angular rates. The linear CIFER model is unable to capture this.

In fact, the models used in Bagnell and Schneider (2001); Ng et al. (2004b); Mettler et al. (1999) all suffer from this same problem. The core of the problem is that the naive body-coordinate representation used in these settings makes it fundamentally difficult for the learning algorithm to capture certain properties of dynamical systems such as inertia and gravity. As such, one places a significantly heavier burden than is necessary on the learning algorithm.

Similar to La Civita et al. (2006) and Gavrilets et al. (2002b), we describe a parameterization for modeling dynamical systems that does not suffer from this deficiency by directly modeling the accelerations of the helicopter as a non-linear function of its current state and inputs.

Our approach is a hybrid of physical knowledge and learning: Although helicopter dynamics are not fully understood, there are also many properties—such as the direction and magnitude of acceleration due to gravity; the effects of inertia; symmetry properties of the dynamical system; and so on—which apply to *all* dynamical systems, and which are well-understood. All of this can therefore be encoded as prior knowledge, and there is little need to demand that our learning algorithms learn them.

While the *global* non-linear parameterization described in this chapter might seem somewhat restrictive at first, it forms the basis for the helicopter models we used to obtain our aerobatic flight results. In Chapters 7 and 8 we describe how the global non-linear models presented in this chapter can be extended to *locally weighted* non-linear models to further improve modeling accuracy in the vicinity of trajectories from which we have flight data.

Given any model class, we can choose the parameter learning criterion used to learn a model within the class. CIFER finds the parameters that minimize a frequency domain error criterion. Alternatively, we can minimize the squared one-step prediction error in the time domain. Forward simulation on a held-out test set is a

standard way to assess model quality, and we use it to compare the linear models learned using CIFER to the same linear models learned by optimizing the one-step prediction error. Following up on our work described in Chapter 4 one can also learn parameters so as to optimize a "lagged criterion" that directly measures simulation accuracy—i.e., predictive accuracy of the model over long time scales. However, the EM algorithm described in Chapter 4 can be computationally expensive when applied in a continuous state-space setting. We present an efficient algorithm that approximately optimizes the lagged criterion. Our experiments show that the resulting model consistently outperforms the linear models trained using CIFER or using the one-step error criterion. Combining this with the acceleration based parameterization results in our best helicopter model.

The work described in this chapter has first appeared in Abbeel, Ganapathi and Ng (2006).

5.2 Helicopter state, input and dynamics

The helicopter state s comprises its position (x, y, z), orientation (roll ϕ, pitch θ, yaw ω), velocity $(\dot{x}, \dot{y}, \dot{z})$ and angular velocity $(\dot{\phi}, \dot{\theta}, \dot{\omega})$. The helicopter is controlled via a 4-dimensional action space:

1. u_1 and u_2: The longitudinal (front-back) and latitudinal (left-right) cyclic pitch controls cause the helicopter to pitch forward/backward or sideways, and can thereby also affect acceleration in the longitudinal and latitudinal directions.

2. u_3: The tail rotor collective pitch control affects tail rotor thrust, and can be used to yaw (turn) the helicopter.

3. u_4: The main rotor collective pitch control affects the pitch angle of the main rotor's blades, by rotating the blades around an axis that runs along the length of the blade. As the main rotor blades sweep through the air, the resulting amount of upward thrust (generally) increases with this pitch angle; thus this control affects the main rotor's thrust.

Following standard practice in helicopter system identification (Ng et al., 2004b; Mettler et al., 1999), the original 12-dimensional helicopter state is reduced to an 8-dimensional state represented in body (or robot-centric) coordinates $s^b = (\phi,\ \theta,\ \dot{x},\ \dot{y},\ \dot{z},\ \dot{\phi},\ \dot{\theta}, \dot{\omega})$. Where there is risk of confusion, we will use superscript s and b to distinguish between spatial (world) coordinates and body coordinates. The body coordinate representation specifies the helicopter state using a coordinate frame in which the x, y, and z axes are forwards, sideways, and down relative to the current orientation of the helicopter, instead of north, east and down. Thus, $\dot{x}^b$ is the forward velocity, whereas $\dot{x}^s$ is the velocity in the northern direction. (ϕ and θ are always expressed in world coordinates, because roll and pitch relative to the body coordinate frame is always zero.) By using a body coordinate representation, we encode into our model certain "symmetries" of helicopter flight, such as that the helicopter's dynamics are the same regardless of its absolute position (x, y, z) and heading ω (assuming the absence of obstacles). Even in the reduced coordinate representation, only a subset of the state variables needs to be modeled explicitly using learning. Given a model that predicts only the angular velocities $(\dot{\phi}, \dot{\theta}, \dot{\omega})$, we can numerically integrate to obtain the orientation (ϕ, θ, ω).

We can integrate the reduced body coordinate states to obtain the complete world coordinate states. Integrating body-coordinate angular velocities to obtain world-coordinate angles is nonlinear, thus the model resulting from this process is necessarily nonlinear.

5.3 Linear model

The linear model we learn with CIFER has the following form:

$$
\begin{aligned}
\dot{\phi}^b_{t+1} - \dot{\phi}^b_t &= \left(C_\phi \dot{\phi}^b_t + C_1(u_1)_t + D_1\right)\Delta t, & \dot{x}^b_{t+1} - \dot{x}^b_t &= \left(C_x \dot{x}^b_t - g\theta_t\right)\Delta t,\\
\dot{\theta}^b_{t+1} - \dot{\theta}^b_t &= \left(C_\theta \dot{\theta}^b_t + C_2(u_2)_t + D_2\right)\Delta t, & \dot{y}^b_{t+1} - \dot{y}^b_t &= \left(C_y \dot{y}^b_t + g\phi_t + D_0\right)\Delta t,\\
\dot{\omega}^b_{t+1} - \dot{\omega}^b_t &= \left(C_\omega \dot{\omega}^b_t + C_3(u_3)_t + D_3\right)\Delta t, & \dot{z}^b_{t+1} - \dot{z}^b_t &= \left(C_z \dot{z}^b_t + g + C_4(u_4)_t + D_4\right)\Delta t,\\
\phi_{t+1} - \phi_t &= \dot{\phi}^b_t \Delta t, & \theta_{t+1} - \theta_t &= \dot{\theta}^b_t \Delta t.
\end{aligned}
$$

Here $g = 9.81m/s^2$ is the acceleration due to gravity and Δt is the time discretization, which is 0.1 seconds in our experiments. The free parameters in the model are $C_x, C_y, C_z, C_\phi, C_\theta, C_\omega$, which model damping, and $D_0, C_1, D_1, C_2, D_2, C_3, D_3, C_4, D_4$, which model the influence of the inputs on the states.[1] This parameterization was chosen using the "coherence" feature selection algorithm of CIFER. CIFER takes as input the state-action sequence $\{(\dot{x}^b_t, \dot{y}^b_t, \dot{z}^b_t, \dot{\phi}^b_t, \dot{\theta}^b_t, \dot{\omega}^b_t, \phi_t, \theta_t, u_t)\}_t$ and learns the free parameters using a frequency domain cost function. See Tischler and Cauffman (1992) for details.

Frequency response methods (as used in CIFER) are not the only way to estimate the free parameters. Instead, we can minimize the average squared prediction error of next state given current state and action. Doing so only requires linear regression. In our experiments (see Section 5.6) we compare the simulation accuracy over several time-steps of the differently learned linear models. We also compare to learning by directly optimizing the simulation accuracy over several time-steps. The latter approach is presented in Section 5.5.

5.4 Acceleration prediction model

Due to inertia, if a forward-flying helicopter turns, it will have sideslip (i.e., the helicopter will not be aligned with its direction of motion). The linear model is unable to capture the sideslip effect, since this effect depends non-linearly on velocity and angular rates. In fact, the models used in Bagnell and Schneider (2001); Ng et al. (2004b); Mettler et al. (1999) all suffer from this problem. More generally, these models do not capture conservation of momentum well. Although careful engineering of (many) additional non-linear features might fix individual effects such as, e.g., sideslip, it is unclear how to capture inertia compactly in the naive body-coordinate representation.

From physics, we have the following update equation for velocity when expressed

[1] D_0 captures the sideways acceleration caused by the tail rotor's thrust.

in body-coordinates:

$$(\dot{x}, \dot{y}, \dot{z})^b_{t+1} = R\left((\dot{\phi}, \dot{\theta}, \dot{\omega})^b_t\right) * \left((\dot{x}, \dot{y}, \dot{z})^b_t + (\ddot{x}, \ddot{y}, \ddot{z})^b_t \Delta t\right). \tag{5.1}$$

Here, $R\left((\dot{\phi}, \dot{\theta}, \dot{\omega})^b_t\right)$ is the rotation matrix that transforms from the body-coordinate frame at time t to the body-coordinate frame at time $t+1$ (and is determined by the angular velocity $(\dot{\phi}, \dot{\theta}, \dot{\omega})^b_t$ at time t); and $(\ddot{x}, \ddot{y}, \ddot{z})^b_t$ denotes the acceleration vector in body-coordinates at time t. Forces and torques (and thus accelerations) are often a fairly simple function of inputs and state. This suggests that a model which learns to predict the accelerations, and then uses Eqn. (5.1) to obtain velocity over time, may perform well. Such a model would naturally capture inertia, by using the velocity update of Eqn. (5.1). In contrast, the models of Section 5.3 try to predict changes in body-coordinate velocity. *But the change in body-coordinate velocity does not correspond directly to physical accelerations, because the body-coordinate velocity at times t and $t+1$ are expressed in different coordinate frames.* Thus, $\dot{x}^b_{t+1} - \dot{x}^b_t$ is not the forward acceleration—because $\dot{x}^b_{t+1}$ and $\dot{x}^b_t$ are expressed in different coordinate frames. To capture inertia, these models therefore need to predict not only the physical accelerations, but also the non-linear influence of the angular rates through the rotation matrix. This makes for a difficult learning problem, and puts an unnecessary burden on the learning algorithm. Our discussion above has focused on linear velocity. The body-coordinate angular velocity vector is an eigenvector of the rotation matrix $R\left((\dot{\phi}, \dot{\theta}, \dot{\omega})^b_t\right)$, and hence when we use Eqn. (5.1) and can leave out the rotation matrix.

The previous discussion suggests that we learn to *predict physical accelerations* and then integrate the accelerations to obtain the state trajectories. To do this, we propose:

$$\begin{aligned}
\ddot{\phi}^b_t &= C_\phi \dot{\phi}_t + C_1(u_1)_t + D_1, \quad \ddot{x}^b_t = C_x \dot{x}^b_t + (g_x)^b_t, \\
\ddot{\theta}^b_t &= C_\theta \dot{\theta}_t + C_2(u_2)_t + D_2, \quad \ddot{y}^b_t = C_y \dot{y}^b_t + (g_y)^b_t + D_0, \\
\ddot{\omega}^b_t &= C_\omega \dot{\omega}_t + C_3(u_3)_t + D_3, \quad \ddot{z}^b_t = C_z \dot{z}^b_t + (g_z)^b_t + C_4(u_4)_t + D_4.
\end{aligned}$$

Here $(g_x)^b_t, (g_y)^b_t, (g_z)^b_t$ are the components of the gravity acceleration vector in each of the body-coordinate axes at time t; and $C., D.$ are the free parameters to be learned from data. The model predicts accelerations in the body-coordinate frame, and is therefore able to take advantage of the same invariants as discussed earlier, such as invariance of the dynamics to the helicopter's (x, y, z) position and heading (ω). Further, it additionally captures the fact that the dynamics are invariant to roll (ϕ) and pitch (θ) once the (known) effects of gravity are subtracted out.

Frequency domain techniques cannot be used to learn the acceleration model above, because it is non-linear. Nevertheless, the parameters can be learned as easily as for the linear model in the time domain: Linear regression can be used to find the parameters that minimize the squared error of the one-step prediction in acceleration.[2]

5.5 The lagged error criterion

To evaluate the performance of a dynamical model, it is standard practice to run a simulation using the model for a certain duration, and then compare the simulated trajectory with the real state trajectory. To do well on this evaluation criterion, it is therefore important for the dynamical model to give not only accurate one-step predictions, but also predictions that are accurate at longer time-scales. Motivated by this, in Chapter 4 we suggested learning the model parameters by optimizing the following "lagged criterion":

$$\sum_{t=1}^{T-H} \sum_{h=1}^{H} \|\hat{s}_{t+h|t} - s_{t+h}\|_2^2. \tag{5.2}$$

Here, H is the time horizon of the simulation, and $\hat{s}_{t+h|t}$ is the estimate (from simulation) of the state at time $t+h$ given the state at time t. (See also Abbeel and Ng, 2005.)

The EM-algorithm presented in Chapter 4 can be expensive in our continuous

[2]Note that, as discussed previously, the one-step difference of body coordinate velocities is not the acceleration. To obtain actual accelerations, the velocity at time $t+1$ must be rotated into the body-frame at t before taking the difference.

state-action space setting. We therefore present a simple and fast algorithm for (approximately) minimizing the lagged error criterion. We begin by considering a linear model with update equation:

$$s_{t+1} - s_t = As_t + Bu_t, \tag{5.3}$$

where A, B are the parameters of the model. Minimizing the one-step prediction error would correspond to finding the parameters that minimize the expected squared difference between the left and right sides of Eqn. (5.3).

By summing the update equations for two consecutive time steps, we get that, for simulation to be exact over two time steps, the following needs to hold:

$$s_{t+2} - s_t = As_t + Bu_t + A\hat{s}_{t+1|t} + Bu_{t+1}. \tag{5.4}$$

Minimizing the expected squared difference between the left and right sides of Eqn. (5.4) would correspond to minimizing the two-step prediction error. More generally, by summing up the update equations for h consecutive timesteps and then minimizing the left and right sides' expected squared difference, we can minimize the h-step prediction error. Thus, it may seem that we can directly solve for the parameters that minimize the lagged criterion of Eqn. (5.2) by running least squares on the appropriate set of linear combinations of state update equations.

The difficulty with this procedure is that the intermediate states in the simulation—for example, $\hat{s}_{t+1|t}$ in Eqn. (5.4)—are also an implicit function of the parameters A and B. This is because $\hat{s}_{t+1|t}$ represents the result of a one-step simulation from s_t using our model. Taking into account the dependence of the intermediate states on the parameters makes the right side of Eqn. (5.4) non-linear in the parameters, and thus the optimization is non-convex. If, however, we make an approximation and neglect this dependence, then optimizing the objective can be done simply by solving a linear least squares problem.

This gives us the following algorithm. We will alternate between a simulation step that finds the necessary predicted intermediate states, and a least squares step that solves for the new parameters.

LEARN-LAGGED-LINEAR:

1. Use least squares to minimize the one-step squared prediction error criterion to obtain an initial model $A^{(0)}, B^{(0)}$. Set $i = 1$.
2. For all $t = 1, \ldots, T$, $h = 1, \ldots, H$, simulate in the current model to compute $\hat{s}_{t+h|t}$.
3. Solve the following least squares problem:

$$(\bar{A}, \bar{B}) = \arg\min_{A,B} \sum_{t=1}^{T-H} \sum_{h=1}^{H} \|(s_{t+h} - s_t) - (\sum_{\tau=0}^{h-1} A\hat{s}_{t+\tau|t} + Bu_{t+\tau})\|_2^2.$$

4. Set $A^{(i+1)} = (1-\alpha)A^{(i)} + \alpha\bar{A}$, $B^{(i+1)} = (1-\alpha)B^{(i)} + \alpha\bar{B}$.[3]
5. If $\|A^{(i+1)} - A^{(i)}\| + \|B^{(i+1)} - B^{(i)}\| \leq \epsilon$ exit. Otherwise go back to step 2.

Our helicopter acceleration prediction model is not of the simple form $s_{t+1} - s_t = As_t + Bu_t$ described above. However, a similar derivation still applies: The change in velocity over several time-steps corresponds to the sum of changes in velocity over several single time-steps. Thus by adding the one-step acceleration prediction equations as given in Section 5.4, we might expect to obtain equations corresponding to the acceleration over several time-steps. However, the acceleration equations at different time-steps are in different coordinate frames. Thus we first need to rotate the equations and then add them. In the algorithm described below, we rotate all accelerations into the world coordinate frame. The acceleration equations from Section 5.4 give us $(\ddot{x}, \ddot{y}, \ddot{z})_t^b = A_{\text{pos}} s_t + B_{\text{pos}} u_t$, and $(\ddot{\phi}, \ddot{\theta}, \ddot{\omega})_t^b = A_{\text{rot}} s_t + B_{\text{rot}} u_t$, where $A_{\text{pos}}, B_{\text{pos}}, A_{\text{rot}}, B_{\text{rot}}$ are (sparse) matrices that contain the parameters to be learned.[4] This gives us the LEARN-LAGGED-ACCELERATION algorithm, which is identical to LEARN-LAGGED-LINEAR except that step 3 now solves the following least squares problems:

$$(\bar{A}_{\text{pos}}, \bar{B}_{\text{pos}}) = \arg\min_{A,B} \sum_{t=1}^{T-H} \sum_{h=1}^{H} \| \sum_{\tau=0}^{h-1} \hat{R}^{b_{t+\tau} \to s} \left((\ddot{x}, \ddot{y}, \ddot{z})_{t+\tau}^b - (A\hat{s}_{t+\tau|t} + Bu_{t+\tau}) \right) \|_2^2$$

$$(\bar{A}_{\text{rot}}, \bar{B}_{\text{rot}}) = \arg\min_{A,B} \sum_{t=1}^{T-H} \sum_{h=1}^{H} \| \sum_{\tau=0}^{h-1} \hat{R}^{b_{t+\tau} \to s} \left((\ddot{\phi}, \ddot{\theta}, \ddot{\omega})_{t+\tau}^b - (A\hat{s}_{t+\tau|t} + Bu_{t+\tau}) \right) \|_2^2$$

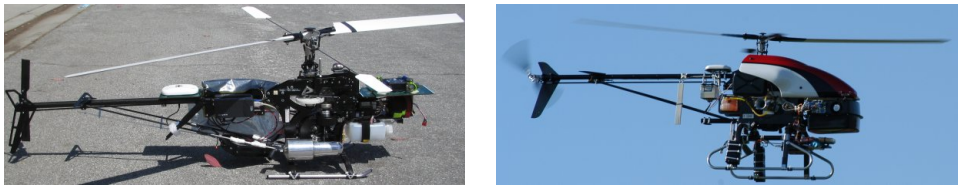
Here $\hat{R}^{b_t \to s}$ denotes the rotation matrix (estimated from simulation using the current model) from the body frame at time t to the world frame.

[3] This step of the algorithm uses a simple line search to choose the stepsize α.

[4] For simplicity of notation we omit the intercept parameters here, but they are easily incorporated, e.g., by having one additional input which is always equal to one.

5.6 Experiments

We performed experiments on two RC helicopters: an XCell Tempest and a Bergen Industrial Twin helicopter. (See Figure 5.1.) The XCell Tempest is a competition-class aerobatic helicopter (length 54", height 19"), is powered by a 0.91-size, two-stroke engine, and has an unloaded weight of 13 pounds. It carries two sensor units: a Novatel RT2 GPS receiver and a Microstrain 3DM-GX1 orientation sensor. The Microstrain package contains triaxial accelerometers, rate gyros, and magnetometers, which are used for inertial sensing. The larger Bergen Industrial Twin helicopter is powered by a twin cylinder 46cc, two-stroke engine, and has an unloaded weight of 18 lbs. It carries three sensor units: a Novatel RT2 GPS receiver, MicroStrain 3DM-G magnetometers, and an Inertial Science ISIS-IMU (triaxial accelerometers and rate gyros).



(a) (b)

Figure 5.1: The XCell Tempest (a) and the Bergen Industrial Twin (b) used in our experiments.

For each helicopter, we collected data from two separate flights. The XCell Tempest train and test flights were 800 and 540 seconds long, the Bergen Industrial Twin train and test flights were each 110 seconds long. A highly optimized Kalman filter integrates the sensor information and reports (at 100Hz) 12 numbers corresponding to the helicopter's state $(x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \omega, \dot{\phi}, \dot{\theta}, \dot{\omega})$. The data is then downsampled to 10Hz before learning. For each of the helicopters, we learned the following models:

1. Linear-One-Step: The linear model from Section 5.3 trained using linear regression to minimize the one-step prediction error.
2. Linear-CIFER: The linear model from Section 5.3 trained using CIFER.
3. Linear-Lagged: The linear model from Section 5.3 trained minimizing the lagged criterion.
4. Acceleration-One-Step: The acceleration prediction model from Section 5.4 trained using linear regression to minimize the one step prediction error.

5. Acceleration-Lagged: The acceleration prediction model from Section 5.4 trained minimizing the lagged criterion.

For Linear-Lagged and Acceleration-Lagged we used a horizon H of two seconds (20 simulation steps). The CPU times for training the different algorithms were: Less than one second for linear regression (algorithms 1 and 4 in the list above); one hour 20 minutes (XCell Tempest data) or 10 minutes (Bergen Industrial Twin data) for the lagged criteria (algorithms 3 and 5 above); about 5 minutes for CIFER. Our algorithm optimizing the lagged criterion appears to converge after at most 30 iterations. Since this algorithm is only approximate, we can then use coordinate descent search to further improve the lagged criterion.[5] This coordinate descent search took an additional four hours for the XCell Tempest data and an additional 30 minutes for the Bergen Industrial Twin data. We report results both with and without this coordinate descent search. Our results show that the algorithm presented in Section 5.5 works well for fast approximate optimization of the lagged criterion, but that locally greedy search (coordinate descent) may then improve it yet further.

For evaluation, the test data was split in consecutive non-overlapping two second windows. (This corresponds to 20 simulation steps, $s_0, \ldots, s_{20}$.) The models are used to predict the state sequence over the two second window, when started in the true state s_0. We report the average squared prediction error (difference between the simulated and true state) at each timestep $t = 1, \ldots, 20$ throughout the two second window. The orientation error is measured by the squared magnitude of the minimal rotation needed to align the simulated orientation with the true orientation. Velocity, position, angular rate and orientation errors are measured in m/s, m, rad/s and rad (squared) respectively. (See Figure 5.2.)

We see that Linear-Lagged consistently outperforms Linear-CIFER and Linear-One-Step. Similarly, for the acceleration prediction models, we have that Acceleration-Lagged consistently outperforms Acceleration-One-Step. These experiments support the case for training with the lagged criterion.

The best acceleration prediction model, Acceleration-Lagged, is significantly more

[5] We used coordinate descent on the criterion of Eqn. (5.2), but reweighted the errors on velocity, angular velocity, position and orientation to scale them to roughly the same order of magnitude.

accurate than any of the linear models presented in Section 3. This effect is mostly present in the XCell Tempest data, which contained data collected from many different parts of the state space (e.g., flying in a circle); in contrast, the Bergen Industrial Twin data was collected mostly near hovering (and thus the linearization assumptions were somewhat less poor there).

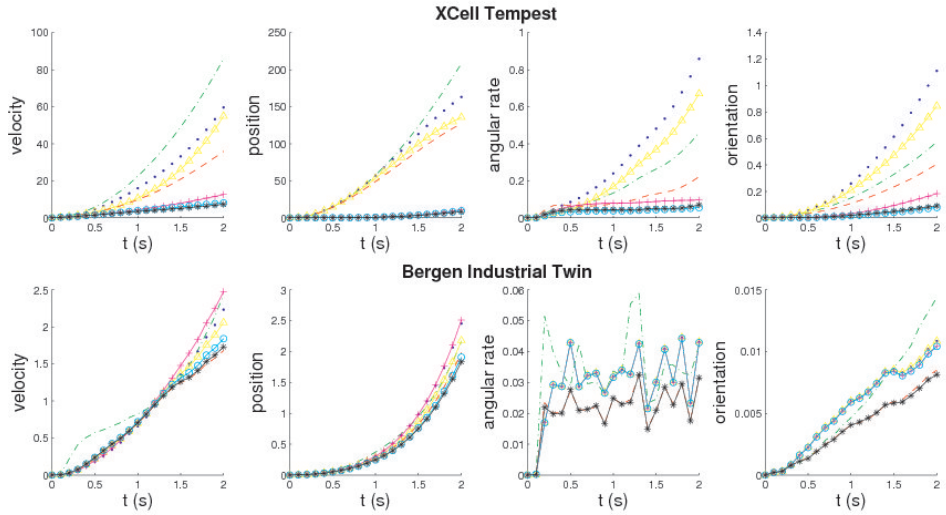


Figure 5.2: Average squared prediction errors throughout two-second simulations. Blue, dotted: Linear-One-Step. Green, dash-dotted: Linear-CIFER. Yellow, triangle: Linear-Lagged learned with fast, approximate algorithm from Section 5.5. Red, dashed: Linear-Lagged learned with fast, approximate algorithm from Section 5.5 followed by greedy coordinate descent search. Magenta, solid: Acceleration-One-Step. Cyan, circle: Acceleration-Lagged learned with fast, approximate algorithm from Section 5.5. Black,*: Acceleration-Lagged learned with fast, approximate algorithm from Section 5.5 followed by greedy coordinate descent search. The magenta, cyan and black lines (visually) coincide in the XCell position plots. The blue, yellow, magenta and cyan lines (visually) coincide in the Bergen angular rate and orientation plots. The red and black lines (visually) coincide in the Bergen angular rate plot. (Best viewed in color. See text for details.)

5.7 Discussion

We presented an acceleration based parameterization for learning vehicular dynamics. The model predicts accelerations, and then integrates to obtain state trajectories. We also described an efficient algorithm for approximately minimizing the lagged criterion, which measures the predictive accuracy of the algorithm over both short

and long time-scales. Our experiments illustrate the effectiveness of the acceleration-based parameterization and the lagged learning criterion.

One of the motivations behind the frequency based approaches is the capability of characterizing noise in the data—in particular, the frequency based approaches assume noise is more present in certain frequencies than others. They accordingly weight the fitting criterion to demand better fit for lower-noise frequencies. It could be interesting to try to incorporate such (or other) noise models into the lagged learning criterion.

Chapter 6

Using Inaccurate Models in Reinforcement Learning

Thus far in this dissertation we have considered the problems of (i) finding a good reward function for the task at hand, and (ii) finding a dynamics model that is sufficiently accurate for the task at hand. We have described apprenticeship learning algorithms, which leverage expert demonstrations to tackle both challenges. Once we have learned the appropriate reward function and dynamics model, we can use model-based reinforcement learning and optimal control algorithms to find a good control policy.

While a more accurate dynamics model will typically only improve the controllers found when using model-based reinforcement learning (RL) algorithms, a fully accurate dynamics model is not always necessary to find good controllers. E.g., when a human learns to drive a car, they can learn to drive the car reliably without necessarily building up a very detailed dynamics model. Instead, we can learn to drive a car based upon a crude model *and* a small number of real-life trials.

In this chapter, we present an algorithm that requires only an approximate model (of a possibly complex system), and only a small number of real-life trials. The key idea is that a real-life trial together with an approximate model can be sufficient to obtain reasonable policy gradient estimates.

6.1 Introduction

In model-based reinforcement learning (or optimal control), one first builds a model (or simulator) for the real system, and finds the control policy that is optimal in the model. Then this policy is deployed in the real system. Research in reinforcement learning and optimal control has generated efficient algorithms to find (near-)optimal policies for a large variety of models and reward functions. (See, e.g., Anderson and Moore (1989); Bertsekas (2001); Bertsekas and Tsitsiklis (1996); Sutton and Barto (1998).)

However, for many important control problems, particularly high-dimensional continuous-state tasks, it is extremely difficult to build an accurate model of the Markov decision process. When we learn a policy using an inaccurate model, we are often left with a policy that works well in simulation (i.e., the policy works well in the model), but not in real-life. In contrast to model-based policy search, there is the other extreme of searching for controllers only on the real system, without ever explicitly building a model. Although successfully applied to a few applications (e.g., Kohl and Stone, 2004), these model-free RL approaches tend to require huge, and often infeasibly large, numbers of real-life trials.

The large number of real-life trials required by model-free RL is in sharp contrast to, e.g., humans learning to perform a task. Consider, for example, a young adult learning to drive a car through a 90-degree turn, more specifically, learning the amount of steering required. On the first trial, she might take the turn wide. She will then adjust and take the next turn much less wide (or maybe even take it short). Typically, it will require only a few trials to learn to take the turn correctly.

The human driver clearly does not have a perfect model of the car. Neither does she need a large number of real-life trials. Instead, we believe that she combines a crude model of the car together with a small number of real-life trials to quickly learn to perform well.

In this chapter, we formalize this idea and develop an algorithm that exploits a crude model to quickly learn to perform well on real systems. Our theoretical results show that—assuming the model derivatives are good approximations of the

true derivatives, and assuming deterministic dynamics—our algorithm will return a policy that is (locally) near-optimal.

The key idea is to use a real-life trial to *evaluate* a policy, but then use the simulator (or model) to estimate the *derivative* of the evaluation with respect to the policy parameters (and suggest local improvements). For example, if in a car our current policy drives a maneuver too far to the left, then driving in real-life will be what tells us that we are driving too far to the left. However, 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. In particular, we do not need additional real-life trials of turning the steering wheel both clockwise and anti-clockwise in order to decide which direction to turn it. Therefore, even a crude model of the car allows us to significantly reduce the number of real-life trials needed compared to model-free algorithms that operate directly on the real system.

Compared to standard model-based algorithms, our approach has the advantage that it does not require a "correct" model of the Markov decision process. Despite the progress in learning algorithms for building better dynamical models, it remains an extremely difficult problem to model the detailed dynamics of many systems, such as helicopters, cars and some aircraft. The approach we present requires only a very crude model of the system.

Although an extension to stochastic systems might be possible, the algorithm and guarantees we present in this chapter only apply to systems that are (close to) deterministic. Despite this limitation, we believe our algorithm still has wide applicability.[1]

Throughout the chapter, we assume a continuous state space and a continuous action space.

Our theoretical results show that this algorithm achieves near-optimal performance in the real system, even when the model is only approximate. Empirical results also demonstrate that—when given only a crude model and a small number of

[1]Even though stochastic models are often used in reinforcement learning, the stochasticity of the models is often artificial and is to a large part used to capture *unmodeled dynamics*, rather than actual stochasticity.

real-life trials—our algorithm can obtain near-optimal performance in the real system.

Key assumptions for our results in this chapter are: (i) The system is (close to) deterministic, (ii) We have a bound on the error of the derivatives of the dynamics model.

The remainder of this chapter is organized as follows: Section 6.2 covers preliminaries. Section 6.3 describes our algorithm in full detail. Section 6.4 gives formal performance guarantees for our algorithm. Section 6.5 demonstrates the effectiveness of our algorithm when applied to flying a fixed-wing aircraft in a flight simulator and when applied to driving a real RC car.

The work described in this chapter has previously appeared in Abbeel, Quigley and Ng (2006).

6.2 Preliminaries

A non-stationary Markov decision process (MDP) can be described by a tuple (S, $\mathcal{A}$, T, H, s_0, R), where $S = \mathbb{R}^n$ is a set of states; $\mathcal{A} = \mathbb{R}^p$ is a set of actions/inputs; $T = \{P_t(\cdot|s,a)\}_{t,s,a}$ is a set of time-dependent state transition probabilities (here, $P_t(\cdot|s,a)$ is the state transition distribution upon taking action a in state s at time t); the horizon H is the number of time steps; s_0 is the initial state at time 0; and $R : S \mapsto \mathbb{R}$ is the reward function. We assume that R is bounded from above, i.e., for all states $s \in S$ we have $R(s) \leq R_{\max}$.

A policy $\pi = (\pi^{(0)}, \pi^{(1)}, \ldots, \pi^{(H)})$ is a set of mappings from the set of states S to the set of actions $\mathcal{A}$ for each time $t = 0, 1, \ldots, H$. The utility of a policy π in an MDP M is given by $U_M(\pi) = \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi, M]$. Here the expectation is over all possible state trajectories $s_0, s_1, \ldots, s_H$ in the MDP M. Throughout the chapter we consider policies parameterized by a parameter θ. We let $\pi_\theta^{(t)}(s)$ denote the action taken by the policy π_θ when in state s at time t.

This chapter focuses on MDPs that are (close to) deterministic. We let $\{f_t : S \times \mathcal{A} \to S\}_{t=0}^{H}$ denote the set of state transition functions. Here, $f_t(s,a)$ gives the expected state at time $t+1$ upon taking action a in state s at time t. For deterministic systems, the system dynamics (T) are specified completely by the time-dependent

state transition functions $\{f_t\}_{t=0}^{H}$.

We use $\|\cdot\|_2$ to denote the matrix 2-norm.[2]

6.3 Algorithm

Our algorithm takes as input an approximate MDP $\hat{M} = (S, \mathcal{A}, \hat{T}, H, s_0, R)$ and a local policy improvement algorithm A. The MDP's dynamics model $\hat{T}$ is a (possibly inaccurate) model of the true dynamics T of the true MDP $M = (S, \mathcal{A}, T, H, s_0, R)$. The local policy improvement algorithm A could be any method that iteratively makes local improvements upon the current policy. Two examples are policy gradient methods and differential dynamic programming.[3]

The algorithm proceeds as follows:

1. Set $i = 0$. Set the initial model estimate $\hat{T}^{(0)} = \hat{T}$.
2. Find the (locally) optimal policy $\pi_{\theta^{(0)}}$ for the MDP $\hat{M}^{(0)} = (S, \mathcal{A}, \hat{T}^{(0)}, H, s_0, R)$ by running the local policy improvement algorithm A.
3. Execute the current policy $\pi_{\theta^{(i)}}$ in the *real* MDP M and record the resulting state-action trajectory $s_0^{(i)}, a_0^{(i)}, s_1^{(i)}, a_1^{(i)}, \ldots, s_H^{(i)}, a_H^{(i)}$.
4. Construct the new model $\hat{T}^{(i+1)}$ by adding a (time-dependent) bias term to the original model $\hat{T}$. More specifically, set $\hat{f}_t^{(i+1)}(s, a) = \hat{f}_t(s, a) + s_{t+1}^{(i)} - \hat{f}_t(s_t^{(i)}, a_t^{(i)})$ for all times t.
5. Use the local policy improvement algorithm A in the MDP $\hat{M}^{(i+1)} = (S, \mathcal{A}, \hat{T}^{(i+1)}, H, s_0, R)$ to find a local policy improvement direction $d^{(i)}$ such that

[2] In the case the matrix is a vector, the matrix 2-norm is equal to the Euclidean norm. In general, the matrix 2-norm is equal to the maximum singular value of the matrix.

[3] Differential dynamic programming iteratively computes a time-varying linear model by linearizing the state transition functions around the current trajectory, and then uses the well-known exact solution to the resulting LQR (linear quadratic regulator) control problem as the step direction to locally improve the current policy (Jacobson & Mayne, 1970). In its full generality, differential dynamic programming also approximates the reward function by a concave quadratic function. The reward function approximation is irrelevant for this chapter: in our experiments the reward functions themselves are already quadratic.

$\hat{U}(\pi_{\theta^{(i)}+\alpha d^{(i)}}) \geq \hat{U}(\pi_{\theta^{(i)}})$ for some step-size $\alpha > 0$. Here $\hat{U}$ is the utility as evaluated in $\hat{M}^{(i+1)}$.[4]

6. Find the next (and improved) policy $\pi_{\theta^{(i+1)}}$, where $\theta^{(i+1)} = \theta^{(i)} + \alpha d^{(i)}$, by a line search over $\alpha > 0$. During the line search, evaluate the policies $\pi_{\theta^{(i)}+\alpha d^{(i)}}$in the real MDP M.

7. If the line-search did not find an improved policy, then return the current policy $\pi_{\theta^{(i)}}$ and exit. Otherwise, set $i = i + 1$ and go back to step 3.

The algorithm is initialized with the policy $\pi_{\theta^{(0)}}$, which is locally optimal for the initial (approximate) model of the dynamics. In subsequent iterations, the performance in the real system improves, and thus the resulting policy performs at least as well as the model-based policy $\pi_{\theta^{(0)}}$, and possibly much better.

In each iteration i, we add a time-dependent bias to the model: the term $s_{t+1}^{(i)} - \hat{f}_t(s_t^{(i)}, a_t^{(i)})$ in step 4 of the algorithm. For the resulting model $\hat{T}^{(i+1)}$ we have that $\hat{f}_t^{(i+1)}(s_t^{(i)}, a_t^{(i)}) = s_{t+1}^{(i)}$ for all times t. Hence, the resulting model $\hat{T}^{(i+1)}$ exactly predicts the real-life state sequence $s_0^{(i)}, s_1^{(i)}, \ldots, s_H^{(i)}$ obtained when executing the policy $\pi_{\theta^{(i)}}$. Thus, when computing the policy gradient (or, more generally, a local policy improvement direction) in step 5, our algorithm evaluates the derivatives along the correct state-action trajectory. In contrast, the pure model-based approach evaluates the derivatives along the state-action trajectory predicted by the original model $\hat{T}$, which—in the case of an imperfect model—would not correspond to the true state-action trajectory.

In contrast to our algorithm, one might also try a model-free algorithm in which real-life trajectories are used to estimate the policy gradient. However, doing so requires at least n real-life trajectories if there are n parameters in the policy.[5] In

[4]When the local policy improvement algorithm is policy gradient, we have $d^{(i)} = \nabla_\theta \hat{U}(\theta^{(i)})$, the policy gradient in the MDP $\hat{M}^{(i+1)}$ evaluated at the current policy $\pi_{\theta^{(i)}}$.

[5]If our policy π_θ is parameterized by $\theta \in \mathbb{R}^n$, we can estimate the derivative of the utility with respect to θ_i by evaluating the policy's performance $U(\pi_\theta)$ using parameters θ and again obtain its utility $U(\pi_{\theta+\epsilon e_i})$ using $\theta + \epsilon \cdot e_i$ (where e_i is the i-th basis vector). Then, we estimate the derivative as $\partial U(\pi_\theta)/\partial\theta_i \approx (U(\pi_{\theta+\epsilon e_i}) - U(\pi_\theta))/\epsilon$. In practice, this approach of using finite differences to numerically estimate derivatives not only suffers from the problem of requiring a large number of real-life trajectories, but is also very sensitive to even small amounts of noise in the estimates of the policy's utility.

contrast, our approach requires only a single real-life trajectory to obtain an estimate of the gradient. So long as the estimated gradient is within 90° of the true gradient, taking a small step in the direction of the estimated gradient will improve the policy. Consequently, our approach requires significantly fewer real-life trials than model-free algorithms to learn a good policy.

6.4 Theoretical results

Throughout this section *we assume that the real system is deterministic*, and we *assume that the local policy improvement algorithm is policy gradient.* In Section 6.4.1 we give an informal argument as to why our algorithm can be expected to outperform model-based algorithms that use the same model. Then in Section 6.4.2 we formalize this intuition and provide formal performance and convergence guarantees.

6.4.1 Gradient Approximation Error

The dependence of the state s_t on the policy π_θ plays a crucial role in our results. To express this dependence compactly, we define the function $h_t(s_0, \theta)$:

$$\begin{aligned} h_1(s_0, \theta) &= f_0(s_0, \pi_\theta(s_0)), \\ h_t(s_0, \theta) &= f_{t-1}\left(h_{t-1}(s_0, \theta)\right). \end{aligned} \tag{6.1}$$

Thus $h_t(s_0, \theta)$ is equal to the state at time t when using policy π_θ and starting in state s_0 at time 0. For an approximate model $\hat{T}$ we similarly define $\hat{h}_t$ in terms of the approximate transition functions $\hat{f}_t$ (instead of the true transition functions f_t).

Let $s_0, s_1, \ldots, s_H$ be the (real-life) state sequence obtained when executing the current policy π_θ. Then the true policy gradient is given by:

$$\nabla_\theta U(\theta) = \sum_{t=0}^{H} \nabla_s R(s_t) \left.\frac{dh_t}{d\theta}\right|_{s_0, s_1, \ldots, s_{t-1}}. \tag{6.2}$$

The derivatives appearing in $\frac{dh_t}{d\theta}$ can be computed by applying the chain rule to the

definition of h_t (Eqn. 6.1).[6] Expanding the chain rule results in a sum and product of derivatives of the transition functions $\{f_t\}_t$ and derivatives of the policy π_θ, which are evaluated at the states of the current trajectory $s_0, s_1, \ldots$.

Let $s_0, \hat{s}_1, \ldots, \hat{s}_H$ be the state sequence according to the model $\hat{T}$ when executing the policy π_θ. Then, according to the model $\hat{T}$, the policy gradient is given by:

$$\sum_{t=0}^{H} \nabla_s R(\hat{s}_t) \left. \frac{d\hat{h}_t}{d\theta} \right|_{\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{t-1}} . \tag{6.3}$$

Two sources of error make the policy gradient estimate (Eqn. 6.3) differ from the true policy gradient (Eqn. 6.2):

1. The *derivatives* appearing in $\frac{d\hat{h}_t}{d\theta}$ according to the model *are only an approximation of the true derivatives* appearing in $\frac{dh_t}{d\theta}$.
2. The *derivatives* appearing in $\nabla_s R$ and in $\frac{d\hat{h}_t}{d\theta}$ *are evaluated along the wrong trajectory*, namely the estimated (rather than the true) trajectory.

In our algorithm, by adding the time-dependent bias to the model (in step 4 of the algorithm), the resulting model perfectly predicts the true state sequence resulting from executing the current policy. This results in the following gradient estimate:

$$\nabla_\theta \hat{U}(\theta) = \sum_{t=0}^{H} \nabla_s R(s_t) \left. \frac{d\hat{h}_t}{d\theta} \right|_{s_0, s_1, \ldots, s_{t-1}} , \tag{6.4}$$

which evaluates the derivatives along the *true* trajectory. *Thus, by successively grounding its policy evaluations in real-life trials, our algorithm's policy gradient estimates are actually evaluated along true trajectories, eliminating the second source of error.*

We discussed the specific case of policy gradient. However, for any local policy improvement algorithm used in conjunction with our algorithm, we get the benefit of evaluating local changes along the *true* trajectory. For example, our experiments

[6] We use the phrasing "derivatives appearing in $\frac{dh_t}{d\theta}$" since for an n-dimensional state-space and a q-dimensional policy parameterization vector θ, the expression $\frac{dh_t}{d\theta}$ denotes the $n \times q$ Jacobian of derivatives of each state-coordinate with respect to each entry of θ.

successfully demonstrate the benefits of our algorithm when used in conjunction with differential dynamic programming.

6.4.2 Formal Results

The following theorem shows that (under certain assumptions) our algorithm converges to a region of local optimality (as defined more precisely in the theorem).

Theorem 12. *Let the local policy improvement algorithm be policy gradient. Let* $\epsilon > 0$ *be such that for all* $t = 0, 1, \ldots, H$ *we have* $\|\frac{df_t}{ds} - \frac{d\hat{f}_t}{ds}\|_2 \leq \epsilon$ *and* $\|\frac{df_t}{da} - \frac{d\hat{f}_t}{da}\|_2 \leq \epsilon$. *Let the line search in our algorithm be* η*-optimal (i.e., it finds a point that is within* η *of the optimal utility along the line). Let all first and second order derivatives of* $\{f_t(\cdot,\cdot)\}_{t=0}^H, \{\hat{f}_t(\cdot,\cdot)\}_{t=0}^H, \pi_\theta(\cdot), R(\cdot)$ *be bounded by a constant* C. *Let* $R \leq R_{\max}$. *Let* n, p, q *denote the dimensionality of the state space, action space and policy parameterization space respectively. Then our algorithm converges to a locally optimal region, in the following sense: there exist constants* K *and* M*—which are expressible as a function only of* n, p, q, *the constant* C, *and the MDP's horizon* H*—such that our algorithm converges to a region where the following holds:*

$$\|\nabla_\theta U(\theta)\|_2^2 \leq 2K^2\epsilon^2 + 2M\eta. \tag{6.5}$$

For the case of exact line search ($\eta = 0$*) we obtain:*

$$\|\nabla_\theta U(\theta)\|_2^2 \leq 2K^2\epsilon^2. \tag{6.6}$$

Proof (sketch). [7] The assumptions on the transition functions $\{f_t\}_{t=0}^H$, the approximate transition functions $\{\hat{f}_t\}_{t=0}^H$, the policy class π_θ and the reward function R imply that there exist constants K and M (expressible as a function of n, p, q, C and H only) such that:

$$\|\nabla_\theta U(\theta) - \nabla_\theta \hat{U}(\theta)\|_2 \leq K\epsilon, \tag{6.7}$$

$$\|\nabla_\theta^2 U(\theta)\|_2 \leq M. \tag{6.8}$$

[7] We give a complete proof in the Appendix.

Thus, our algorithm performs an iterative gradient ascent (with approximate gradient directions and approximate line search) to optimize the utility function U, which is bounded from above and which has bounded second order derivatives. Our proof proceeds by showing that this optimization procedure converges to a region where Eqn. (6.5) holds. □

Theorem 12 shows that—if certain boundedness and smoothness conditions hold for the true MDP and the model—our algorithm will converge to a region of small gradient. It also shows that, as the model's derivatives approach the true system's derivatives, the gradients in the (shrinking) convergence region approach zero (assuming exact line search).[8]

In contrast, no non-trivial bound holds for the model-based approach without additional assumptions.[9]

Theorem 12 assumes a deterministic MDP. Space constraints preclude an in-depth discussion, but we note that the ideas can be extended to systems with (very) limited stochasticity.[10]

[8]We note that, like exact gradient ascent, our algorithm could (pathologically) end up in a local minimum, rather than a local maximum.

[9]Consider the MDP with horizon H=1, state-space $S = \mathbb{R}$, transition function $f(s, a) = a$, and $R(s) = C \exp(-s^2)$. Now consider the model with transition function $\hat{f}(s, a) = a + b$, with $b \in \mathbb{R}$. Then, for C and b arbitrarily large, the model-based optimal policy will be arbitrarily worse than the true optimal policy. However, the assumptions of Theorem 12 are satisfied with $\epsilon = 0$, and our algorithm will find the optimal policy in a single iteration.

[10]The results can be extended to those stochastic systems for which we can bound the change in the utility and the change in the values of the derivatives (used in our algorithm) resulting from being evaluated along state trajectories from different random trials under the same policy. We note that the assumption concerns the trajectories obtained under a fixed policy in the policy class π_θ, rather than any policy or sequence of inputs. Feedback can often significantly reduce the diversity of state trajectories obtained, and thus help in satisfying the assumption. Still, the stochasticity needs to be very limited for the resulting bounds to be useful. A significant change in the algorithm (or at least in the analysis of the algorithm) seems necessary to obtain a useful bound for systems with significant stochasticity.

6.5 Experimental results

In all of our experiments, the local policy improvement algorithm is differential dynamic programming (DDP). We also used DDP to find the initial policy $\pi_{\theta^{(0)}}$.[11]

6.5.1 Flight simulation

We first tested our algorithm using a fixed-wing aircraft flight simulator. The flight simulator model contains 43 parameters corresponding to mass, inertia, drag coefficients, lift coefficients etc. We randomly generated "approximate models" by multiplying each parameter with a random number between 0.8 and 1.2 (one independent random number per parameter).[12] The model with the original (true) parameters is the "real system" in our algorithm. Our (closed-loop) controllers control all four standard fixed-wing aircraft inputs: throttle, ailerons, elevators and rudder. Our reward function quadratically penalizes for deviation from the desired trajectory (which in our case was a figure-8 at fixed altitude), for non-zero inputs, and for changes in inputs at consecutive time steps.[13]

Table 6.1 compares the performance of our algorithm and the model-based algorithm. Our algorithm significantly improves the utility (sum of accumulated rewards), namely by about 76%, within 5 iterations. Since typically only 1 iteration was required in the line searches, this typically corresponded to (only) 5 real-life trials. Figure 6.1 shows the result of one representative run: the desired trajectory (black, dotted), the trajectory with the controller from the (approximate) model (blue, solid), and the

[11] In iteration i, let $\pi_{\theta^{(i)}}$ be the current policy, and let $\pi_{\theta^{(i)\prime}}$ be the policy returned by DDP (applied to the MDP $\hat{M}^{(i+1)}$). Then, we have that $d^{(i)} = \theta^{(i)\prime} - \theta^{(i)}$ is the policy improvement direction. For the line search, we initialized $\alpha = 1$ (which corresponds to the policy $\pi_{\theta^{(i)\prime}}$), and then reduced α by 50% until an improved policy was found.

[12] We discarded approximate models for which the initial (model-based) controller was unstable in the real system. (This was $\pm 20\%$ of the approximate models.)

[13] Details of the setup: The fixed-wing flight simulator experiments were produced using the linearized 6-DOF model developed in Chapter 2 of Stevens and Lewis (2003). The parameterization is intended to model a small (1-kilogram, 1.5-meter wingspan) wing-body UAV that flies at low airspeeds (10 m/s). The simulation ran at 50Hz. The target trajectory was a figure-8 at fixed altitude with varying speed: slower in the turns, faster in the straight segments. The penalties on the inputs and on the changes in inputs make the controller more robust to model inaccuracies. The trajectory was 60 seconds long.

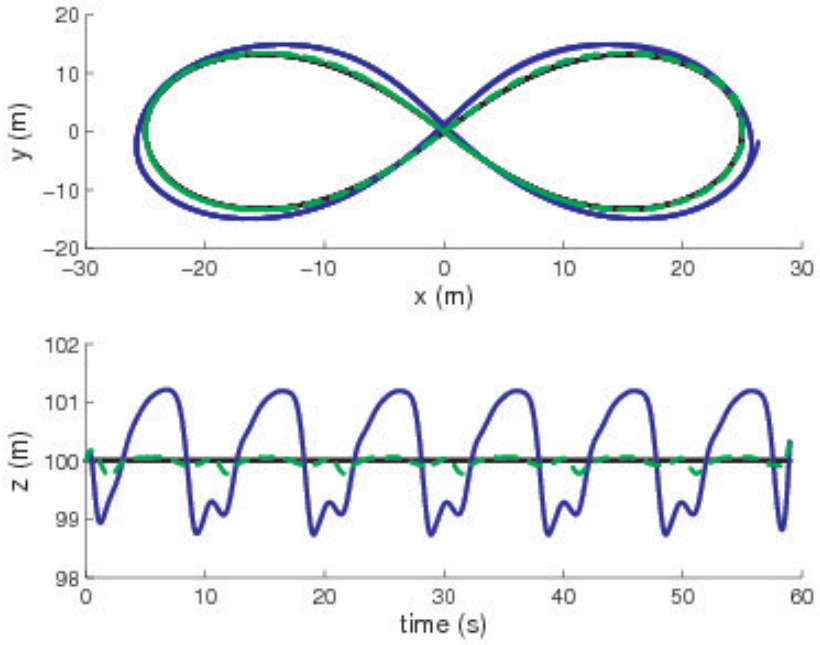


Figure 6.1: Results of our algorithm for the flight simulator. Black, dotted: desired trajectory; blue, solid: controller from model; green, dashed: controller after five iterations of our algorithm. (See text for details.)

Table 6.1: Utilities achieved in the flight simulator: mean and one standard error for the mean (10 runs).

Model-based	Our algorithm	Improvement
$-13,028$ (± 330)	$-3,000$ (± 100)	76% ($\pm 2\%$)

trajectory obtained after 5 iterations of our algorithm (green, dashed). The top plot shows the (x, y) coordinates (North, East), the bottom plot shows z (the altitude) over time. The initial controller (based on the perturbed model) performs reasonably well at following the (x, y) coordinates of the trajectory. However, it performs poorly at keeping altitude throughout the maneuver. Our algorithm significantly improves the performance, especially the altitude tracking. We note that it is fairly common for badly-tuned controllers to fail to accurately track altitude throughout transient parts of maneuvers. (Such as rolling left-right to stay in the figure-8 for our control task.)

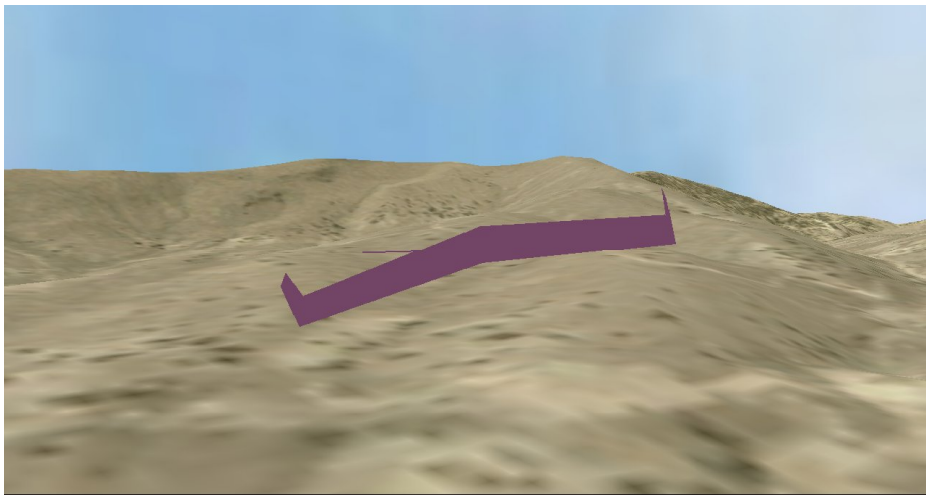


Figure 6.2: Screenshot of the flight-simulator.

A graphical simulation[14] of the trajectories flown by the model-based controller and by the controller found by our algorithm is available at the following url:

`http://www.cs.stanford.edu/~pabbeel/rl-videos.`

Figure 6.2 shows a screenshot of the graphical simulator.

6.5.2 RC car

In our second set of experiments, we used the RC car shown in Figure 6.3. A ceiling-mounted camera was used to obtain position estimates of the front and the rear of the car (marked with red and blue markers). This allowed us to track the car reliably within a rectangle of 3 by 2 meters. We then used an extended Kalman filter to track the car's state based on these observations in real-time.[15]

[14]The (OpenGL) graphical flight simulator was originally developed by Morgan Quigley, and is now maintained at http://sourceforge.net/projects/aviones.

[15]Experimental setup details: The RC car is an off-the-shelf XRAY M18, which has an electric drive motor and an electric steering servo motor. The car was fitted with lower gearing to allow it to run reliably at lower speeds (necessary to stay within the 3 by 2 meters area that can be tracked by the camera). It was fitted with a switching voltage regulator to prevent battery decay from affecting performance. The video stream captured by the ceiling mounted camera was digitized by a 2GHz Linux workstation and processed by OpenCV (Intel, 2001) to dewarp the camera image. Custom software was then used to locate the largest centroid of red and blue in the image, and to map those

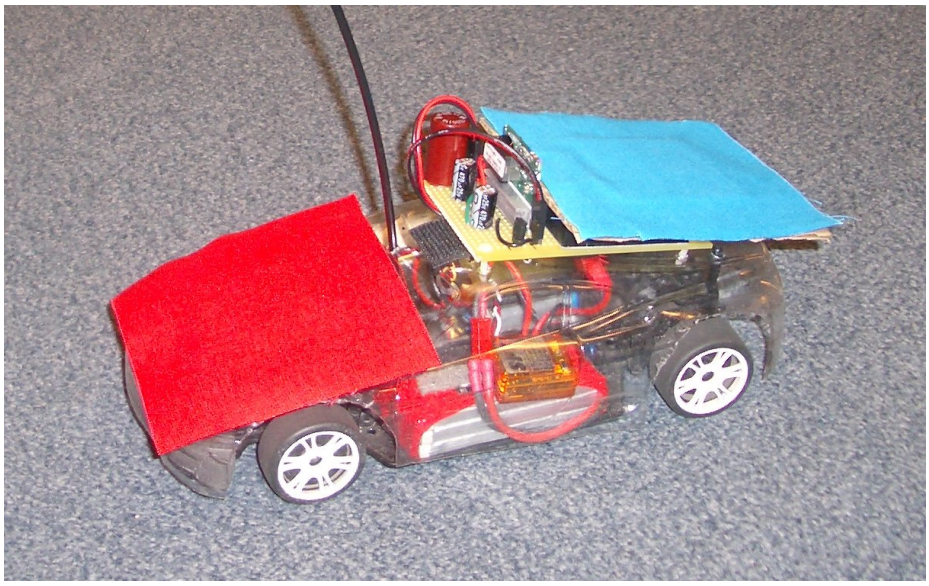


Figure 6.3: The RC car used in our experiments.

We modeled the RC car with six state-variables: the steering angle δ, the forward velocity of the front wheels v, the position coordinates x, y, the heading angle ψ and the heading rate $\dot{\psi}$. The inputs are the steering wheel servo input ($u_1 \in [-1, +1]$) and the drive motor throttle ($u_2 \in [-1, +1]$). At the speeds we consider, the slippage of front and rear wheels is negligible. Rigid-body kinematics then gives the heading rate $\dot{\psi}$ as a function of steering angle and velocity.[16] Data collected from the car showed that, when the throttle is decreased, the car slows down with fixed acceleration. However, when the throttle is increased, the response is similar to a first order linear

image locations back to their respective positions in the ground plane. This experimental setup was inspired in part by a similar one in Andrew W. Moore's Auton Lab at Carnegie-Mellon University.

[16]See, e.g., Gillespie (1992) pp. 196-201, for details on low-speed turning and details on car modeling in general.

model. This results in the following model:

$$\begin{aligned}
\delta_{\text{ss}} &= a_1 u_1 + b_1, \\
\dot{\delta} &= (\delta_{\text{ss}} - \delta)/\tau_\delta, \\
v_{\text{ss}} &= a_2 u_2 + a_3 |u_1| + b_2, \\
\dot{v} &= \mathbf{1}\{v_{\text{ss}} \geq v\}(v_{\text{ss}} - v)/\tau_v + \mathbf{1}\{v_{\text{ss}} < v\} a_4, \\
\dot{\psi} &= \frac{v \sin\delta}{L}.
\end{aligned}$$

Here $\mathbf{1}\{\cdot\}$ is the indicator function, which returns one if its argument is true and zero otherwise; δ_{ss} is the steady state steering angle for fixed input u_1; v_{ss} is the steady-state velocity for fixed inputs u_1 and u_2. We find the state variables x, y, ψ by numerical integration. The parameters $(a_1, b_1, a_2, b_2, a_3, a_4, \tau_\delta, \tau_v, L)$ were estimated from measurements from the car and data collected from manually driving the car.[17]

We considered three control problems: a turn with an open-loop controller, circles with a closed-loop controller, and a figure-8 with a closed-loop controller. During the open-loop turn, the desired velocity and turning radius change. Thus, the control task requires a sequence of inputs, not just a single throttle and steering input setting. The circle maneuver is non-trivial because of the velocity (1.4m/s) and the tight turning radius. Moreover, the carpet threading makes the RC car drift, and thus the circle cannot be driven with a fixed throttle and steering input. The figure-8 maneuver is the most challenging of the three maneuvers: the car is required to drive at varying speeds (fast in straight segments (1.4m/s), slower in turns (1.0m/s)) and is

[17] We measured the wheelbase of the car L. We measured the steering angle for various inputs u_1, and used linear regression to initialize the parameters a_1, b_1. We collected about 30 minutes of data (at 30Hz) from the RC car driving in circles with different radii and at various speeds, including the velocities and turning radii for the control tasks to come. We used linear regression to estimate the parameters a_2, b_2, a_3 from the data segments where steady-state velocity was reached. We used step-input responses to initialize the parameters τ_δ, τ_v, a_4. We fine-tuned the parameters $a_1, b_1, \tau_\delta, \tau_v, a_4$ by maximizing the likelihood of the data (using coordinate ascent). The model included that when the throttle input was very close to 0 (more specifically, below 0.18), the RC car's drive motor would stall, i.e., the drive motor acted as if its input was 0. The model included the latency from inputs to camera observation: it was estimated from step-input responses to be 0.1 seconds. Modeled latency does not affect, and is orthogonal to, our method. However, unmodeled latency contributes to model error and could thus decrease performance.

required to make sharp turns, which requires fast and accurate control at the desired velocity. In each case, our reward function penalizes quadratically for deviation from the target trajectory (which defines a target state for each time step) and penalizes quadratically for inputs and changes in the inputs. The penalty terms on the inputs make the controller more robust to model inaccuracies. (See, e.g., Anderson and Moore (1989) for more details.)[18]

For all three control problems our algorithm significantly outperforms the model-based controller after 5-10 iterations, with typically 1-4 real-life executions required per iteration (due to the line search). Figure 6.4 shows the improvements throughout the algorithm for the case of the turn with open loop control. We annotated the trajectories with the index of the corresponding iteration. We see that the algorithm consistently makes progress to finally follow the turn almost perfectly. This is in sharp contrast to the performance of the model-based controller. Figure 6.5 shows the initial (model-based) trajectory and the trajectory obtained with our algorithm for the cases of closed-loop control for following a circle and a figure-8. Our algorithm improved the performance (utility) by 97%, 88% and 63% for the turn, circles and figure-8, respectively. Videos of the real-life trials when learning to drive the different trajectories are available at the url given previously.[19]

The sub-optimal performance of the model-based policy approach indicates model inaccuracies. Given the amount of training data collected (covering the velocities and turning radii required for the control tasks later), we believe that the model inaccuracies are caused by our car modeling assumptions, rather than lack of data. This suggests a more complex car model might be needed for the model-based approach. Additional model inaccuracy results from the carpet threading. The asymmetric carpet threading causes the RC car to drift by up to 0.10m per circle when driving circles of 1m radius in open-loop (fixed throttle and steering inputs). The carpet threading

[18]Control setup details: differential dynamic programming generated a sequence of linear feedback controllers (for the closed-loop case) and a sequence of inputs (for the open-loop case), all at 20Hz. The extended Kalman filter updated its state at roughly 30Hz (the frame rate of the camera). Whenever the state estimate was updated, the controller for the time closest to the current time was used. All three target trajectories started from zero velocity.

[19]We repeated the experiments 3-4 times for each maneuver, with similar results. The reported results correspond to the movies we put available online.

also seems to affect the stalling speed of the drive motor.

To evaluate how deterministic the RC car setup is, we repeatedly ran a few open-loop input sequences. For the same input sequence, the end-points of the trajectories differed from a few centimeters up to 50cm (for 2 second runs). We believe the (apparent) non-determinism was caused mostly by the (unavoidably) slightly differing initial positions, which result in different interactions with the carpet throughout the trajectory. This effect is most significant when the car drives around the stalling speed of the drive motor.

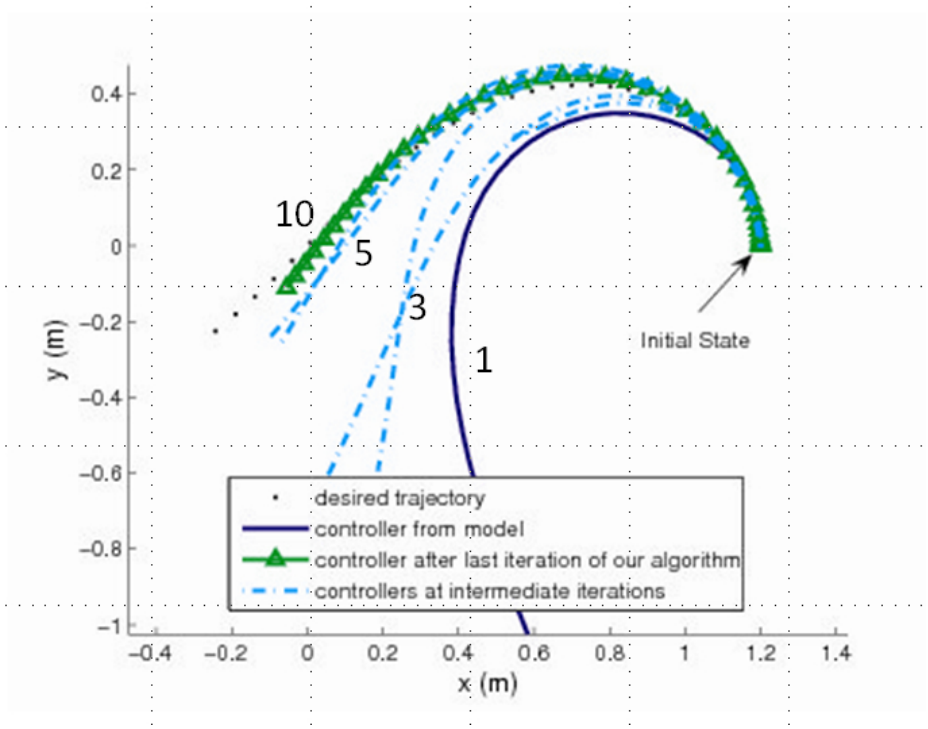


Figure 6.4: Results of our algorithm for the task of a turn with open-loop controller. (See text for details.)

6.6 Related work

Classical and (especially) robust control theory consider the design of controllers that work well for a large set of systems that are similar to the model that is used for

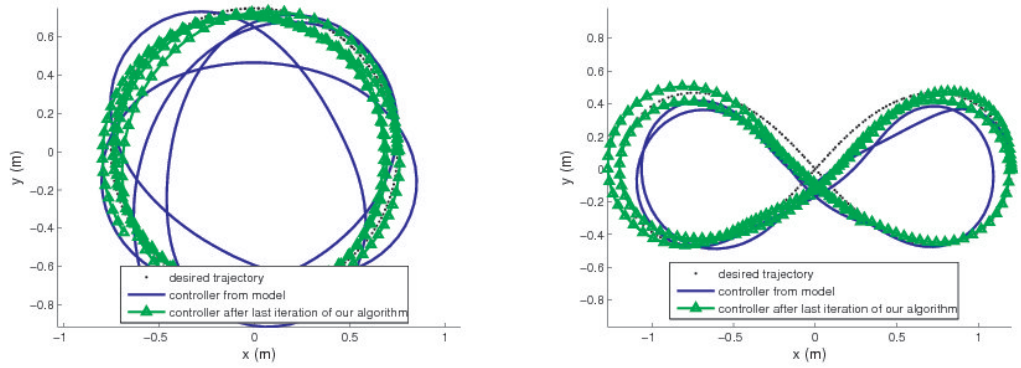


Figure 6.5: Results of our algorithm for the task of following a circle (a) and a figure-8 (b). (See text for details.)

designing the controller. The resulting controllers are less vulnerable to modeling errors, but do not achieve optimal performance in the true system. (See, e.g., Zhou et al. (1995); Dullerud and Paganini (2000) for more details on robust control. See, e.g., Bagnell et al. (2001); Nilim and El Ghaoui (2005); Morimoto and Atkeson (2002) for some examples of robust-control work within the RL community.) Also within the formalism of optimal control one can give up optimality and, instead, design controllers that are more robust to mismatches between model and real-life, namely by including additional terms into the reward function. For example, one penalizes high frequency inputs (since the simulator is typically less accurate at high frequencies), or one penalizes the integral of the error over time to avoid steady-state errors. (See, e.g., Anderson and Moore (1989) for the linear quadratic setting.)

Another approach is to use non-parametric learning algorithms to build a model. (Atkeson et al., 1997a) Although they might require more training data, non-parametric learning algorithms have the advantage of being less prone to modeling errors.

We note that Atkeson and Schaal (1997) and Morimoto and Doya (2001) also achieved successful robot control with a limited number of real-life trials. Our work significantly differs from both. Atkeson and Schaal (1997) started from a human demonstration of the (swing-up) task. Morimoto and Doya (2001) use a (hierarchical) Q-learning variant.

Iterative learning control (ILC) is the research area most closely related to our work, as our work could be seen as an instance of iterative learning control. ILC refers to the entire family of iterative control design approaches where the errors from past trials are used to compute the current inputs. In its most common form, ILC iteratively optimizes an open-loop (or feed-forward) input sequence. In contrast to our approach, most work in ILC does not use a model, rather, it just uses the past trials to compute the next iteration's controller. See, e.g., Moore (1998), for an overview and references to other work in iterative learning control.

6.7 Discussion

We presented an algorithm that uses a crude model and a small number of real-life trials to find a controller that works well in real-life. Our theoretical results show that—assuming the model derivatives are good approximations of the true derivatives, and assuming a deterministic setting—our algorithm will return a policy that is (locally) near-optimal. Our experiments (with a flight simulator and a real RC car) demonstrate that our algorithm can significantly outperform the model-based control approach, even when using only a few real-life trials on top of the crude model that is shared with the model-based approach.

While the fixed-task setting might seem restrictive at first, we believe many complicated tasks can be executed by putting together good controllers for fixed subtasks. E.g., a substantial fraction of car driving consists of starting, stopping, forward driving, lane changing, and a small number of turns.

An interesting future research direction is to extend our algorithm to the case of stochastic dynamics.

Chapter 7

Learning trajectories from multiple demonstrations

In Chapter 2 of this dissertation we described apprenticeship learning algorithms that leverage expert demonstrations to extract a definition of the task in the form of a reward function that trades of different features.

In this chapter we consider learning a different task representation from demonstration for another very common class of tasks in robotics: trajectory following. We apply our trajectory learning algorithm to learn aerobatic airshows for a helicopter.

7.1 Introduction

Many tasks in robotics can be described as a trajectory that the robot should follow. Unfortunately, specifying the desired trajectory and building an appropriate model for the robot dynamics along that trajectory are often non-trivial tasks. For example, when asked to describe the trajectory that a helicopter should follow to perform an aerobatic flip, one would have to specify a trajectory that (i) corresponds to the aerobatic flip task, and (ii) is consistent with the helicopter's dynamics. The latter requires (iii) an accurate helicopter dynamics model for all of the flight regimes encountered in the vicinity of the trajectory. These coupled tasks are non-trivial for systems with complex dynamics, such as helicopters. Failing to adequately address

these points leads to a significantly more difficult control problem.

In the apprenticeship learning setting, where an expert is available, rather than relying on a hand-engineered target trajectory, one can instead have the expert demonstrate the desired trajectory. The expert demonstration yields both a desired trajectory for the robot to follow, as well as data to build a dynamics model in the vicinity of this trajectory. Unfortunately, perfect demonstrations can be hard (if not impossible) to obtain. However, repeated expert demonstrations are often suboptimal in different ways, suggesting that a large number of suboptimal expert demonstrations could implicitly encode the ideal trajectory the suboptimal expert is trying to demonstrate.

In this chapter we propose an algorithm that approximately extracts this implicitly encoded optimal demonstration from multiple suboptimal expert demonstrations, and then builds a model of the dynamics in the vicinity of this trajectory suitable for high-performance control. In doing so, the algorithm learns a target trajectory and a model that allows the robot to not only mimic the behavior of the expert but even perform significantly better. (See, Chapter 8 for details on the helicopter flight experiments.)

Properly extracting the underlying ideal trajectory from a set of suboptimal trajectories requires a significantly more sophisticated approach than merely averaging the states observed at each time-step. A simple arithmetic average of the states would result in a trajectory that does not even obey the constraints of the dynamics model. Also, in practice, each of the demonstrations will occur at different rates so that attempting to combine states from the same time-step in each trajectory will not work properly.

We propose a generative model that describes the expert demonstrations as noisy observations of the unobserved, intended target trajectory, where each demonstration is possibly warped along the time axis. We present an EM algorithm—which uses a (extended) Kalman smoother and an efficient dynamic programming algorithm to perform the E-step—to both infer the unobserved, intended target trajectory and a time-alignment of all the demonstrations. The time-aligned demonstrations provide the appropriate data to learn good local models in the vicinity of the trajectory—such trajectory-specific local models tend to greatly improve control performance.

Our algorithm allows one to easily incorporate prior knowledge to further improve the quality of the learned trajectory. For example, for a helicopter performing in-place flips, it is known that the helicopter can be roughly centered around the same position over the entire sequence of flips. Our algorithm incorporates this prior knowledge, and successfully factors out the position drift in the expert demonstrations.

We apply our algorithm to learn trajectories and dynamics models for aerobatic flight with a remote controlled helicopter. Our experimental results show that (i) our algorithm successfully extracts a good trajectory from the multiple sub-optimal demonstrations, and (ii) the resulting flight performance significantly extends the prior state of the art in aerobatic helicopter flight (Abbeel et al., 2007; Gavrilets et al., 2002a). (See Chapter 8 for details on the helicopter flight experiments.) Most importantly, our resulting controllers are the first to perform as well, and often even better, than our expert pilot.

The remainder of this chapter is organized as follows: Section 7.2 presents our generative model for (multiple) suboptimal demonstrations; Section 7.3 describes our trajectory learning algorithm in detail; Section 7.4 describes our local model learning algorithm; Section 7.5 describes our helicopter platform and experimental results; Section 7.6 discusses related work.

The work described in this chapter has first appeared in Coates, Abbeel and Ng (2008).

7.2 Generative Model

7.2.1 Basic Generative Model

We are given M demonstration trajectories of length N^k, for $k = 0..M-1$. Each trajectory is a sequence of states, s_j^k, and control inputs, u_j^k, composed into a single state vector:

$$y_j^k = \begin{bmatrix} s_j^k \\ u_j^k \end{bmatrix}, \text{ for } j = 0..N^k - 1, k = 0..M-1.$$

Our goal is to estimate a "hidden" target trajectory of length T, denoted similarly:

$$z_t = \begin{bmatrix} s_t^\star \\ u_t^\star \end{bmatrix}, \text{ for } t = 0..T-1.$$

We use the following notation: $\mathbf{y} = \{y_j^k \mid j = 0..N^k - 1, k = 0..M-1\}$, $\mathbf{z} = \{z_t \mid t = 0..T-1\}$, and similarly for other indexed variables.

The generative model for the ideal trajectory is given by an initial state distribution $z_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ and an approximate model of the dynamics

$$z_{t+1} = f(z_t) + \omega_t^{(z)}, \qquad \omega_t^{(z)} \sim \mathcal{N}(0, \Sigma^{(z)}). \tag{7.1}$$

The dynamics model does not need to be particularly accurate—in our experiments, we use a single generic model learned from a large corpus of data that is not specific to the trajectory we want to perform. In our experiments (Section 7.5) we provide some concrete examples showing how accurately the generic model captures the true dynamics for our helicopter.[1]

Our generative model represents each demonstration as a set of independent "observations" of the hidden, ideal trajectory $\mathbf{z}$. Specifically, our model assumes

$$y_j^k = z_{\tau_j^k} + \omega_j^{(y)}, \qquad \omega_j^{(y)} \sim \mathcal{N}(0, \Sigma^{(y)}). \tag{7.2}$$

Here τ_j^k is the time index in the hidden trajectory to which the observation y_j^k is mapped. The noise term in the observation equation captures both inaccuracy in estimating the observed trajectories from sensor data, as well as errors in the maneuver that are the result of the human pilot's imperfect demonstration.[2]

[1] The state transition model also predicts the controls as a function of the previous state and controls. In our experiments we predict $u_{t+1}^\star$ as $u_t^\star$ plus Gaussian noise.

[2] Even though our observations, $\mathbf{y}$, are correlated over time with each other due to the dynamics governing the observed trajectory, our model assumes that the observations y_j^k are independent for all $j = 0..N^k - 1$ and $k = 0..M-1$.

The time indices τ_j^k are unobserved, and our model assumes the following distribution with parameters d_i^k:

$$\mathbb{P}(\tau_{j+1}^k | \tau_j^k) = \begin{cases} d_1^k & \text{if } \tau_{j+1}^k - \tau_j^k = 1 \\ d_2^k & \text{if } \tau_{j+1}^k - \tau_j^k = 2 \\ d_3^k & \text{if } \tau_{j+1}^k - \tau_j^k = 3 \\ 0 & \text{otherwise} \end{cases} \tag{7.3}$$

$$\tau_0^k \equiv 0. \tag{7.4}$$

To accommodate small, gradual shifts in time between the hidden and observed trajectories, our model assumes the observed trajectories are subsampled versions of the hidden trajectory. We found that having a hidden trajectory length equal to twice the average length of the demonstrations, i.e., $T = 2(\frac{1}{M}\sum_{k=1}^{M} N^k)$, gives sufficient resolution.

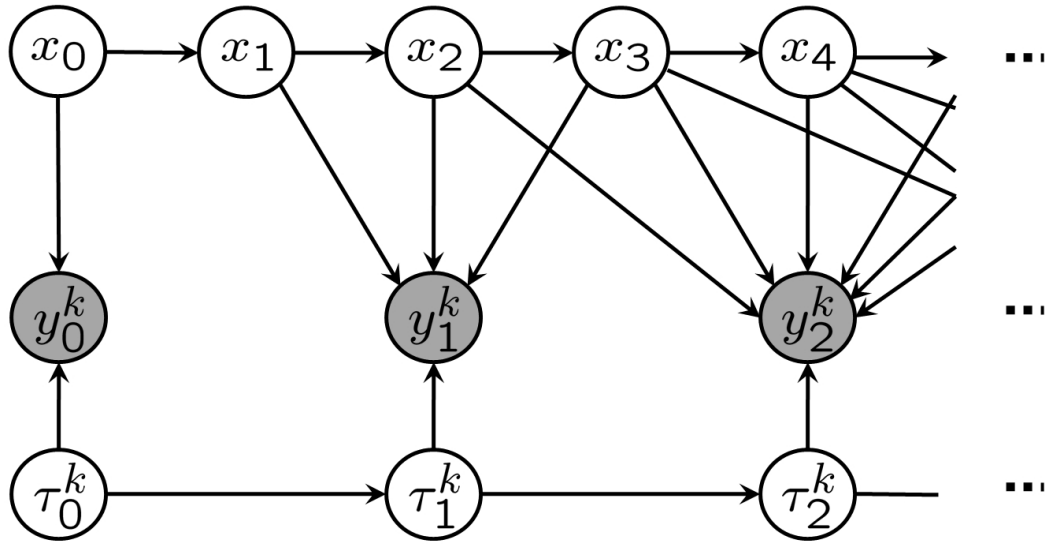


Figure 7.1: Graphical model representing our trajectory assumptions. (Shaded nodes are observed.)

Figure 7.1 depicts the graphical model corresponding to our basic generative

model. Note that each observation y_j^k depends on the hidden trajectory's state at time τ_j^k, which means that for τ_j^k unobserved, y_j^k depends on all states in the hidden trajectory that it could be associated with.

7.2.2 Extensions to the Generative Model

Thus far we have assumed that the expert demonstrations are misaligned copies of the ideal trajectory merely corrupted by Gaussian noise. Listgarten et al. have used this same basic generative model (for the case where $f(\cdot)$ is the identity function) to align speech signals and biological data (Listgarten, 2006; Listgarten et al., 2005). We now augment the basic model to account for other sources of error which are important for modeling and control.

Learning Local Model Parameters

For many systems, we can substantially improve our modeling accuracy by using a time-varying model $f_t(\cdot)$ that is specific to the vicinity of the intended trajectory at each time t. We express f_t as our "crude" model, f, augmented with a bias term[3], $\beta_t^\star$:

$$z_{t+1} = f_t(z_t) + \omega_t^{(z)} \equiv f(z_t) + \beta_t^\star + \omega_t^{(z)}.$$

To regularize our model, we assume that $\beta_t^\star$ changes only slowly over time. We have $\beta_{t+1}^\star \sim \mathcal{N}(\beta_t^\star, \Sigma^{(\beta)})$.

We incorporate the bias into our observation model by computing the observed bias $\beta_j^k = y_j^k - f(y_{j-1}^k)$ for each of the observed state transitions, and modeling this as a direct observation of the "true" model bias corrupted by Gaussian noise. The result of this modification is that the ideal trajectory must not only look similar to the demonstration trajectories, but it must also obey a dynamics model which includes those errors consistently observed in the demonstrations.

[3]Our generative model can incorporate richer local models. We discuss our choice of merely using biases in our generative trajectory model in more detail in Section 7.4.

Factoring out Demonstration Drift

It is often difficult, even for an expert pilot, during aerobatic maneuvers to keep the helicopter centered around a fixed position. The recorded position trajectory will often drift around unintentionally. Since these position errors are highly correlated, they are not explained well by the Gaussian noise term in our observation model.

To capture such slow drift in the demonstrated trajectories, we augment the latent trajectory's state with a "drift" vector δ_t^k for each time t and each demonstrated trajectory k. We model the drift as a zero-mean random walk with (relatively) small variance. The state observations are now noisy measurements of $z_t + \delta_t^k$ rather than merely z_t.

Incorporating Prior Knowledge

Even though it might be hard to specify the complete ideal trajectory in state space, we might still have prior knowledge about the trajectory. Hence, we introduce additional observations $\rho_t = \rho(z_t)$ corresponding to our prior knowledge about the ideal trajectory at time t. The function $\rho(z_t)$ computes some features of the hidden state z_t and our expert supplies the value ρ_t that this feature should take. For example, for the case of a helicopter performing an in-place flip, we use an observation that corresponds to our expert pilot's knowledge that the helicopter should stay at a fixed position while it is flipping. We assume that these observations may be corrupted by Gaussian noise, where the variance of the noise expresses our confidence in the accuracy of the expert's advice. In the case of the flip, the variance expresses our knowledge that it is, in fact, impossible to flip perfectly in-place and that the actual position of the helicopter may vary slightly from the position given by the expert.

Incorporating prior knowledge of this kind can greatly enhance the learned ideal trajectory. We give more detailed examples in Section 7.5.

Model Summary

In summary, we have the following generative model:

$$
\begin{aligned}
z_{t+1} &= f(z_t) + \beta_t^\star + \omega_t^{(z)}, && (7.5)\\
\beta_{t+1}^\star &= \beta_t^\star + \omega_t^{(\beta)}, && (7.6)\\
\delta_{t+1}^k &= \delta_t^k + \omega_t^{(\delta)}, && (7.7)\\
\rho_t &= \rho(z_t) + \omega_t^{(\rho)}, && (7.8)\\
y_j^k &= z_{\tau_j^k} + \delta_j^k + \omega_j^{(y)}, && (7.9)\\
\tau_j^k &\sim \mathbb{P}(\tau_{j+1}^k | \tau_j^k) && (7.10)
\end{aligned}
$$

Here $\omega_t^{(z)}, \omega_t^{(\beta)}, \omega_t^{(\delta)}, \omega_t^{(\rho)}, \omega_j^{(y)}$ are zero mean Gaussian random variables with respective covariance matrices $\Sigma^{(z)}, \Sigma^{(\beta)}, \Sigma^{(\delta)}, \Sigma^{(\rho)}, \Sigma^{(y)}$. The transition probabilities for τ_j^k are defined by Eqs. (7.3, 7.4) with parameters d_1^k, d_2^k, d_3^k (collectively denoted $\mathbf{d}$).

7.3 Trajectory Learning Algorithm

Our learning algorithm automatically finds the time-alignment indexes $\boldsymbol{\tau}$, the time-index transition probabilities $\mathbf{d}$, and the covariance matrices $\Sigma^{(\cdot)}$ by (approximately) maximizing the joint likelihood of the observed trajectories $\mathbf{y}$ and the observed prior knowledge about the ideal trajectory $\boldsymbol{\rho}$, while marginalizing out over the unobserved, intended trajectory $\mathbf{z}$. Concretely, our algorithm (approximately) solves

$$\max_{\boldsymbol{\tau}, \Sigma^{(\cdot)}, \mathbf{d}} \log \mathbb{P}(\mathbf{y}, \boldsymbol{\rho}, \boldsymbol{\tau} \ ; \ \Sigma^{(\cdot)}, \mathbf{d}). \tag{7.11}$$

Then, once our algorithm has found $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$, it finds the most likely hidden trajectory, namely the trajectory $\mathbf{z}$ that maximizes the joint likelihood of the observed trajectories $\mathbf{y}$ and the observed prior knowledge about the ideal trajectory $\boldsymbol{\rho}$ for the learned parameters $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$.[4]

[4]Note maximizing over the hidden trajectory and the covariance parameters simultaneously introduces undesirable local maxima: the likelihood score would be highest (namely infinity) for a hidden trajectory with a sequence of states exactly corresponding to the (crude) dynamics model

The joint optimization in Eq. (7.11) is difficult because (as can be seen in Figure 7.1) the lack of knowledge of the time-alignment index variables $\boldsymbol{\tau}$ introduces a very large set of dependencies between all the variables. However, when $\boldsymbol{\tau}$ is known, the optimization problem in Eq. (7.11) greatly simplifies thanks to context specific independencies (Boutilier et al., 1996). When $\boldsymbol{\tau}$ is fixed, we obtain a model such as the one shown in Figure 7.2. In this model we can directly estimate the multinomial parameters $\mathbf{d}$ in closed form; and we have a standard HMM parameter learning problem for the covariances $\Sigma^{(\cdot)}$, which can be solved using the EM algorithm (Dempster et al., 1977)—often referred to as Baum-Welch in the context of HMMs. Concretely, for our setting, the EM algorithm's E-step computes the pairwise marginals over sequential hidden state variables by running a (extended) Kalman smoother; the M-step then uses these marginals to update the covariances $\Sigma^{(\cdot)}$.

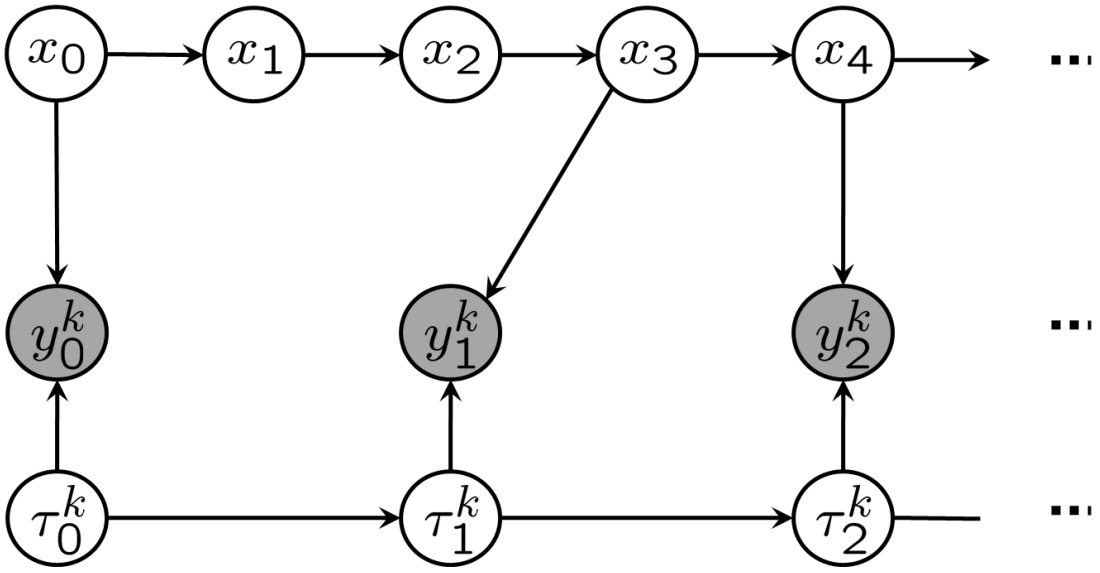


Figure 7.2: Example of graphical model when $\boldsymbol{\tau}$ is known. (Shaded nodes are observed.)

To also optimize over the time-indexing variables $\boldsymbol{\tau}$, we propose an alternating optimization procedure. For fixed $\Sigma^{(\cdot)}$ and $\mathbf{d}$, and for fixed $\mathbf{z}$, we can find the optimal

$f(\cdot)$ and state-transition covariance matrices equal to all-zeros as long as the observation covariances are non-zero. Hence we marginalize out the hidden trajectory to find $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$.

time-indexing variables $\boldsymbol{\tau}$ using dynamic programming over the time-index assignments for each demonstration independently. The dynamic programming algorithm to find $\boldsymbol{\tau}$ is known in the speech recognition literature as dynamic time warping (Sakoe & Chiba, 1978) and in the biological sequence alignment literature as the Needleman-Wunsch algorithm (Needleman & Wunsch, 1970). The fixed $\mathbf{z}$ we use, is the one that maximizes the likelihood of the observations for the current setting of parameters $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$.[5]

In practice, rather than alternating between complete optimizations over $\Sigma^{(\cdot)}, \mathbf{d}$ and $\boldsymbol{\tau}$, we only partially optimize over $\Sigma^{(\cdot)}$, running only one iteration of the EM algorithm.

We provide the complete details of our algorithm in Appendix E.

7.4 Local Model Learning

For complex dynamical systems, the state z_t used in the dynamics model often does not correspond to the "complete state" of the system, since the latter could involve large numbers of previous states or unobserved variables that make modeling difficult.[6] However, when we only seek to model the system dynamics along a specific trajectory, knowledge of both z_t and how far we are along that trajectory is often sufficient to accurately predict the next state z_{t+1}.

Once the alignments between the demonstrations are computed by our trajectory learning algorithm, we can use the time aligned demonstration data to learn a sequence of trajectory-specific models. The time indices of the aligned demonstrations now accurately associate the demonstration data points with locations along the learned trajectory, allowing us to build models for the state at time t using the

[5] Fixing $\mathbf{z}$ means the dynamic time warping step only approximately optimizes the original objective. Unfortunately, without fixing $\mathbf{z}$, the independencies required to obtain an efficient dynamic programming algorithm do not hold. In practice we find our approximation works very well.

[6] This is particularly true for helicopters. Whereas the state of the helicopter is very crudely captured by the 12D rigid-body state representation we use for our controllers, the "true" physical state of the system includes, among others, the airflow around the helicopter, the rotor head speed, and the actuator dynamics.

appropriate corresponding data from the demonstration trajectories.[7]

To construct an accurate nonlinear model to predict z_{t+1} from z_t, using the aligned data, one could use locally weighted linear regression (Atkeson et al., 1997b), where a linear model is learned based on a weighted dataset. Data points from our aligned demonstrations that are nearer to the current time index along the trajectory, t, and nearer the current state, z_t, would be weighted more highly than data far away. While this allows us to build a more accurate model from our time-aligned data, the weighted regression must be done online, since the weights depend on the current state, z_t. For performance reasons[8] this may often be impractical. Thus, we weight data only based on the time index, and learn a parametric model in the remaining variables (which, in our experiments, has the same form as the global "crude" model, $f(\cdot)$). Concretely, when estimating the model for the dynamics at time t, we weight a data point at time t' by:[9]

$$W(t') = \exp\left(-\frac{(t-t')^2}{\sigma^2}\right),$$

where σ is a bandwidth parameter. Typical values for σ are between one and two seconds in our experiments. Since the weights for the data points now only depend on the time index, we can precompute all models $f_t(\cdot)$ along the entire trajectory. The ability to precompute the models is a feature crucial to our control algorithm, which relies heavily on fast simulation.

[7]We could learn the richer local model within the trajectory alignment algorithm, updating the dynamics model during the M-step. We chose not to do so since these models are more computationally expensive to estimate. The richer models have minimal influence on the alignment because the biases capture the average model error—the richer models capture the derivatives around it. Given the limited influence on the alignment, we chose to save computational time and only estimate the richer models after alignment.

[8]During real-time control execution, our model is queried roughly 52000 times per second. Even with KD-tree or cover-tree data structures a full locally weighted model would be much too slow.

[9]In practice, the data points along a short segment of the trajectory lie in a low-dimensional subspace of the state space. This sometimes leads to an ill-conditioned parameter estimation problem. To mitigate this problem, we regularize our models toward the "crude" model $f(\cdot)$.

7.5 Experimental Results

7.5.1 Experimental Setup

To test our algorithm, we had our expert helicopter pilot fly our XCell Tempest helicopter (Figure 8.1), which can perform professional, competition-level maneuvers. We instrumented the helicopter with a Microstrain 3DM-GX1 orientation sensor. A ground-based camera system measures the helicopter's position. A Kalman filter uses these measurements to track the helicopter's position, velocity, orientation and angular rate. (See Chapter 8 Section 8.2 for more details on our helicopter platform.)

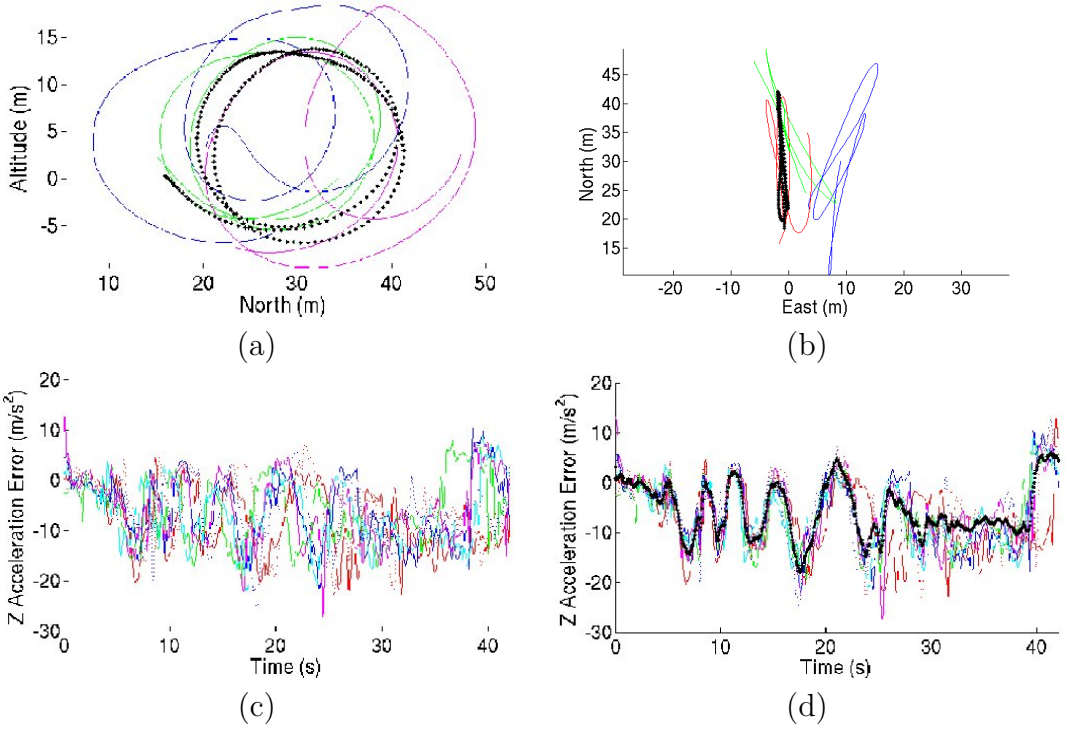


Figure 7.3: Colored lines: demonstrations. Black dotted line: trajectory inferred by our algorithm. (See text for details.)

We collected multiple demonstrations from our expert for a variety of aerobatic trajectories: continuous in-place flips and rolls, a continuous tail-down "tic toc," and an airshow, which consists of the following maneuvers in rapid sequence: split-S, snap roll, stall-turn, loop, loop with pirouette, stall-turn with pirouette, "hurricane" (fast

backward funnel), knife-edge, flips and rolls, tic-toc and inverted hover.

The (crude) helicopter dynamics $f(\cdot)$ is constructed using the method of Abbeel et al. (2006a).[10] The helicopter dynamics model predicts linear and angular accelerations as a function of current state and inputs. The next state is then obtained by integrating forward in time using the standard rigid-body equations.

In the trajectory learning algorithm, we have bias terms $\beta_t^\star$ for each of the predicted accelerations. We use the state-drift variables, δ_t^k, for position only.

For the flips, rolls, and tic-tocs we incorporated our prior knowledge that the helicopter should stay in place. We added a measurement of the form:

$$0 = p(z_t) + \omega^{(\rho_0)}, \quad \omega^{(\rho_0)} \sim \mathcal{N}(0, \Sigma^{(\rho_0)})$$

where $p(\cdot)$ is a function that returns the position coordinates of z_t, and $\Sigma^{(\rho_0)}$ is a diagonal covariance matrix. This measurement—which is a direct observation of the pilot's intended trajectory—is similar to advice given to a novice human pilot to describe the desired maneuver: A good flip, roll, or tic-toc trajectory stays close to the same position.

We also used additional advice in the airshow to indicate that the vertical loops, stall-turns and split-S should all lie in a single vertical plane; that the hurricanes should lie in a horizontal plane and that a good knife-edge stays in a vertical plane. These measurements take the form:

$$c = N^\top p(z_t) + \omega^{(\rho_1)}, \quad \omega^{(\rho_1)} \sim \mathcal{N}(0, \Sigma^{(\rho_1)})$$

where, again, $p(z_t)$ returns the position coordinates of z_t. N is a vector normal to the plane of the maneuver, c is a constant, and $\Sigma^{(\rho_1)}$ is a diagonal covariance matrix.

[10] The model of Abbeel et al. (2006a) naturally generalizes to any orientation of the helicopter regardless of the flight regime from which data is collected. Hence, even without collecting data from aerobatic flight, we can reasonably attempt to use such a model for aerobatic flying, though we expect it to be relatively inaccurate.

7.5.2 Trajectory Learning Results

Figure 7.3(a) shows the horizontal and vertical position of the helicopter during the two loops flown during the airshow. The colored lines show the expert pilot's demonstrations. The black dotted line shows the inferred ideal path produced by our algorithm. The loops are more rounded and more consistent in the inferred ideal path. We did not incorporate any prior knowledge to this extent. Figure 7.3(b) shows a top-down view of the same demonstrations and inferred trajectory. The prior successfully encouraged the inferred trajectory to lie in a vertical plane, while obeying the system dynamics.

Figure 7.3(c) shows one of the bias terms, namely the model prediction errors for the Z-axis acceleration of the helicopter computed from the demonstrations, before time-alignment. Figure 7.3(d) shows the result after alignment (in color) as well as the inferred acceleration error (black dotted). We see that the unaligned bias measurements allude to errors approximately in the -1G to -2G range for the first 40 seconds of the airshow (a period that involves high-G maneuvering that is not predicted accurately by the "crude" model). However, only the aligned biases precisely show the magnitudes and locations of these errors along the trajectory. The alignment allows us to build our ideal trajectory based upon a much more accurate model that is tailored to match the dynamics observed in the demonstrations.

Results for other maneuvers and state variables are similar. At the URL below we provide movies that simultaneously replay the different demonstrations, before alignment and after alignment. The movies visualize the alignment results in many state dimensions simultaneously.

`http://heli.stanford.edu`

7.6 Related Work

Although no prior works span our entire setting of learning for control from multiple demonstrations, there are separate pieces of work that relate to various components of our approach.

Atkeson and Schaal (1997) use multiple demonstrations to learn a model for a robot arm, and then find an optimal controller in their simulator, initializing their optimal control algorithm with one of the demonstrations.

The work of Calinon et al. (2007) considered learning trajectories and constraints from demonstrations for robotic tasks. There, they do not consider the system's dynamics or provide a clear mechanism for the inclusion of prior knowledge. Our formulation presents a principled, joint optimization which takes into account the multiple demonstrations, as well as the (complex) system dynamics and prior knowledge. While Calinon et al. (2007) also use some form of dynamic time warping, they do not try to optimize a joint objective capturing both the system dynamics and time-warping.

Among others, An et al. (1988) and, more recently, Abbeel et al. (2006b) have exploited the idea of trajectory-indexed model learning for control. However, contrary to our setting, their algorithms do not time align nor coherently integrate data from multiple trajectories.

While the work by Listgarten et al. (Listgarten, 2006; Listgarten et al., 2005) does not consider robotic control and model learning, they also consider the problem of multiple continuous time series alignment with a hidden time series.

The work described in this chapter also has strong connections with recent work on inverse reinforcement learning, which extracts a reward function (rather than a trajectory) from the expert demonstrations. We refer the reader to Chapter 2 for more details.

7.7 Discussion

We presented an algorithm that takes advantage of multiple suboptimal trajectory demonstrations to (i) extract (an estimate of) the ideal demonstration, (ii) learn a local model along this trajectory. Our algorithm is generally applicable for learning trajectories and dynamics models along trajectories from multiple demonstrations.

Chapter 8

An Application to Aerobatic Helicopter Flight

In this chapter we describe the application of the apprenticeship learning methods presented in preceding chapters to the problem of autonomous helicopter flight. They have enabled us to significantly extend the state of the art in autonomous helicopter aerobatics. Our results include the first autonomous in-place flips, in-place rolls, tic-tocs, loops and hurricane, and a complete airshow, which requires autonomous transitions between these and various other maneuvers. Our controllers perform as well, and often even better, than our expert pilot.

Videos of our autonomous helicopter flight results are available at:

`http://heli.stanford.edu`.

8.1 Introduction

Autonomous helicopter flight is a challenging control problem with high-dimensional, asymmetric, noisy, nonlinear, non-minimum phase dynamics. Helicopters are widely regarded to be significantly harder to control than fixed-wing aircraft. (See, e.g., Leishman, 2000; Seddon, 1990.) At the same time, helicopters provide unique capabilities, such as in-place hover and low-speed flight, important for many applications. The

control of autonomous helicopters thus provides a challenging and important testbed for learning and control algorithms.

In the "upright flight regime" there has recently been considerable progress in autonomous helicopter flight. For example, Bagnell and Schneider (2001) achieved sustained autonomous hover. Both La Civita et al. (2006) and Ng et al. (2004b) achieved sustained autonomous hover and accurate flight in regimes where the helicopter's orientation is fairly close to upright. Roberts et al. (2003) and Saripalli et al. (2003) achieved vision based autonomous hover and landing.

By contrast, autonomous flight achievements in other flight regimes have been very limited. Gavrilets et al. (2004) achieved a split-S, a stall turn and a roll in forward flight. Ng et al. (2004a) achieved sustained autonomous inverted hover.

The results presented in this chapter significantly expand the limited set of successfully completed aerobatic maneuvers. We present the first successful autonomous completion of the following maneuvers: continuous in-place flips and rolls, a continuous tail-down "tic-toc," loops, loops with pirouettes, stall-turns with pirouette, "hurricane" (fast backward funnel), knife-edge, immelmann, slapper, sideways tic-toc, traveling flips, inverted tail-slide. Not only are we first to autonomously complete such maneuvers, our controllers are also able to continuously repeat the maneuvers without any pauses in between. Thus the controller has to provide continuous feedback *during* the maneuvers, and cannot, for example, use a period of hovering to correct errors from the first execution of the maneuver before performing the maneuver a second time. In fact, we also have our helicopter fly complete aerobatic airshows, during which our helicopter executes a wide variety of aerobatic maneuvers in rapid sequence. (See Section 8.6 for details.)

The remainder of this chapter is organized as follows: Section 8.2 describes our helicopter platform. Section 8.3 describes our state estimation: we use an (extended) Kalman filter, which gives us state estimates by fusing our sensor data. Section 8.4 describes our approach, going from expert demonstrations to fully autonomous helicopter aerobatics. Our algorithm assumes access to a reinforcement learning (or optimal control) algorithm which can solve for the optimal policy when given a dynamics model and a reward function. In Section 8.5 we describe the details of our

controller, which is based upon receding horizon differential dynamic programming. Section 8.6 describes our autonomous flight results. Section 8.7 concludes the chapter.

Movies of our autonomous helicopter flights are available at the following url, also mentioned in the first paragraph of this chapter:

`http://heli.stanford.edu`.

The work presented in this chapter has previously appeared in Abbeel, Coates, Quigley and Ng (2007); Coates, Abbeel and Ng (2008).

8.2 Helicopter platform

We have used a variety of helicopter models and configurations throughout various experiments. The helicopters themselves have been off-the-shelf RC helicopter kits, which we then assemble the standard way. The two helicopter platforms we have flown autonomously are: (i) the 90-size XCell Tempest (length 135cm, height 51cm, weight 5.10kg, main rotor blade length 720mm), featured in Figure 8.1, and (ii) the 90-size Synergy N9 (length 138cm, height 40cm, weight 4.85kg, main rotor blade length 720mm), featured in Figure 8.2. Our instrumentation (inertial unit, radio, battery, packaging/mounting) adds 0.44kg. Both platforms are competition-class aerobatic helicopters powered by two-stroke engines. Between the two, the Synergy N9 platform offers more in terms of extreme aerobatics: the center of gravity (CG) is higher (closer to the rotor), the fuel tank is under the CG (doesn't change during flight), the canopy is more aerodynamic (ability to carry inertia), the engine has better cooling (hence can sustain a higher power output).

We instrument our helicopters with a Microstrain 3DM-GX1 orientation sensor. The Microstrain package contains triaxial accelerometers, rate gyros, and magnetometers at 333Hz.

For position sensing we have used various solutions. Our autonomous flight approach does not change with the type of position sensing we use, but it does assume we regularly get position information.

Figure 8.1: XCell Tempest.

Figure 8.2: Synergy N9.

When we only fly in the close-to-upright flight regime, we have instrumented our helicopters with the off-the-shelf Novatel RT2 GPS receiver, which uses carrier-phase differential GPS to provide real-time position estimates at 10Hz with approximately 2cm accuracy as long as its antenna is pointing at the sky, i.e., as long as its antenna gets sufficient satellite coverage.

When we fly helicopter aerobatics, we use a ground-based vision system. We have two (or more) cameras in known position and orientation on the ground. We track the helicopter in the camera images and obtain position estimates for the helicopter through triangulation. We use Point Grey Research DragonFly2 cameras. We get position estimates at 5Hz that are about 25cm accurate at about 40m distance from the cameras. The accuracy increases a bit closer in, and decreases farther out.

We use Maxstream's (now Digi) XBee Pro chips, which send the inertial (and possibly GPS) data from the helicopter to a ground-based PC. The ground-based PC also receives the vision-based position estimates when applicable. The ground-based PC then fuses the inertial and position sensing information into an (extended) Kalman filter to obtain state estimates. Our ground-based PC is a Dell Precision 490n Workstation, with a dual 2.66GHz Intel Xeon, 2GB RAM, running Gentoo Linux (2.6.19 with low-latency kernel configuration).

Our controller generates control signals for each of the helicopter's control channels based upon the control task and the current state estimate. The governor automatically controls the throttle channel such as to keep the engine rpm at the desired rate. This leaves us with the following four helicopter control inputs:

- u_1 and u_2: The latitudinal (left-right) and longitudinal (front-back) cyclic pitch controls. They are also often referred to by elevator and aileron. They cyclically change the pitch angle of the main rotor throughout each cycle and can thereby cause the helicopter to pitch forward/backwards or to roll sideways. By pitching and rolling the helicopter, the pilot can affect the direction of the main thrust, and hence the direction in which the helicopter moves.

- u_3: The tail rotor collective pitch control. This control is also often referred to by rudder and it affects the tail rotor thrust. It can be used to yaw (turn) the helicopter.

- u_4: The main rotor collective pitch control, similarly to the cyclic controls, causes the main rotor blades to rotate along an axis that turns along the length of the rotor blades, and thereby affects the angle at which the main rotor's blades are tilted relative to the plane of rotation. As the main rotor blades sweep through the air, they generate an amount of upward thrust that (generally) increases with this angle. By varying the collective pitch angle, we can affect the main rotor's thrust. For inverted flight, by setting a negative collective pitch angle, we can cause the helicopter to produce negative thrust.

To send the controls up to the helicopter, we use a standard hobby radio transmitter. In particular, we use the Spektrum transmitter/receiver setup, and we feed pulse

width modulated (PWM) signal into the buddy port of the transmitter. The buddy-switch on the transmitter controls whether our human pilot than well our computer is in control of the helicopter. To generate the control signals, we connect our PC through a serial port to an Atmel microcontroller. Similarly, to record the control signals (for logging and modeling purposes), we have a copy of the helicopter's receiver on the ground, and have it output the received controls through its servo lines, which we feed into a second Atmel, which in turn measures their pulse width lengths and sends them on to our PC over a serial port.

8.3 State estimation: our extended Kalman filter

Kalman filters and their extensions for non-linear systems, such as extended Kalman filters and unscented Kalman filters are widely used for state estimation. We refer the reader to, e.g., Kalman (1960); Gelb (1974); Bertsekas (2001); Anderson and Moore (1989) for more details on Kalman filters. We use an extended Kalman filter.[1] In this section we focus on the characteristics that are specific to our setup: (i) The choice of state space, measurement model and dynamics model, and (ii) A good way to represent 3D orientation.

For state estimation purposes (not for modeling and control, which we will cover later), we use the following variables in our state representation:

- (n, e, d): the North, East, Down position coordinates of the helicopter.
- $(\dot{n}, \dot{e}, \dot{d})$: the North, East, Down velocity.
- $(\ddot{n}, \ddot{e}, \ddot{d})$: the North, East, Down acceleration.
- (q_x, q_y, q_z, q_w): a quaternion representing the helicopter's orientation.
- (p, q, r): the angular rate of the helicopter along *its* forward, sideways and downward axis.

[1] For implementational convenience, we use numerical (finite-difference) linearizations of the non-linear measurement and dynamics functions.

- $(\dot{p}, \dot{q}, \dot{r})$: the angular acceleration of the helicopter along *its* forward, sideways and downward axis.

- (b_x, b_y, b_z): gyro bias terms, which track the (slowly varying) bias in each of the axes of the inertial unit's gyro's.

Our Kalman filter uses a very simple dynamics model: it assumes both linear and angular accelerations at the next time equal the linear and angular accelerations at the current time plus some Gaussian noise. Similarly, it assumes the bias terms remain the same up to Gaussian noise. Velocity, position, angular rate, and orientation are obtained through integration. We also add a very small, yet non-zero noise contribution to the orientation, as the Euler integration is only an approximation. We use very small variances, of the order of $1e-9$.

Our position sensing system (GPS or ground-based vision) gives us a noisy measurement of (n, e, d). The inertial unit gives us noisy measurements of: (i) acceleration plus gravity, as measured in the helicopter frame, (ii) the Earth's magnetic field, as measured in the helicopter frame, and (iii) the angular rate of the helicopter plus the bias on the gyros, i.e., $(p, q, r) + (b_x, b_y, b_z)$. Each of these are readily incorporated into the standard measurement updates for an (extended) Kalman filter.

For most state variables we use the standard linearization of the measurement and dynamics functions around the current state. However, we have a specific way of linearizing the orientation representation. Quaternions represent orientation with four variables, with the fourth variable being (crudely speaking) a deterministic nonlinear function of the three other variables. Naively treating the four quaternion variables as the other state variables results in a singular covariance matrix. This singularity is difficult to maintain numerically due to the accumulation of round-off error. In fact, the covariance matrix may even develop a negative eigenvalue. (Lefferts et al., 1982)

We found the following approach works better (see also Lefferts et al., 1982). The EKF uses a three dimensional "error quaternion" δq to represent the orientation, which is now represented relative to the current most likely orientation $\bar{q}$. Whenever the EKF updates the current state estimate, and hence changes δq, we "reset" the

orientation representation by updating the current most likely orientation $\bar{q} \leftarrow \bar{q} * \delta q$, and we reset $\delta q \leftarrow 0$. Here $*$ is a quaternion multiply. The uncertainty over orientation, represented by the covariance matrix in the EKF, is now over the error quaternion δq. To find the linearization (jacobian) F of a function f which takes in a quaternion, we compute for $i = 1, 2, 3$:

$$F_i = \frac{f(\bar{q} * dq_i)}{\epsilon},$$

where ϵ is a small number (e.g., $1e-4$), and $dq_i \in \mathbb{R}^4, dq_1 = (\epsilon, 0, 0, \sqrt{1-\epsilon^2}), dq_2 = (0, \epsilon, 0, \sqrt{1-\epsilon^2}), dq_3 = (0, 0, \epsilon, \sqrt{1-\epsilon^2})$ is a vector with all entries equal to zero, except for the i'th entry, which equals one. Note this is different from a standard linearization, which would instead compute $F_1 = \frac{f(\bar{q}+(\epsilon,0,0,0))}{\epsilon}$, and similarly for F_2, F_3.

To set the variances for the dynamics and the measurements, we ran the expectation-maximization (EM) algorithm, which alternates between running a Kalman filter/smoother and updating the variances. Although hand-tuning the variances had given us reasonable state estimation performance, the learned parameters performed significantly better, especially during high angular rate maneuvers. (See, e.g., Neal and Hinton, 1999 for more details on EM.)

8.4 Complete algorithm

Throughout this section, we will assume access to a reinforcement learning (or optimal control) algorithm which can solve for the optimal policy when given a dynamics model and a reward function. In Section 8.5 we will describe our reinforcement learning algorithm, receding horizon differential dynamic programming.

Our experimental procedure proceeds as follows:

1. Collect 20 minutes of flight data (we log the state estimates and control inputs over time) to build a crude dynamics model of the form described in Chapter 5. Typically we ask our pilot to excite each of the control sticks, as this reduces the amount of data collection required.

2. Collect about 10 demonstrations of the desired maneuver or airshow from our expert pilot. Typically about 5 of the demonstrations are reasonably high performance, and we choose these and feed them (together with the crude dynamics model from Step 1) into our trajectory learning algorithm described in Chapter 7. This gives us: (i) A target trajectory, and (ii) a set of locally weighted models, each of the form described in Chapter 5, which together provide a high accuracy dynamics model for the state-space region around the intended trajectory.

3. Choose a reward function that penalizes for deviation from the target trajectory. As the target trajectory is a feasible trajectory, we found that there is a fairly wide range of reward functions that works well. This was not the case before we started using feasible trajectories as our targets.

4. Run differential dynamic programming to find a sequence of quadratic cost-to-go functions for each time. The cost-to-go function for time t intends to capture the expected sum of costs accumulated from time t until the end of the trajectory when executing an optimal controller from then on.

5. Fly our helicopter autonomously: we run a receding horizon differential dynamic programming controller, which uses the cost-to-go functions from the off-line differential dynamic programming run.

6. If flight performance is satisfactory, we are done. Otherwise, incorporate the data from the autonomous flight to learn an improved dynamics model (see Chapters 3 and 7). Then go to Step 4.

8.5 Optimal control: receding horizon differential dynamic programming

8.5.1 Differential dynamic programming

The linear quadratic regulator (LQR) control problem is a special class of MDPs, for which the optimal policy can be computed efficiently. In LQR the set of states is given by $S = \mathbb{R}^n$, the set of actions/inputs is given by $\mathcal{A} = \mathbb{R}^p$, and the dynamics model is given by:

$$s(t+1) = A(t)s(t) + B(t)u(t) + w(t),$$

where for all $t = 0, \ldots, H$ we have that $A(t) \in \mathbb{R}^{n \times n}, B(t) \in \mathbb{R}^{n \times p}$ and $w(t)$ is a zero mean random variable (with finite variance). The reward for being in state $s(t)$ and taking action/input $u(t)$ is given by:

$$-s(t)^\top Q(t)s(t) - u(t)^\top R(t)u(t).$$

Here $Q(t), R(t)$ are positive semi-definite matrices which parameterize the reward function. It is well-known that the optimal policy for the LQR control problem is a linear feedback controller which can be efficiently computed using dynamic programming. Although the standard formulation presented above assumes the all-zeros state is the most desirable state, the formalism is easily extended to the task of tracking a desired trajectory $s_0^*, \ldots, s_H^*$. The standard extension (which we use) expresses the dynamics and reward function as a function of the error state $e(t) = s(t) - s^*(t)$ rather than the actual state $s(t)$. (See, e.g., Anderson and Moore, 1989, for more details on linear quadratic methods.)

Differential dynamic programming (DDP) approximately solves general continuous state-space MDPs by iterating the following two steps:

1. Compute a linear approximation to the dynamics and a quadratic approximation to the reward function around the trajectory obtained when using the current policy.
2. Compute the optimal policy for the LQR problem obtained in Step 1 and set

the current policy equal to the optimal policy for the LQR problem.

In our experiments, we have a quadratic reward function, thus the only approximation made in the first step is the linearization of the dynamics. To bootstrap the process, we linearize around the target trajectory in the first iteration.

To get DDP to converge we started from a model in which the control is trivial (the dynamics is modified such that helicopter automatically follows the target trajectory) and then slowly changed the model to the actual model. In particular, we change the model such that the next state is α times the target state plus $1-\alpha$ times the next state according to the true model. By slowly varying α from 0.999 to zero throughout DDP iterations, the linearizations obtained throughout are good approximations and DDP converges to a good policy.[2]

8.5.2 DDP Design Choices

Error state. We use the following error state $e = (\dot{x}^b - (\dot{x}^b)^*, \dot{y}^b - (\dot{y}^b)^*, \dot{z}^b - (\dot{z}^b)^*, x - x^*, y - y^*, z - z^*, \dot{\omega}_x^b - (\dot{\omega}_y^b)^*, \dot{\omega}_y^b - (\dot{\omega}_y^b)^*, \dot{\omega}_z^b - (\dot{\omega}_z^b)^*, \Delta_q)$. Here Δ_q is the axis-angle representation of the rotation that transforms the coordinate frame of the target orientation into the coordinate frame of the actual state. This axis angle representation results in the linearizations being more accurate approximations of the non-linear model since the axis angle representation maps more directly to the angular rates than naively differencing the quaternions or Euler angles.

Cost for change in inputs. Similar to frequency shaping for LQR controllers (see, e.g., Anderson and Moore, 1989), we added a term to the reward function that penalizes the change in inputs over consecutive time steps. In particular, this term will penalize the controller for wildly changing its input from its input at the previous time step.

Controller design in two phases. To allow aggressive maneuvering, while penalizing for changes in inputs, we split our controller design into two phases. In the first phase, we used DDP to find the open-loop input sequence that would be

[2]See, e.g., J. & P., 1998 for another example of homotopy/continuation methods applied in reinforcement learning.

optimal in the noise-free setting. (This can be seen as a planning phase and is similar to designing a feedforward controller in classical control.) In the second phase, we used DDP to design our actual flight controller, but we now redefine the inputs as the deviation from the nominal open-loop input sequence. Penalizing for changes in the new inputs penalizes only unplanned changes in the control inputs.

Integral control. Due to modeling error and wind, the controllers (so far described) have non-zero steady-state error. Each controller generated by DDP is designed using linearized dynamics. The orientation used for linearization greatly affects the resulting linear model. As a consequence, the linear model becomes significantly worse an approximation with increasing orientation error. This in turn results in the control inputs being less suited for the current state, which in turn results in larger orientation error, etc. To reduce the steady-state orientation errors—similar to the I term in PID control—we augment the state vector with integral terms for the orientation errors. More specifically, the state vector at time t is augmented with an additional three-dimensional vector $\sum_{\tau=0}^{t-1} 0.99^{t-\tau}\Delta_q(\tau)$. [3]

Factors affecting control performance. Our simulator included process noise (Gaussian noise on the accelerations as estimated when learning the model from data), measurement noise (Gaussian noise on the measurements as estimated from the Kalman filter residuals), as well as the Kalman filter and the low-pass filter, which is designed to remove the high-frequency noise from the IMU measurements.[4] Simulator tests showed that the low-pass filter's latency and the noise in the state estimates affect the performance of our controllers most. Process noise on the other hand did not seem to affect performance very much.

8.5.3 Receding horizon DDP

As the dynamics of the helicopter is highly non-linear, differential dynamic programming tends to result in good flight performance *only* when the linearizations used in

[3] When adding the integrated error in position to the cost we did not experience any benefits. Even worse, when increasing its weight in the cost function, the resulting controllers were often unstable.

[4] The high frequency noise on the IMU measurements is caused by the vibration of the helicopter. This vibration is mostly caused by the blades spinning at 30Hz.

the control design are a good approximation around the state the helicopter visits in reality. In practice, various perturbations, such as wind and unmodeled aspects of the dynamics, result in the helicopter deviating from the desired trajectory, which in turn results in the linearizations used during DDP being inaccurate, which in turn often results in quickly degrading performance.

To avoid depending on the off-line linearizations, we use *receding horizon DDP*, an instantiation of model predictive control (MPC). First we run DDP as described in the previous sections. During flight, rather than using the sequence of linear feedback controllers, at every discrete time control step, we re-run DDP with the current state as the starting state. This gives us the (locally) optimal controller from the current state.

In practice, it is infeasible to re-run DDP for the entire trajectory. Hence, we re-run DDP over a two-second horizon. As the cost incurred in the final state should represent the expected cost-to-go over the remainder of the trajectory, we use the quadratic cost-to-go function that we obtain from the off-line DDP run over the entire trajectory. We control at 20Hz, which gives us limited computational time, even for a two-second long trajectory. To ensure DDP finishes timely, we only run three iterations. When running DDP from scratch, three iterations is typically insufficient. Instead, we use the solution from the previous time-step as the starting point. If at some point DDP does not converge within three iterations, we use the off-line DDP controller as a fall-back. Then, on the next iteration, we restart the receding horizon DDP from scratch using $\alpha = \{0.99, 0.5, 0\}$.

8.6 Autonomous flight results

Our algorithm (described in Section 8.4) enabled our helicopter to fly a variety of challenging aerobatic maneuvers which have not been flown by any other autonomous helicopter to-date, including in-place flips, in-place rolls, tic-tocs, and complete aerobatic airshows which consist of a rapid sequence of several aerobatic maneuvers. For example, our "airshow 1" consists of the following maneuvers in rapid sequence:

split-S, snap roll, stall-turn, loop, loop with pirouette, stall-turn with pirouette, "hurricane" (fast backward funnel), knife-edge, flips and rolls, tic-toc and inverted hover. Our "airshow 2" consists of the following maneuvers in rapid sequence: pirouetting stall-turn, immelmann, split-S, slapper, tic-toc, stationary rolls, hurricane, sideways flip, forward inverted funnel, sideways tic-toc, stationary flips, traveling flips, inverted tail-slide, and inverted hover.

The trajectory-specific local model learning typically captures the dynamics well enough to fly all the aforementioned maneuvers reliably. Since our computer controller flies the trajectory very consistently, however, this allows us to repeatedly acquire data from the same vicinity of the target trajectory on the real helicopter. We incorporate this flight data into our model learning, allowing us to improve flight accuracy even further. For example, during the first autonomous execution of airshow 1 our controller achieves an RMS position error of 3.29 meters, and this procedure improved performance to 1.75 meters RMS position error.

Figures 8.3, 8.4, 8.5, 8.6, 8.7 illustrate our flight performance in detail. For each of the flight tasks: rolls, flips, tictoc, airshow 1, airshow 2. We plotted three of the pilot demonstrations (thin lines in blue), the target trajectory (thicker, dashed line in black), and a representative actually flown trajectory (thicker, dotted line in green). Inspecting the plots, we see that our autonomous flights closely track the target trajectory—typically more consistently than the expert. In fact, our expert agreed that our autonomous controller attained better flight performance than he could in his demonstrations.

Figure 8.8 shows a few snapshots of our Synergy N9 during autonomous aerobatics.

Videos of our autonomous flights (illustrating all the maneuvers) are available at the following url:

`http://heli.stanford.edu`.

8.6.1 Comparison with our earlier work

In earlier work, we did not follow the algorithm specified in Section 8.4, rather we used (i) a hand-specified target trajectory, and (ii) a single non-linear model (of the same

form), rather than locally weighted non-linear models. With this earlier approach, we managed to have our helicopter fly expert-level funnels, and *novice-level* stationary flips and rolls. While the flips and rolls were reliable using this earlier approach, they performed significantly less well then our expert and our current algorithm's controllers.

Figure 8.9 shows a quantitative comparison between our current algorithm and our earlier work. Figure 8.9 (a) shows the Y-Z position[5] and the collective (thrust) control inputs for the in-place rolls for both their controller and ours. Our controller achieves (i) better position performance (standard deviation of approximately 2.3 meters in the Y-Z plane, compared to about 4.6 meters and (ii) lower overall collective control values (which roughly represents the amount of energy being used to fly the maneuver). Similarly, Figure 8.9 (b) shows the X-Z position and the collective control inputs for the in-place flips for both controllers. Like for the rolls, we see that our current controller significantly outperforms that of our earlier work (Abbeel et al., 2007), both in position accuracy and in control energy expended.

When using this earlier approach, it was very challenging to specify the desired trajectory by hand. In fact, we partially succeeded (to some extent) for flips and rolls, we tried extensively to use the hand-coded approach for the tic-toc maneuver without any success. During the (tail-down) tic-toc maneuver the helicopter pitches quickly backward and forward in-place with the tail pointed toward the ground (resembling an inverted clock pendulum). The complex relationship between pitch angle, horizontal motion, vertical motion, and thrust makes it extremely difficult to create a feasible tic-toc trajectory by hand. Our attempts to use such a hand-coded trajectory failed repeatedly. By contrast, our current algorithm readily yields an excellent feasible trajectory that was successfully flown on the first attempt.

8.7 Discussion

We described the application of the apprenticeship learning methods presented in preceding chapters to the problem of autonomous helicopter flight. They have enabled

[5]These are the position coordinates projected into a plane orthogonal to the axis of rotation.

us to significantly extend the state of the art in autonomous helicopter aerobatics. In fact, our helicopter is the first capable of flying a very wide range of highly challenging aerobatics at the same level as human expert pilots.

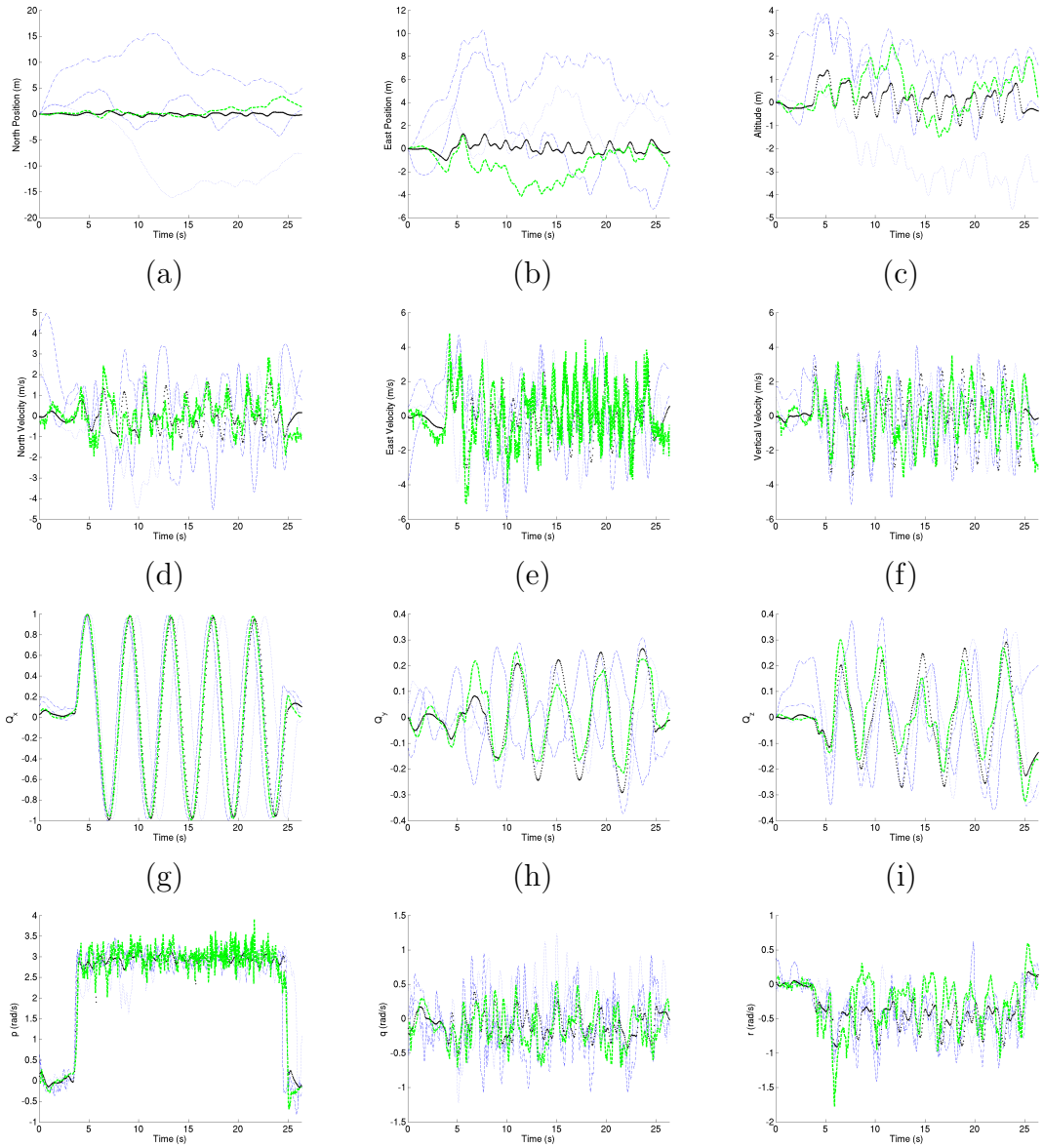


Figure 8.3: Rolls. Thick, dotted, green line: autonomous flight. Thick, dashed, black line: target trajectory. Thin, blue lines: three of the expert pilot's demonstrations. (Best viewed in color. See text for details.)

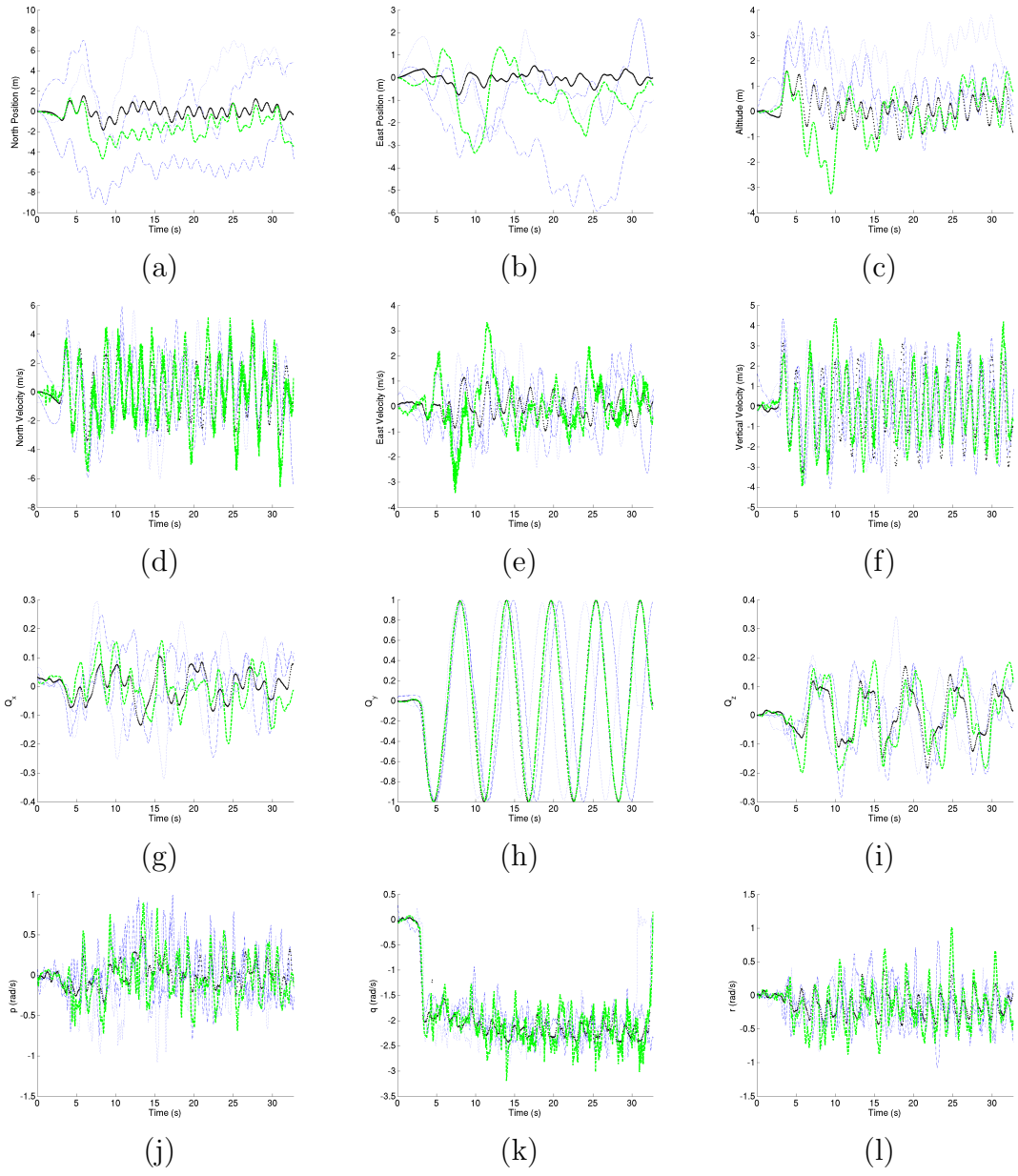


Figure 8.4: Flips. Thick, dotted, green line: autonomous flight. Thick, dashed, black line: target trajectory. Thin, blue lines: three of the expert pilot's demonstrations. (Best viewed in color. See text for details.)

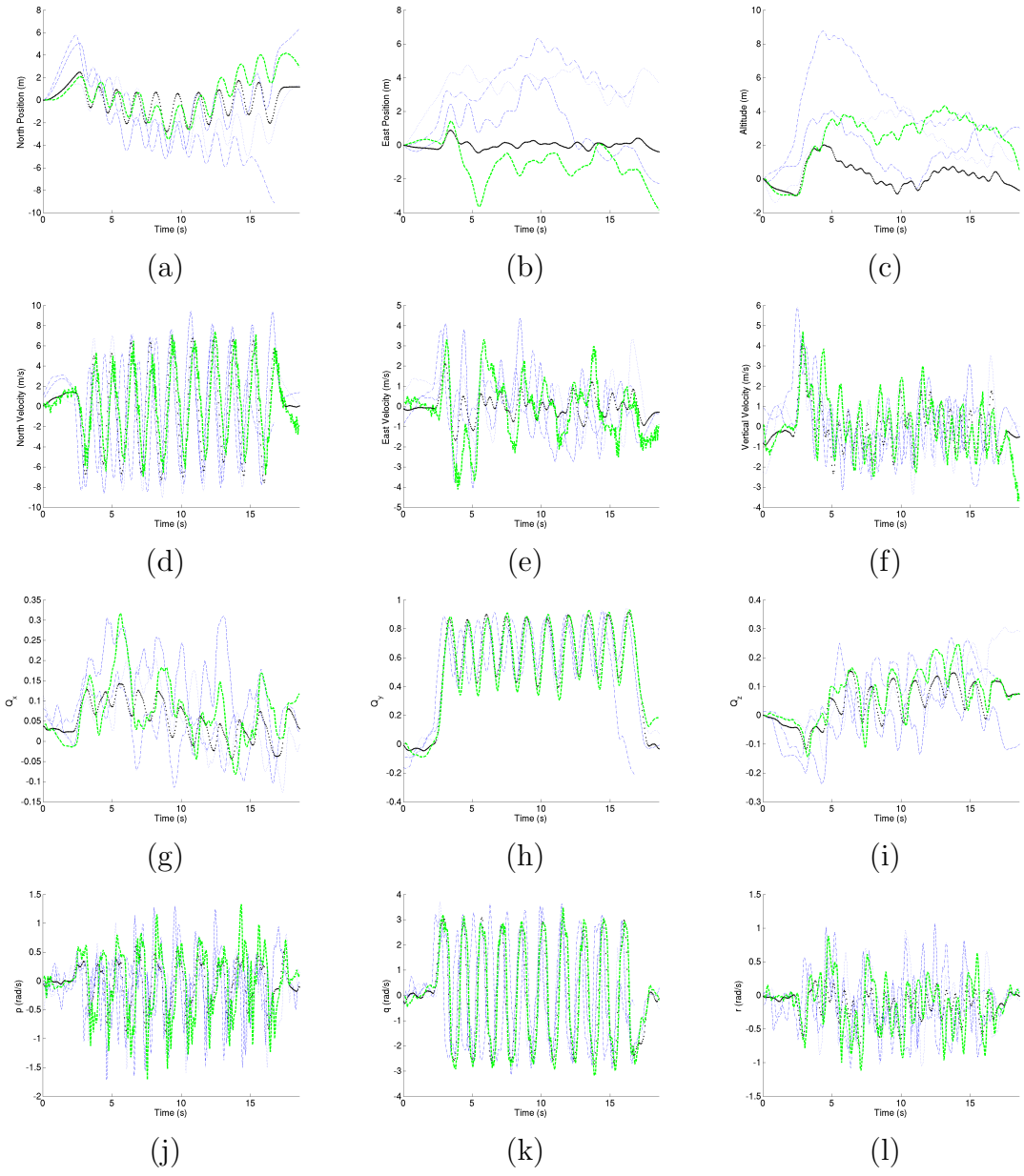


Figure 8.5: Tictocs. Thick, dotted, green line: autonomous flight. Thick, dashed, black line: target trajectory. Thin, blue lines: three of the expert pilot's demonstrations. (Best viewed in color. See text for details.)

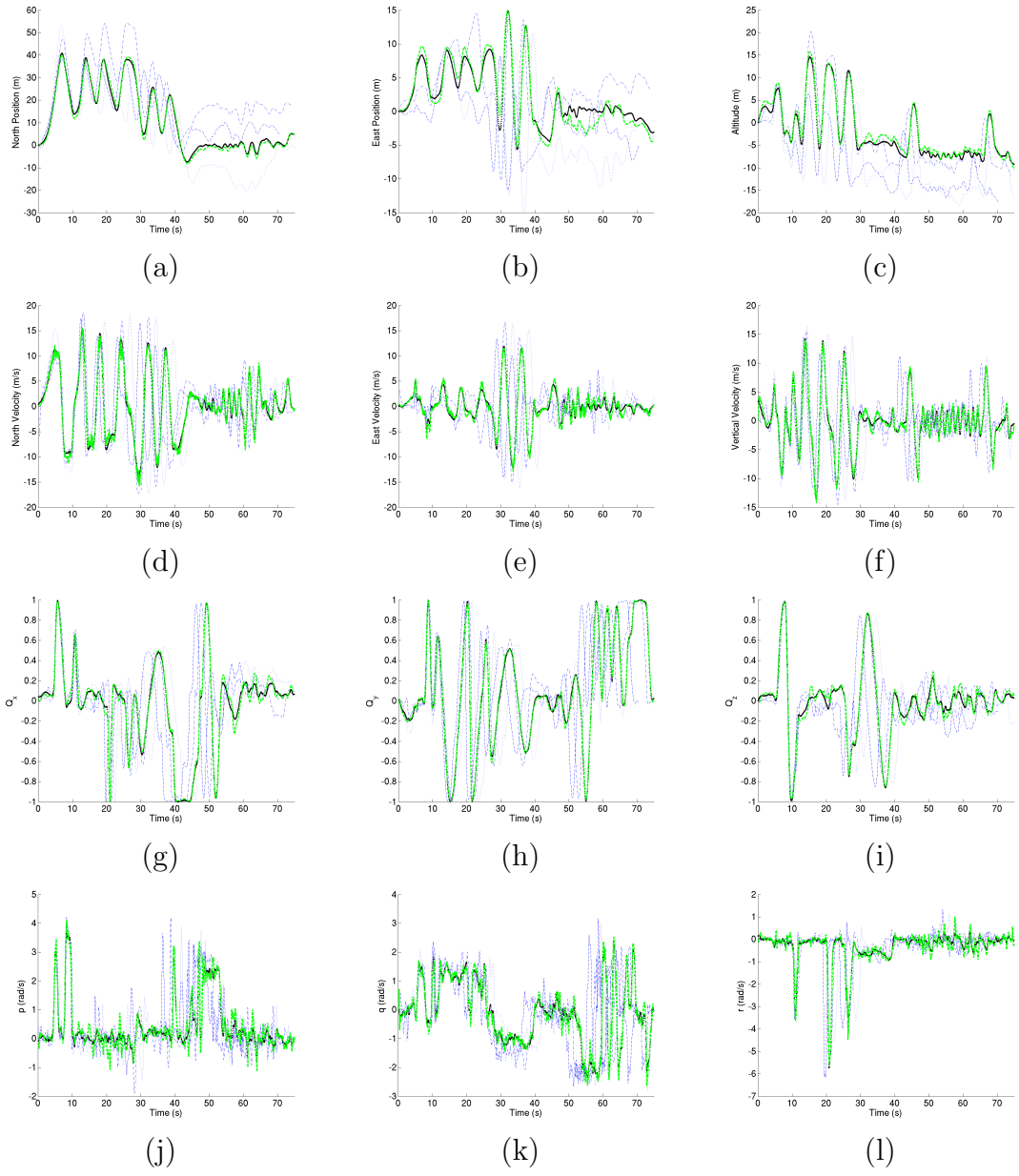


Figure 8.6: Airshow 1. Thick, dotted, green line: autonomous flight. Thick, dashed, black line: target trajectory. Thin, blue lines: three of the expert pilot's demonstrations. (Best viewed in color. See text for details.)

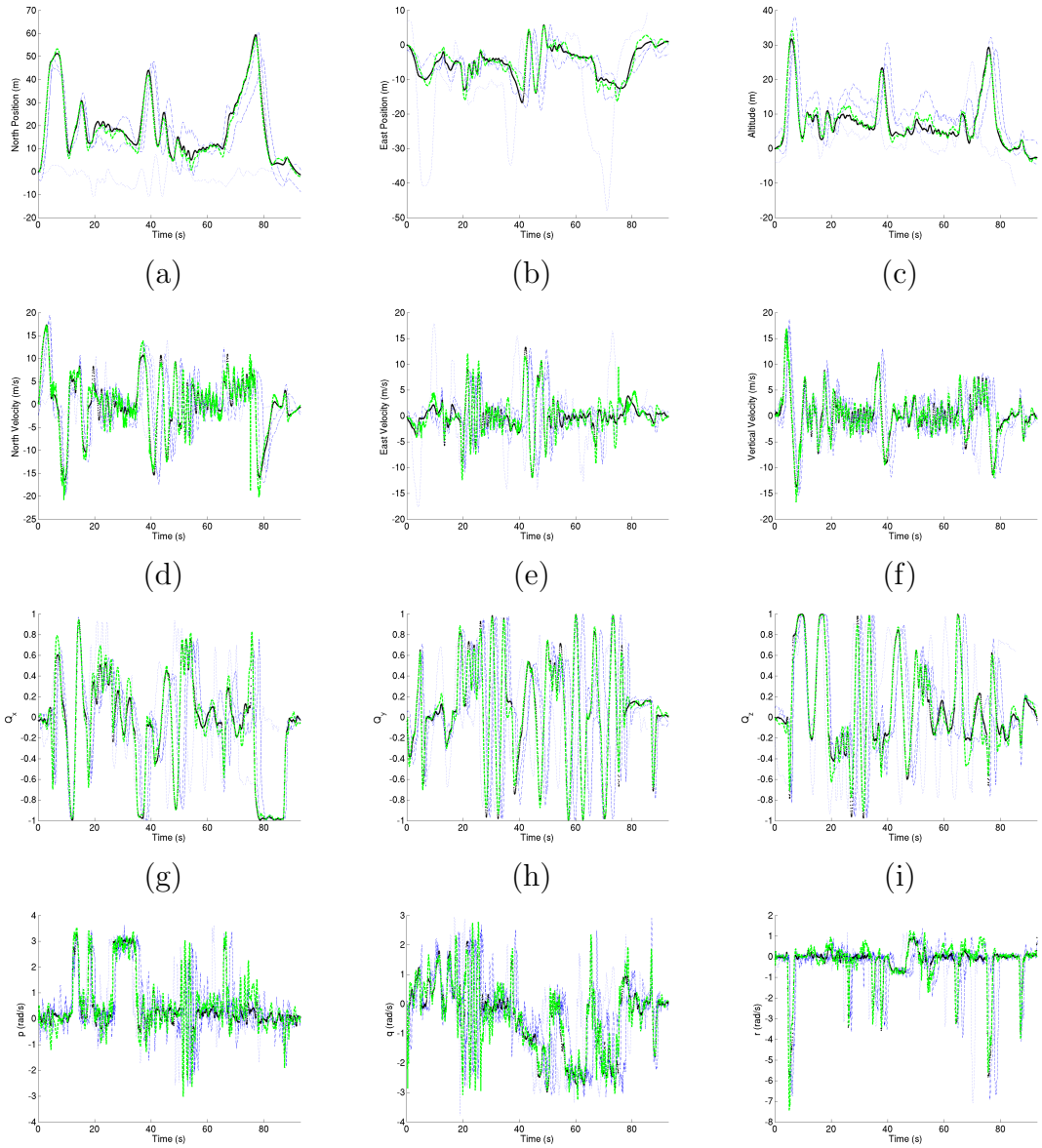


Figure 8.7: Airshow 2. Thick, dotted, green line: autonomous flight. Thick, dashed, black line: target trajectory. Thin, blue lines: three of the expert pilot's demonstrations. (Best viewed in color. See text for details.)

Figure 8.8: Snapshots of our Synergy N9 during autonomous aerobatics. Right column is (digitally) zoomed in on helicopter in corresponding picture in left column.

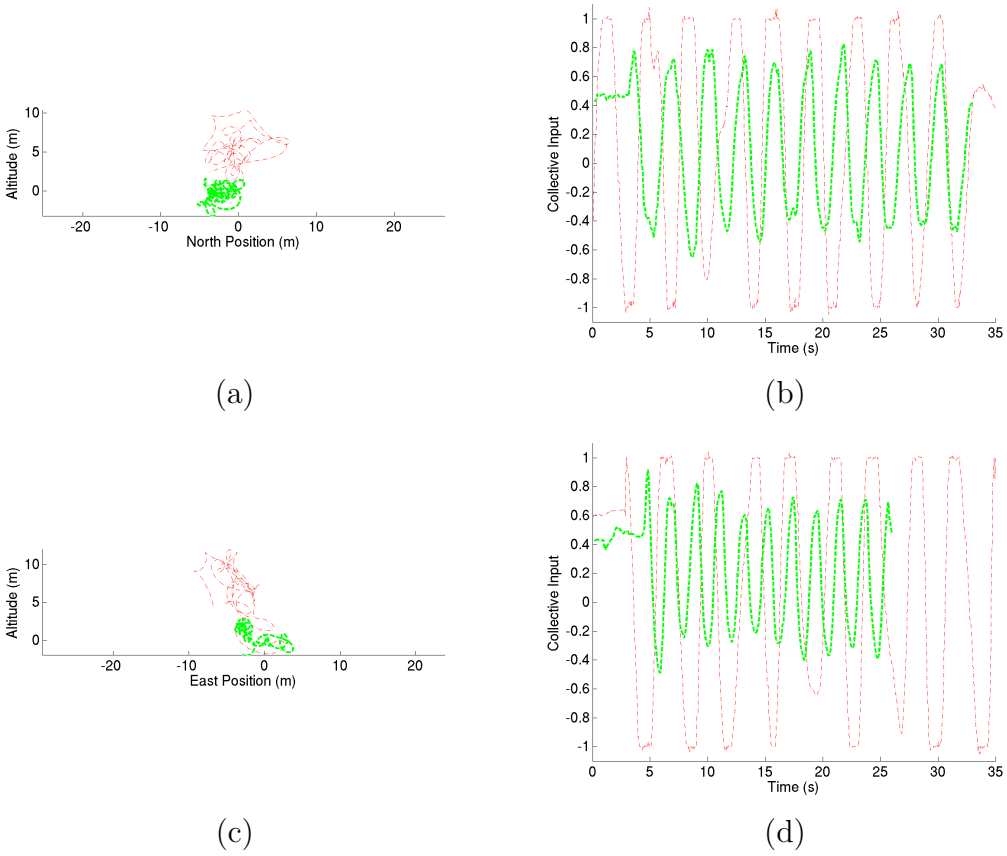


Figure 8.9: A comparison of autonomous flight results obtained with our algorithm described in Section 8.4 and our earlier work, which is the prior state of the art (Abbeel et al. 2007), which uses hand-specified target trajectories, and a single non-linear model. The figure compares (a) position during flips, (b) collective control input during flips, (c) position during rolls, and (d) collective control input during rolls. Red dash-dotted: our prior work; green dashed: algorithm presented in this chapter. (See text for details.)

Chapter 9

Conclusion

Some of the key challenges we encounter when applying traditional control and model-based reinforcement learning are: (i) We need to articulate the control task in a form that is amenable to control algorithms. (ii) We need to specify a dynamics model for the system. (iii) Even when these two issues are resolved, it can be computationally expensive to find good closed-loop controllers.

In this dissertation we described efficient algorithms that address each of these three issues in the apprenticeship learning setting—when we have access to expert demonstrations of the task at hand. Our apprenticeship learning algorithms leverage expert demonstrations to (i) Learn a description of the control task either in the form of a reward function that trades off between a (possibly large) set of known features, or in the form of an intended trajectory. (ii) Learn a dynamics model without requiring explicit exploration. We have also presented RL/optimal control algorithms that resolve (iii) in a variety of settings. Our apprenticeship learning algorithms are guaranteed to return a control policy that performs comparably to the expert.

We have not only provided theoretical guarantees, but have also shown how these algorithms have enabled us to solve some very challenging, previously unsolved control tasks. They have enabled a quadruped robot to traverse challenging, previously unseen terrain. They have significantly extended the state-of-the-art in autonomous helicopter flight. Our helicopter has performed by far the most challenging aerobatic

maneuvers performed by any autonomous helicopter to date, including maneuvers such as continuous in-place flips, rolls and tic-tocs, which only exceptional expert human pilots can fly. Our aerobatic flight performance is comparable to that of the best human pilots.

Appendix A

Proofs for Chapter 2

First we present a "greedy point approximation algorithm" and provide convergence guarantees. We show the relationship with our apprenticeship learning algorithms and then provide theoretical guarantees for our apprenticeship learning algorithms.

A.1 A greedy point approximation algorithm

In this section we present an algorithm that greedily approximates a point by a convex combination of points generated throughout the iterations of the algorithm. The points generated throughout the iterations of the algorithm cannot be arbitrarily chosen (as then merely choosing the target point would already solve the problem). Instead, one is restricted to choosing points by picking a weight vector w, and having a (possibly black-box) function return a point that is relatively far in the direction w. The perfect black-box function would return the point furthest in the direction w while still within the domain of feasible points.

Our apprenticeship learning algorithms are a special version of the presented greedy point approximation algorithm—in that, they use a reinforcement learning algorithm as the black-box function that generates a new point in each iteration.

Input:

- $x^* \in \mathbb{R}^n$,

- $\rho > 0, \epsilon > 0$.
- a function $f : \mathbb{R}^n \to \mathbb{R}^n$ which takes in a direction vector $w \in \mathbb{R}^n$ and returns a point $x = f(w) \in \mathbb{R}^n$ which satisfies:

$$x^\top \frac{w}{\|w\|_2} \geq x^{*\top} \frac{w}{\|w\|_2} - \rho, \tag{A.1}$$

  i.e., the returned point x is as far into the direction w as x^* up to some accuracy ρ.

The desired output from the greedy point approximation algorithm is a sequence of weight vectors $\{w^{(i)}\}_i$ and corresponding points $\{x^{(i)}\}_i$ such that a convex combination $\bar{x}$ of the points $\{x^{(i)}\}_i$ satisfies: $\|x^* - \bar{x}\|_2 \leq \rho + \epsilon$.

The greedy point approximation algorithm operates as follows:

1. Set $w^{(0)}$ to some random vector, compute $x^{(0)} = f(w^{(0)})$, set $\bar{x}^{(0)} = x^{(0)}$, set $i = 0$.
2. Set $w^{(i+1)} = x^* - \bar{x}^{(i)}$.
3. Compute $x^{(i+1)} = f(w^{(i+1)})$.
4. Find $\bar{x}^{(i+1)}$ as follows:
   (a) Max-margin version.

$$\begin{aligned} \min_{p,x} \quad & \|x^* - x\|_2^2 \\ \text{s.t.} \quad & x = Xp \\ & 1^\top p = 1 \\ & p \geq 0. \end{aligned}$$

   Here $X = \left[x^{(0)} x^{(1)} \dots x^{(i)}\right]$. Set $\bar{x}^{(i+1)} = x$.

(b) Projection version.

$$\begin{aligned} \min_{p,x} \quad & \|x^* - x\|_2^2 \\ \text{s.t.} \quad & x = \lambda x^{(i+1)} + (1-\lambda)\bar{x}^{(i)} \\ & 0 \leq \lambda \leq 1. \end{aligned}$$

Set $\bar{x}^{(i+1)} = x$.

The same point $\bar{x}^{(i+1)}$ can also be found by first computing:

$$x = \bar{x}^{(i)} + (x^* - \bar{x}^{(i)})^\top \frac{(x^{(i+1)} - \bar{x}^{(i)})}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2} \frac{(x^{(i+1)} - \bar{x}^{(i)})}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2}.$$

Then set $\bar{x}^{(i+1)} = x$ if x lies in the convex hull of $\bar{x}^{(i)}$ and $x^{(i+1)}$, otherwise set $\bar{x}^{(i+1)} = \text{argmin}_{x \in \{x^{(i+1)}, \bar{x}^{(i)}\}} \|x^* - x\|_2$.

5. If $\|x^* - \bar{x}^{(i+1)}\|_2 \leq \rho + \epsilon$ the algorithm converged and we return $\{x^{(i)}, i = 0, 1, \ldots\}$ and $\{w^{(i)}, i = 0, 1, \ldots\}$, otherwise set $i = i + 1$ and continue with Step 2.

Lemma 13. *Assume a fixed $\rho > 0$, $\epsilon > 0$, $x^* \in \mathbb{R}^n$. Assume a function $f : \mathbb{R}^n \to \mathbb{R}^n$ is given such that the point $x = f(w)$ is "reasonably" far in the direction w as measured relative to the target point, in that:*

$$x^\top \frac{w}{\|w\|_2} \geq x^{*\top} \frac{w}{\|w\|_2} - \rho, \tag{A.2}$$

Let $M = \max_{w_1, w_2 \in \mathbb{R}^n} \|f(w_1) - f(w_2)\|_2$. Then the greedy point approximation algorithm will have converged after at most $\frac{M^2}{\epsilon^2(1-(\frac{\rho}{\rho+\epsilon})^2)} - 1$ iterations. When converged, it will have found a set of points $\{x^{(0)}, x^{(1)}, \ldots\}$ such that a convex combination $\bar{x}$ of these points satisfies $\|x^* - \bar{x}\|_2 \leq \rho + \epsilon$.

Proof. The proof proceeds in three phases. First we show that the point

$$\bar{x}^{(i+1)} = \bar{x}^{(i)} + (1 - \frac{\rho}{\|x^* - \bar{x}^{(i)}\|_2})(x^* - \bar{x}^{(i)})^\top \frac{(x^{(i+1)} - \bar{x}^{(i)})}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2} \frac{(x^{(i+1)} - \bar{x}^{(i)})}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2}. \tag{A.3}$$

lies in the convex hull of $x^{(i)}$ and $\bar{x}^{(i+1)}$. Hence, it is a feasible point for the optimization problems of both the max-margin version and the projection version. Then we characterize the progress made in one iteration when using this point $\bar{x}^{(i+1)}$, which in turn implies progress made in one iteration for both the max-margin and projection version of the algorithm. Finally, we show how the per-iteration-progress implies the convergence stated in the lemma.

The point $\bar{x}^{(i+1)}$ as defined in Eqn. (A.3) lies in the convex hull of $x^{(i+1)}$ and $\bar{x}^{(i)}$: we have directly from Eqn. (A.3) that $\bar{x}^{(i+1)} = \lambda x^{(i+1)} + (1-\lambda)\bar{x}^{(i)}$, for $\lambda = (1-\frac{\rho}{\|x^*-\bar{x}^{(i)}\|_2})(x^*-\bar{x}^{(i)})^\top \frac{(x^{(i+1)}-\bar{x}^{(i)})}{\|x^{(i+1)}-\bar{x}^{(i)}\|_2} \frac{1}{\|x^{(i+1)}-\bar{x}^{(i)}\|_2}$. We have $\lambda \geq 0$ since $\|x^*-\bar{x}^{(i)}\|_2 \geq \rho$ and $(x^* - \bar{x}^{(i)})^\top (x^{(i+1)} - \bar{x}^{(i)}) \geq 0$ as long as the algorithm has not converged. We also have $\lambda \leq 1$, since we have, after re-arranging terms:

$$\begin{aligned} \lambda &= \frac{\|x^* - \bar{x}^{(i)}\|_2 - \rho}{\|x^* - \bar{x}^{(i)}\|_2} \frac{(x^{(i+1)} - \bar{x}^{(i)})^\top (x^* - \bar{x}^{(i)})}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2 \|x^* - \bar{x}^{(i)}\|_2} \\ &\leq \frac{\|x^* - \bar{x}^{(i)}\|_2 - \rho}{\|x^* - \bar{x}^{(i)}\|_2} \\ &\leq 1. \end{aligned}$$

Here we used Cauchy-Schwartz for the first step, and the second step is directly implied by Eqn. (A.2) for $w = \frac{x^*-\bar{x}^{(i)}}{\|x^*-\bar{x}^{(i)}\|_2}$ and $x = x^{(i+1)}$.

Now we will characterize progress in a single iteration. Using the definition of $\bar{x}^{(i+1)}$ of Eqn. (A.3) and simplification we obtain:

$$\|x^* - \bar{x}^{(i+1)}\|_2^2 = \|x^* - \bar{x}^{(i)}\|_2^2 - (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|_2^2}) \frac{\left((x^* - \bar{x}^{(i)})^\top (x^{(i+1)} - \bar{x}^{(i)})\right)^2}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2^2} \quad \text{(A.4)}$$

Using $w = x^* - \bar{x}^{(i)}$, Eqn. (A.2) becomes:

$$(x^{(i+1)} - \bar{x}^{(i)})^\top \frac{(x^* - \bar{x}^{(i)})}{\|x^* - \bar{x}^{(i)})\|_2} \geq \|x^* - \bar{x}^{(i)})\|_2 - \rho. \quad \text{(A.5)}$$

Combining Eqn. (A.5), Eqn. (A.4) and $\rho \geq \|x^* - \bar{x}^{(i)}\|_2$ before convergence, we

obtain:

$$\|x^* - \bar{x}^{(i+1)}\|_2^2 \leq \|x^* - \bar{x}^{(i)}\|_2^2 - (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|_2^2}) \frac{\left(\|x^* - \bar{x}^{(i)}\|_2^2 - \rho\|x^* - \bar{x}^{(i)}\|\right)^2}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2^2}.$$

Dividing both sides by $\|x^* - \bar{x}^{(i)}\|_2^2$ we obtain:

$$\frac{\|x^* - \bar{x}^{(i+1)}\|_2^2}{\|x^* - \bar{x}^{(i)}\|_2^2} \leq$$
$$1 - (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|_2^2}) \frac{(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2^2} =$$
$$\frac{\|x^{(i+1)} - \bar{x}^{(i)}\|_2^2 - (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|_2^2})(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2^2 - (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|_2^2})(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2 + (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|_2^2})(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2}.$$

As long as the algorithm has not converged, we have that $\|x^* - \bar{x}^{(i+1)}\| > 0$, and $\|x^* - \bar{x}^{(i)}\|_2 > 0$, hence $\frac{(\|x^* - \bar{x}^{(i+1)}\|_2 - \rho)^2}{(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2} \leq \frac{\|x^* - \bar{x}^{(i+1)}\|_2^2}{\|x^* - \bar{x}^{(i)}\|_2^2}$, and thus we have:

$$\frac{(\|x^* - \bar{x}^{(i+1)}\|_2 - \rho)^2}{(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2} \leq$$
$$\frac{\|x^{(i+1)} - \bar{x}^{(i)}\|_2^2 - (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|_2^2})(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2^2 - (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|_2^2})(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2 + (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|_2^2})(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2}.$$

As all terms are positive, we can rewrite this in the following form:

$$\frac{1}{(\|x^* - \bar{x}^{(i+1)}\|_2 - \rho)^2} \geq$$
$$\frac{1}{\left((\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2\right)} + \frac{(1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|^2})}{\|x^{(i+1)} - \bar{x}^{(i)}\|_2^2 - (1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|^2})(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2}.$$

Taking into account that all points returned by f have their 2-norm bounded by M, we obtain:

$$\frac{1}{\|x^* - \bar{x}^{(i+1)} - \rho\|_2^2} \geq \frac{1}{(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2} + \frac{1 - \frac{\rho^2}{\|x^* - \bar{x}^{(i)}\|^2}}{M^2}.$$

Finally, until convergence, we have that $\|x^* - \bar{x}^{(i)}\|_2 \geq \rho + \epsilon$, and hence we have the

following improvement in a single iteration of the algorithm:

$$\frac{1}{\|x^* - \bar{x}^{(i+1)} - \rho\|_2^2} \geq \frac{1}{(\|x^* - \bar{x}^{(i)}\|_2 - \rho)^2} + \frac{1 - \frac{\rho^2}{(\rho+\epsilon)^2}}{M^2}. \tag{A.6}$$

Expanding Eqn. (A.6) gives us

$$\frac{1}{(\|x^* - \bar{x}^{(i+1)}\|_2 - \rho)^2} \geq (i+2)\frac{1 - \frac{\rho^2}{(\rho+\epsilon)^2}}{M^2}.$$

Hence, for $\|x^* - \bar{x}^{(i+1)}\|_2 \leq \rho + \epsilon$ to hold, it suffices to run the algorithm for

$$\frac{M^2}{\epsilon^2(1 - (\frac{\rho}{\rho+\epsilon})^2)} - 1$$

iterations.

To complete the proof, it remains to show that $\bar{x}^{(i)}$ is a convex combination of $x^{(0)}, x^{(1)}, \ldots, x^{(i)}$. Through induction, it is sufficient to show the following: as long as the greedy point approximation algorithm has not converged, the point $\bar{x}^{(i+1)}$ lies in the convex hull of the points $\bar{x}^{(i)}$ and $x^{(i+1)}$. This is easily seen to hold true by construction: the points $\bar{x}^{(i+1)}$ are constrained to be such a convex combination, both in the max-margin and in the projection version. □

A.2 Proof of Theorem 1

Proof. We first establish that the number of Monte Carlo samples of the expert is sufficient to guarantee that with probability $1 - \delta$ we have that $\|\hat{\mu}_E - \mu_E\|_2 \leq \frac{\epsilon}{4}$.

Recall $\phi \in [0,1]^k$ so $\mu \in [0, \frac{1}{1-\gamma}]^k$. Let μ_i denote the i'th component of μ. Then applying the Hoeffding inequality on the m-sample estimate $(1-\gamma)\hat{\mu}_i$ of $(1-\gamma)\mu_i \in [0,1]$ gives

$$P((1-\gamma)|\mu_i - \hat{\mu}_i| > \tau) \leq 2\exp(-2\tau^2 m). \tag{A.7}$$

Using Eq. (A.7) for all i and the union bound gives

$$P(\exists i \in \{1 \dots k\}.(1-\gamma)|\mu_i - \hat{\mu}_i| > \tau) \leq 2k \exp(-2\tau^2 m),$$

which can be rewritten as

$$P((1-\gamma)\|\mu_E - \hat{\mu}_E\|_\infty \leq \tau) \geq 1 - 2k \exp(-2\tau^2 m).$$

Substituting $\tau = (1-\gamma)\epsilon/(4\sqrt{k})$ gives

$$P(\|\mu_E - \hat{\mu}_E\|_\infty \leq \frac{\epsilon}{4\sqrt{k}}) \geq 1 - 2k \exp(-2(\frac{\epsilon(1-\gamma)}{4\sqrt{k}})^2 m),$$

where k is the dimension of the feature vectors ϕ and feature expectations μ, and m is the number of sample trajectories used for the estimate $\hat{\mu}_E$. So if we take $m \geq \frac{4k}{(\epsilon(1-\gamma))^2} \log \frac{2k}{\delta}$ then with probability at least $(1-\delta)$ we have that $\|\mu_E - \hat{\mu}_E\|_\infty \leq \frac{\epsilon}{2\sqrt{k}}$. Using $\|\cdot\|_2 \leq \sqrt{k}\|\cdot\|_\infty$ for k-dimensional spaces, we get

$$\|\mu_E - \hat{\mu}_E\|_2 \leq \frac{\epsilon}{4} \text{ w.p. } 1-\delta \text{ for } m \geq \frac{4k}{(\epsilon(1-\gamma))^2} \log \frac{2k}{\delta}. \tag{A.8}$$

Now we can apply Lemma 13, which gives us that the algorithms will converge to within $\frac{\epsilon}{4}$ after at most

$$n \leq \frac{48k}{(1-\gamma)^2\epsilon^2}$$

iterations. Here we used $\rho = \frac{\epsilon}{2}$, as the MDP solver is $\frac{\epsilon}{4}$ accurate, and the Monte Carlo estimate of the expert's feature counts (our target point for the point approximation) might be outside the feasible set of feature expectations by at most $\frac{\epsilon}{4}$. We also used that the maximum distance between two feature expectation vectors equals $\frac{2}{1-\gamma}$— corresponding to M in the lemma.

At convergence, we have that we found a policy such that the corresponding feature expectations $\bar{\mu}$ satisfy:

$$\|\bar{\mu} - \hat{\mu}_E\|_2 \leq 3\frac{\epsilon}{4},$$

as the greedy point approximation converges within $\epsilon/4$, and uses an oracle that is $\rho = \epsilon/2$ accurate. As a result we have:

$$\|\bar{\mu} - \mu_E\|_2 \leq \epsilon. \tag{A.9}$$

This in turn implies that:

$$w^{*\top}\bar{\mu} \geq w^{*\top}\mu_E - \epsilon\|w^*\|_2.$$

As we have $\|w^*\|_2 \leq \|w^*\|_1 \leq 1$, this implies the statement of the theorem.

□

Note that in case the underlying reward function R^* does not lie exactly in the span of basis functions, we have graceful degradation of performance. Let $R^*(s) = w^{*\top}\phi(s) + f(s)$, then it is easy to see from Eqn. (A.9) in the proof above, that performance degradation is bounded by $\frac{\|f\|_\infty}{1-\gamma}$.

It has come to our attention that our proof has some similarities with the proof in Jones (1992), which also considers convergence rates of a greedy approximation algorithm.

Appendix B

Proofs for Chapter 3

B.1 Some properties of the variational distance

In this section we state some known results involving the variational distance between probability distributions. (We include proofs to keep the paper self-contained.) We prove (and sometimes state) the propositions for probability distributions over a discrete domain only. The proofs (and statements) for continuous domains are very similar.

Proposition 14. *Let Q, Q^* be probability distributions over a domain $\mathcal{X}$, let f be a bounded random variable over $\mathcal{X}$. Then*

$$|E_Q f - E_{Q^*} f| \leq (\sup_{x \in \mathcal{X}} f(x) - \inf_{x \in \mathcal{X}} f(x)) d_{\text{var}}(Q, Q^*).$$

Proof. Let $c = \inf_{x\in\mathcal{X}} f(x)$. Then we have

$$\begin{aligned}
& E_Q f - E_{Q^*} f \\
= \;& E_Q(f-c) - E_{Q^*}(f-c) \\
= \;& \sum_{x:Q(x)>Q^*(x)} (f(x)-c)(Q(x)-Q^*(x)) \\
& + \sum_{x:Q(x)\le Q^*(x)} (f(x)-c)(Q(x)-Q^*(x)) \\
\le \;& \sum_{x:Q(x)>Q^*(x)} (f(x)-c)(Q(x)-Q^*(x)) \\
\le \;& \sup_{x\in\mathcal{X}}(f(x)-c) \sum_{x:Q(x)>Q^*(x)} (Q(x)-Q^*(x)) \\
= \;& (\sup_{x\in\mathcal{X}} f(x) - \inf_{x\in\mathcal{X}} f(x)) d_{\text{var}}(Q,Q^*). \qquad \text{(B.1)}
\end{aligned}$$

Here we used in order: adding and subtracting c; splitting the summation into positive and negative terms; dropping the negative terms from the summation; $f(x)$ is bounded by $\sup_{x\in\mathcal{X}} f(x)$; definition of d_{var}[1] and c. The same argument with roles of Q and Q^* interchanged gives us:

$$E_Q f - E_{Q^*} f \le (\sup_{x\in\mathcal{X}} f(x) - \inf_{x\in\mathcal{X}} f(x)) d_{\text{var}}(Q,Q^*). \qquad \text{(B.2)}$$

Eqn. (B.1) and (B.2) combined prove the proposition. □

Proposition 15. *Let $Q_0(\cdot)$ and $Q_0^*(\cdot)$ be probability distributions over a domain $\mathcal{X}$. Let $\forall x \in \mathcal{X}$ $P(\cdot|x)$ be a probability distribution over $\mathcal{X}$. Let*

$$Q_1(\cdot) = \sum_{x_0\in\mathcal{X}} P(\cdot|x_0) Q_0(x_0)$$

and let

$$Q_1^*(\cdot) = \sum_{x_0\in\mathcal{X}} P(\cdot|x_0) Q_0^*(x_0),$$

[1]Let $P(\cdot), Q(\cdot)$ be two probability distributions over a set $\mathcal{X}$, then in the main body we defined the variational distance $d_{\text{var}}(P,Q)$ as follows: $d_{\text{var}}(P,Q) = \frac{1}{2}\int_{x\in\mathcal{X}} |P(x)-Q(x)|$. It is well known the following definition is equivalent: $d_{\text{var}}(P,Q) = \int_{x\in\mathcal{X}:P(x)>Q(x)} |P(x)-Q(x)|$.

then

$$d_{\text{var}}(Q_1(\cdot), Q_1^*(\cdot)) \leq d_{\text{var}}(Q_0(\cdot), Q_0^*(\cdot)).$$

Proof. We have

$$\begin{aligned}
& d_{\text{var}}(Q_1(\cdot), Q_1^*(\cdot)) \\
= \; & \frac{1}{2} \sum_{x_1 \in \mathcal{X}} \Big| \sum_{x_0 \in \mathcal{X}} P(x_1|x_0) Q_0(x_0) - P(x_1|x_0) Q_0^*(x_0) \Big| \\
= \; & \frac{1}{2} \sum_{x_1 \in \mathcal{X}} \Big| \sum_{x_0 : Q_0(x_0) > Q_0^*(x_0)} P(x_1|x_0)(Q_0(x_0) - Q_0^*(x_0)) \\
& + \sum_{x_0 : Q_0(x_0) < Q_0^*(x_0)} P(x_1|x_0)(Q_0(x_0) - Q_0^*(x_0)) \Big| \\
\leq \; & \frac{1}{2} \sum_{x_1 \in \mathcal{X}} \Big(\sum_{x_0 : Q_0(x_0) > Q_0^*(x_0)} P(x_1|x_0)(Q_0(x_0) - Q_0^*(x_0)) \\
& + \sum_{x_0 : Q_0(x_0) < Q_0^*(x_0)} P(x_1|x_0)(-Q_0(x_0) + Q_0^*(x_0)) \Big) \\
= \; & \frac{1}{2} \sum_{x_0 : Q_0(x_0) > Q_0^*(x_0)} (Q_0(x_0) - Q_0^*(x_0)) \\
& + \frac{1}{2} \sum_{x_0 : Q(x_0) < Q_0^*(x_0)} (-Q_0(x_0) + Q_0^*(x_0)) \\
= \; & d_{\text{var}}(Q_0(\cdot), Q_0^*(\cdot)).
\end{aligned}$$

Here we used in order: the definition of d_{var}; splitting the summation into two summations; triangle inequality (all terms are now positive); switching order of summation; the definition of d_{var}. □

Proposition 16. *Let $Q_0(\cdot)$ and $Q_0^*(\cdot)$ be probability distributions over a domain $\mathcal{X}$. Let $\forall x \in \mathcal{X}$ $P^*(\cdot|x)$ and $P(\cdot|x)$ be probability distributions over $\mathcal{X}$. Let $Q_1(\cdot) = \sum_{x_0 \in \mathcal{X}} P(\cdot|x_0) Q_0(x_0)$ and let $Q_1^*(\cdot) = \sum_{x_0 \in \mathcal{X}} P^*(\cdot|x_0) Q_0^*(x_0)$, then*

$$\begin{aligned}
d_{\text{var}}(Q_1(\cdot), Q_1^*(\cdot)) \; \leq \; & d_{\text{var}}(Q_0(\cdot), Q_0^*(\cdot)) \\
& + \sup_{x \in \mathcal{X}} d_{\text{var}}(P(\cdot|x), P^*(\cdot|x)).
\end{aligned}$$

Proof. Let $\bar{Q}_1(\cdot) = \sum_{x_0 \in \mathcal{X}} P(\cdot|x_0) Q_0^*(x_0)$. Due to triangle inequality we have

$$d_{\text{var}}(Q_1(\cdot), Q_1^*(\cdot)) \leq d_{\text{var}}(Q_1(\cdot), \bar{Q}_1(\cdot)) + d_{\text{var}}(\bar{Q}_1(\cdot), Q_1^*(\cdot)). \tag{B.3}$$

For the first term Proposition 15 gives us

$$d_{\text{var}}(Q_1(\cdot), \bar{Q}_1(\cdot)) \leq d_{\text{var}}(Q_0(\cdot), Q_0^*(\cdot)). \tag{B.4}$$

For the second term we have

$$\begin{aligned}
& d_{\text{var}}(\bar{Q}_1(\cdot), Q_1^*(\cdot)) \\
= \; & \frac{1}{2} \sum_{x_1 \in \mathcal{X}} \Big| \sum_{x_0 \in \mathcal{X}} P(x_1|x_0) Q_0^*(x_0) \\
& \qquad - \sum_{x_0} P^*(x_1|x_0) Q_0^*(x_0) \Big| \\
\leq \; & \frac{1}{2} \sum_{x_1 \in \mathcal{X}, x_0 \in \mathcal{X}} |P(x_1|x_0) - P^*(x_1|x_0)| \, Q_0^*(x_0) \\
= \; & \sum_{x_0 \in \mathcal{X}} d_{\text{var}}(P(\cdot|x_0), P^*(\cdot|x_0)) Q_0^*(x_0) \\
\leq \; & \sup_{x_0 \in X} d_{\text{var}}(P(\cdot|x_0), P^*(\cdot|x_0)).
\end{aligned} \tag{B.5}$$

Combining Eqn. (B.3), (B.4) and (B.5) gives us the statement of the proposition. □

B.2 Proofs for Section 3.2

B.2.1 Proof of Lemma 2

We first prove the following lemma.

Lemma 17. *Let an MDP $M = (S, \mathcal{A}, T, H, D, R)$ be given. Let another MDP $\hat{M} = (S, \mathcal{A}, \hat{T}, H, D, R)$ – which only differs from M in its transition probabilities – be given. Let any $\epsilon > 0$ be given. If $\hat{T} = \{\hat{P}(\cdot|s,a)\}_{s,a}$ satisfies*

$$\forall s \in S, a \in A \;\; d_{\text{var}}(P(\cdot|s,a), \hat{P}(\cdot|s,a)) \leq \epsilon,$$

then we have for any policy π mapping from S to (probability distributions over) A that

$$|U_M(\pi) - U_{\hat{M}}(\pi)| \leq H^2 \epsilon R_{\max}.$$

Proof. With some abuse of notation, let P_t and $\hat{P}_t$ denote the distributions over states at time t induced by the policy π, the initial state distribution D and the transition probabilities T and $\hat{T}$ respectively. Then from using Prop. 16 inductively we have for all $t \in 0 : H$ that

$$d_{\text{var}}(P_t(\cdot), \hat{P}_t(\cdot)) \leq t\epsilon. \tag{B.6}$$

So we get that

$$\begin{aligned}
|U_M(\pi) - U_{\hat{M}}(\pi)| &= \Big| \sum_{t=0}^{H} \mathrm{E}_{P_t}[R(s_t)] - \sum_{t=0}^{H} \mathrm{E}_{\hat{P}_t}[R(s_t)] \Big| \\
&= \Big| \sum_{t=0}^{H} \mathrm{E}_{P_t}[R(s_t)] - \mathrm{E}_{\hat{P}_t}[R(s_t)] \Big| \\
&\leq \sum_{t=0}^{H} \Big| \mathrm{E}_{P_t} R(s_t) - \mathrm{E}_{\hat{P}_t} R(s_t) \Big| \\
&\leq \sum_{t=0}^{H} d_{\text{var}}(P_t, \hat{P}_t) R_{\max} \\
&\leq \sum_{t=0}^{H} t\epsilon R_{\max} \\
&\leq H^2 \epsilon R_{\max}.
\end{aligned}$$

Where we used in order the definition of U; reordering terms; standard inequality for the absolute value of a sum; Prop. 14; Eqn. (B.6); simple algebra. □

Proof of Lemma 2. Consider the auxiliary MDP $M^{(1)} = (S, \mathcal{A}, T^{(1)}, H, D, R)$, where $P^{(1)}(\cdot|s,a) = P(\cdot|s,a)$ if $(s,a) \in \overline{SA}_\eta$ and $P^{(1)}(\cdot|s,a) = \hat{P}(\cdot|s,a)$ otherwise. Then we have $\forall s \in S, a \in \mathcal{A}$ $d_{\text{var}}(\hat{P}(\cdot|s,a), P^{(1)}(\cdot|s,a)) \leq \epsilon$. Using Lemma 17 gives us that

$$|U_{\hat{M}}(\pi) - U_{M^{(1)}}(\pi)| \leq H^2 \epsilon R_{\max}. \tag{B.7}$$

Also, using condition (ii) of Lemma 2 and the definitions of $M, M^{(1)}$ we trivially get

$$|U_{M^{(1)}}(\pi) - U_M(\pi)| \leq \eta H R_{\max}. \tag{B.8}$$

Combining Eqn. (B.7) and Eqn. (B.8) using the triangle inequality gives the statement of the lemma. □

B.2.2 Proof of Lemma 3

We prove the following slightly stronger lemma, of which the statement in Lemma 3 is a subset.

Lemma 18. *Let any $\delta > 0$ and $a > 0$ be given. Let $\{X_i\}_{i=1}^m$ be IID Bernoulli(ϕ) random variables. Then for*

$$P(\sum_{i=1}^{m} X_i \geq a) \tag{B.9}$$

to hold with probability $1 - \delta$ it suffices that

$$m \geq \frac{1}{\phi}\left(a + \log\frac{1}{\delta} + \sqrt{(a + \log\frac{1}{\delta})^2 - a^2}\right). \tag{B.10}$$

We can relax the condition above to get the following less tight, but simpler sufficient condition on m:

$$m \geq \frac{2}{\phi}(a + \log\frac{1}{\delta}).$$

If $a \geq 2$ and $\log\frac{1}{\delta} \geq 2$ we have that the following is a sufficient condition on m

$$m \geq \frac{2}{\phi} a \log\frac{1}{\delta}.$$

Proof. We start from the multiplicative 1-sided Chernoff bound for a Bernoulli distribution, with expectation ϕ. Let $\epsilon > 0$ then

$$P(\sum_{i=1}^{m} X_i < m(\phi - \epsilon\sqrt{\phi})) \leq \exp(\frac{-m\epsilon^2}{2}). \tag{B.11}$$

Now let $a = m(\phi - \epsilon\sqrt{\phi})$, which implies $\epsilon = \frac{m\phi - a}{m\sqrt{\phi}}$. Substituting this into Eqn. (B.11) gives

$$\begin{aligned} P(\sum_{i=1}^{m} X_i < a) &\leq \exp(\frac{-m}{2}(\frac{m\phi - a}{m\sqrt{\phi}})^2) \\ &= \exp(\frac{-1}{2}(\frac{m\phi - a}{\sqrt{m\phi}})^2). \end{aligned} \tag{B.12}$$

Note for Eqn. (B.11) to be valid, we needed $\epsilon > 0$. So Eqn. (B.12) is only valid if

$$m\phi - a > 0. \tag{B.13}$$

So for $P(\sum_{i=1}^{m} X_i < a) \leq \delta$ to hold, it is sufficient that Eqn. (B.13) holds and that

$$\exp(\frac{-1}{2}(\frac{m\phi - a}{\sqrt{m\phi}})^2) \leq \delta.$$

By taking log on both sides, and multiplying with -2 we get the equivalent condition

$$(\frac{m\phi - a}{\sqrt{m\phi}})^2 \geq 2\log\frac{1}{\delta}.$$

Algebraic manipulation gives the equivalent condition

$$m^2\phi^2 - 2(a + \log\frac{1}{\delta})m\phi + a^2 \geq 0. \tag{B.14}$$

Eqn. (B.14) is satisfied if

$$\begin{aligned} m\phi \in &\Big(-\infty, a + \log\frac{1}{\delta} - \sqrt{(a + \log\frac{1}{\delta})^2 - a^2}\Big] \\ &\bigcup \Big[a + \log\frac{1}{\delta} + \sqrt{(a + \log\frac{1}{\delta})^2 - a^2}, \infty\Big). \end{aligned}$$

Now recall that the Chernoff bound was only meaningful for $\epsilon > 0$, which corresponds to the condition given by Eqn. (B.13). So we get that for $P(\sum_{i=1}^{m} X_i < a) \leq \delta$ to

hold it suffices that

$$m\phi \geq a + \log\frac{1}{\delta} + \sqrt{(a + \log\frac{1}{\delta})^2 - a^2}.$$

□

B.3 Proofs for Section 3.6

B.3.1 Proof of Lemma 5

We first prove the following lemma.

Lemma 19. *Let any $\epsilon, \delta > 0$ be given. Let $X_i \sim$ Bernoulli(ϕ) be IID random variables. Let $\hat{\phi}_n$ be the estimate for ϕ after n observations. I.e., we have $\hat{\phi}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then for*

$$\max_{n\geq N}|\phi - \hat{\phi}_n| \leq \epsilon$$

to hold with probability $1 - \delta$, it suffices that

$$N \geq \frac{1}{\epsilon^2}\log(\frac{2}{\delta\epsilon}).$$

Proof.

$$\begin{aligned} P(\max_{n\geq N}|\phi - \hat{\phi}_n| > \epsilon) &= P(\bigcup_{n\geq N}\{|\phi - \hat{\phi}_n| > \epsilon\}) \\ &\leq \sum_{n\geq N} P(|\phi - \hat{\phi}_n| > \epsilon) \\ &\leq \sum_{n\geq N} 2e^{-2\epsilon^2 n} \\ &= \frac{2(e^{-2\epsilon^2})^N}{1 - e^{-2\epsilon^2}} \end{aligned}$$

Hence, in order to guarantee that $P(\max_{n\geq N}|\phi - \hat{\phi}_n| > \epsilon) \leq \delta$, it is sufficient for

N to satisfy

$$\begin{aligned}
\frac{2(e^{-2\epsilon^2})^N}{1-e^{-2\epsilon^2}} &\leq \delta \\
(e^{-2\epsilon^2})^N &\leq \delta(1-e^{-2\epsilon^2})/2 \\
\log((e^{-2\epsilon^2})^N) &\leq \log(\delta(1-e^{-2\epsilon^2})/2) \\
(-2\epsilon^2)N &\leq \log\delta + \log(1-e^{-2\epsilon^2}) - \log 2 \\
N &\geq \frac{1}{2\epsilon^2}(-\log\delta - \log(1-e^{-2\epsilon^2}) + \log 2) \\
N &\geq \frac{1}{2\epsilon^2}(\log\frac{1}{\delta} + \log(\frac{1}{1-e^{-2\epsilon^2}}) + \log 2).
\end{aligned}$$

Now using the fact that for any $x : -\xi < x < 0$ we have that $\exp(x) < 1 - \frac{|x|}{\xi}(1 - \exp(-\xi))$, and more specifically, for any $x : -2 < x < 0$ we have that $\exp(x) < 1 - \frac{|x|}{2}(1-\exp(-2)) < 1 - |x|/4$. Thus for $\epsilon \in (0,1)$ it is sufficient to have

$$\begin{aligned}
N &\geq \frac{1}{2\epsilon^2}(\log\frac{1}{\delta} + \log\frac{1}{1-(1-(2\epsilon^2)/4)} + \log 2) \\
N &\geq \frac{1}{2\epsilon^2}(\log\frac{1}{\delta} + \log\frac{2}{\epsilon^2} + \log 2) \\
N &\geq \frac{1}{\epsilon^2}\log\frac{2}{\delta\epsilon}.
\end{aligned}$$

In case $\epsilon \notin (0,1)$, more specifically when $\epsilon \geq 1$, the lemma holds with probability one independent of the number of samples. □

Proof of Lemma 5. Let the multinomial distribution P be parameterized by the k parameters $(\phi^{(1)}, \cdots, \phi^{(k)})$. Let $\hat{P}_n$ be parameterized by $(\hat{\phi}_n^{(1)}, \cdots, \hat{\phi}_n^{(k)})$ which are the respective n-sample estimates for all parameters $\phi^{(\cdot)}$. Then from Lemma 19 and the union bound we have that for

$$\forall i \in 1:k \;\; \max_{n\geq N}|\phi^{(i)} - \hat{\phi}_n^{(i)}| \leq \epsilon'$$

to hold with probability $1 - k\delta'$, it suffices that

$$N \geq \frac{1}{\epsilon'^2} \log(\frac{2}{\delta'\epsilon'}). \tag{B.15}$$

Now we use the fact that

$$d_{\text{var}}(P, \hat{P}_n) \;\; = \;\; \frac{1}{2} \sum_{i=1}^{k} |\phi^{(i)} - \hat{\phi}_n^{(i)}|.$$

We get that condition (B.15) is sufficient to ensure that

$$\max_{n \geq N} d_{\text{var}}(P, \hat{P}_n) \leq \frac{k}{2}\epsilon'$$

holds with probability at least $1 - k\delta'$. Choosing $\epsilon = \frac{k}{2}\epsilon'$ and $\delta = k\delta'$ proves the lemma. □

B.3.2 Proof of Lemma 6

We first show that state-action pairs that are visited with probability sufficiently bounded away from zero under a policy π will be accurately modeled after observing sufficient trials under that policy π.

Lemma 20. *Let an MDP $(S, \mathcal{A}, T, H, D, R)$ be given. Let π be a policy for this MDP. Let any $\epsilon, \delta, \xi > 0$ be given. Let $\{\hat{P}_n(\cdot|s,a)\}_{s,a}$ be the maximum likelihood transition probability estimates based upon observing the policy π for n trials of duration H. Let $SA_\xi \subseteq S \times A$ be the set of state-action pairs such that the probability of seeing any specific state-action pair $(s,a) \in SA_\xi$ under the policy π in a single trial of duration H is at least ξ. Then for*

$$\forall n \geq N, \;\; \forall (s,a) \in SA_\xi, d_{\text{var}}(P(\cdot|s,a), \hat{P}_n(\cdot|s,a)) \leq \epsilon$$

to hold with probability $1 - \delta$, it suffices that

$$N \geq \frac{2}{\xi}(\frac{|S|^2}{4\epsilon^2} \log \frac{2|S|^3|\mathcal{A}|}{\delta\epsilon} + \log \frac{2|S||\mathcal{A}|}{\delta}).$$

Proof. Let $\tilde{P}_k(\cdot|s,a)$ denote the transition probability estimate after observing the state-action pair (s,a) k times. From Lemma 5, for

$$\forall k \geq K, \quad d_{\text{var}}(P(\cdot|s,a), \tilde{P}_k(\cdot|s,a)) \leq \epsilon$$

to hold with probability $1-\delta'$, it suffices that

$$K \geq \frac{|S|^2}{4\epsilon^2} \log \frac{|S|^2}{\delta'\epsilon}.$$

Now for $(s,a) \in SA_\xi$ we combine this result with Lemma 3. This gives that for

$$\forall n \geq N \quad d_{\text{var}}(P(\cdot|s,a), \hat{P}_n(\cdot|s,a)) \leq \epsilon \tag{B.16}$$

to hold with probability $1-\delta'-\delta''$, it is sufficient to have

$$N \geq \frac{2}{\xi}\left(\frac{|S|^2}{4\epsilon^2} \log \frac{|S|^2}{\delta'\epsilon} + \log \frac{1}{\delta''}\right). \tag{B.17}$$

Taking a union bound over all state-action pairs $(s,a) \in SA_\xi$ (note $|SA_\xi| \leq |S||\mathcal{A}|$) gives that for Eqn. (B.16) to hold for all $(s,a) \in SA_\xi$ with probability $1-|S||\mathcal{A}|\delta' - |S||\mathcal{A}|\delta''$, it suffices that Eqn. (B.17) is satisfied. Choosing $\delta' = \delta'' = \frac{\delta}{2|S||\mathcal{A}|}$ gives the lemma. □

The above lemma tells us only a polynomial number of sample trajectories under the teacher's policy are necessary, to guarantee that the state-action pairs frequently visited under the teacher's policy are accurate in all models $\{M^{(i)}\}_i$. Now we use the Simulation Lemma to translate this into accurate evaluation of the utility of the teacher's policy in the models $\{M^{(i)}\}_i$.

Lemma 6 (restated). Let any $\alpha, \delta > 0$ be given. Assume we use the algorithm as described in Section 3.4. Let N_T satisfy the following condition

$$N_T \geq \frac{32|S||\mathcal{A}|HR_{\max}}{\alpha}\Big(\log \frac{2|S||\mathcal{A}|}{\delta} + \frac{64|S|^2H^4R_{\max}^2}{\alpha^2} \log \frac{32H^2R_{\max}|S|^3|\mathcal{A}|}{\delta\alpha}\Big),$$

or simplified and less tight (Notice that if $\alpha/8 > HR_{\max}$, the statement is trivially

true with probability 1. So we can simplify using the fact that $\alpha/8 \leq HR_{\max}$.)

$$N_T \geq \frac{4096|S|^3|\mathcal{A}|H^5R_{\max}^3}{\alpha^3} \log \frac{32H^2R_{\max}|S|^3|\mathcal{A}|}{\delta\alpha}$$

Then with probability $1-\delta$ we have that

$$\forall i \quad |U_{\hat{M}^{(i)}}(\pi_T) - U_M(\pi_T)| \leq \alpha/8.$$

Proof. Let

$$\epsilon = \frac{1}{2}\frac{\alpha/8}{H^2R_{\max}}$$

and let

$$\eta = \frac{1}{2}\frac{\alpha/8}{HR_{\max}}.$$

From the Simulation Lemma we have that if there exists a set of state-action pairs $\overline{SA}_\eta \subseteq S \times \mathcal{A}$ such that the following holds

$$\begin{aligned} &\text{(i)} \quad \forall (s,a) \in \overline{SA}_\eta, \quad d_{\text{var}}(P(\cdot|s,a), \hat{P}(\cdot|s,a)) \leq \epsilon, \\ &\text{(ii)} \quad P(\{(s_t,a_t)\}_{t=0}^H \subseteq \overline{SA}_\eta | \pi_T, M) \geq 1-\eta. \end{aligned}$$

Then we have

$$|U_M(\pi_T) - U_{\hat{M}}(\pi_T)| \leq H^2\epsilon R_{\max} + \eta H R_{\max} = \frac{\alpha}{8}.$$

Where the last inequality follows from our choice of ϵ and η above. Now choose $\overline{SA}_\eta = \{(s,a) \in S \times \mathcal{A} : \text{P}((s,a) \text{ is visited in a trial of length } H \text{under } \pi_T) \geq \frac{\eta}{|S||\mathcal{A}|}\}$, then $\overline{SA}_\eta$ satisfies condition (ii). So it remains to show that condition (i) is satisfied. From Lemma 20 we have that for all $(s,a) \in \overline{SA}_\eta$ for

$$d_{\text{var}}(P(\cdot|s,a), \hat{P}_n(s,a)) \leq \epsilon$$

to hold with probability $1-\delta$ it suffices that

$$N \geq \frac{2|S||\mathcal{A}|}{\eta}(\frac{|S|^2}{4\epsilon^2}\log\frac{2|S|^3|\mathcal{A}|}{\delta\epsilon} + \log\frac{2|S||\mathcal{A}|}{\delta}).$$

Filling in the choices of ϵ and η into this condition gives the sufficient condition on N_T as stated in this lemma. □

B.3.3 Proof of Lemma 7

We now give a formal proof of Lemma 7.

Proof of Lemma 7. From Lemma 6 we have that for Eqn. (3.12) to hold with probability $1-\delta'$ it suffices that

$$N_T \geq \frac{4096|S|^3|\mathcal{A}|H^5R_{\max}^3}{\alpha^3}\log\frac{32H^2R_{\max}|S|^3|\mathcal{A}|}{\delta'\alpha}. \tag{B.18}$$

Eqn. (3.14) is trivially true when $\alpha > 16HR_{\max}$. If $\alpha \leq 16HR_{\max}$, the Hoeffding inequality gives us that for Eqn. (3.14) to hold with probability at least $1-\delta'$, it is (more than) sufficient that N_T satisfies Eqn. (B.18).

The Hoeffding inequality also gives us that for Eqn. (3.13) to hold with probability $1-N\delta''$ it suffices that

$$k_1 \geq \frac{16^2H^2R_{\max}^2}{2\alpha^2}\log\frac{2}{\delta''}. \tag{B.19}$$

Now since the algorithm only exits in iteration N, we must have for all $i = 1:N-1$ that

$$\hat{U}_M(\pi^{(i)}) < \hat{U}_M(\pi_T) - \alpha/2. \tag{B.20}$$

Combining Eqn. (B.20), (3.12), (3.13) and (3.14) and the fact that $\pi^{(i)}$ is $\alpha/8$-optimal for $\hat{M}^{(i)}$ we get

$$\forall i\ (1 \leq i < N-1)\ \ U_{\hat{M}^{(i)}}(\pi^{(i)}) \geq U_M(\pi^{(i)}) + \alpha/8. \tag{B.21}$$

So far we have shown that for Eqn. (3.12), (3.13), (3.14) and (B.21) to hold with probability $1-2\delta'-N\delta''$, it suffices that Eqn. (B.18) and Eqn. (B.19) are satisfied.

Now using the contrapositive of the Simulation Lemma and choosing $\epsilon = \frac{1}{2}\frac{\alpha/8}{H^2 R_{\max}}$ we get from Eqn. (B.21) that the policy $\pi^{(i)}$ must be visiting a state-action pair (s, a) that satisfies

$$d_{\text{var}}(P(\cdot|s,a), \hat{P}^{(i)}(\cdot|s,a)) > \frac{\alpha}{16H^2 R_{\max}} \tag{B.22}$$

with probability at least $\frac{\alpha}{16HR_{\max}}$ in every trial of horizon H.

[High-level proof intuition. Eqn. (B.21) states that the models $\hat{M}^{(i)}$ are inaccurate; they overestimate the utility of the policies $\{\pi^{(i)}\}_{i=1:N-1}$. We used the contrapositive of the Simulation Lemma to show this implies that an "inaccurately" modeled state-action pair must be visited with probability $\frac{\alpha}{16HR_{\max}}$ by such a policy $\pi^{(i)}$. This means that data collected under such a policy will improve the model. It remains to show that this can only happen a limited number of times until the model has become a good model.]

Let $\tilde{P}_k(\cdot|s,a)$ be the estimate of $P(\cdot|s,a)$ based upon k observations of the state-action pair (s, a). From Lemma 5 we have that for any state-action pair (s, a) for

$$\forall k > K, \quad d_{\text{var}}(P(\cdot|s,a), \tilde{P}_k(\cdot|s,a)) \leq \frac{\alpha}{16H^2 R_{\max}}$$

to hold with probability $1 - \delta'''$, it suffices that

$$K \geq \frac{16^2 H^4 R_{\max}^2 |S|^2}{4\alpha^2} \log \frac{16H^2 R_{\max}|S|^2}{\alpha\delta'''}.$$

The above equation bounds the number of times a state-action pair can be visited until it is "accurately" modeled with probability $1 - \delta'''$. So with probability $1 - |S||\mathcal{A}|\delta'''$ one can encounter state-action pairs (s, a) that—at the moment of encounter—satisfy Eqn. (B.22) (i.e., that are inaccurately modeled) at most

$$|S||\mathcal{A}| \frac{16^2 H^4 R_{\max}^2 |S|^2}{4\alpha^2} \log \frac{16H^2 R_{\max}|S|^2}{\alpha\delta'''} \tag{B.23}$$

times. Since (from above) such a state-action pair is encountered with probability at least $\frac{\alpha}{16HR_{\max}}$ in each iteration of the algorithm before it exits, Lemma 3 gives that

w.p. $1-\delta''''$ after a number of iterations

$$N_{\text{ubound}} = \frac{32HR_{\max}}{\alpha}\Big(\log\frac{1}{\delta''''} + |S||\mathcal{A}|\frac{16^2H^4R_{\max}^2|S|^2}{4\alpha^2}\log\frac{16H^2R_{\max}|S|^2}{\alpha\delta'''}\Big) \tag{B.24}$$

such a state-action pair has been encountered as many times as stated in Eqn. (B.23). So N_{ubound} is an upperbound on the number of iterations of the algorithm with probability $1-2\delta'-N\delta''-|S||\mathcal{A}|\delta'''-\delta''''$. Choose $\delta',\delta'',\delta''',\delta''''$ such that

$$2\delta' = N_{\text{ubound}}\delta'' = |S||\mathcal{A}|\delta''' = \delta'''' = \frac{1}{4}\delta. \tag{B.25}$$

Substituting these choices into Eqn. (B.24), Eqn. (B.19) and Eqn. (B.18) gives us Eqn. (3.8), Eqn. (3.9) and Eqn. (3.10). □

B.4 Proofs for Section 3.7.1

Proof of Proposition 8. W.l.o.g. assume a coordinate system $x_{1:n}$ such that

$$\forall i \in 2:n, (\mu_1)_i = (\mu_2)_i = 0,$$

and

$$(\mu_1)_1 \leq (\mu_2)_1.$$

Also, let $a = (\mu_1)_1$ and $b = (\mu_2)_1$. Let

$$f(z;\mu) = \frac{1}{(2\pi)^{\frac{n}{2}}\sigma^n}\exp\left(\frac{-\|z-\mu\|_2^2}{2\sigma^2}\right).$$

Then we have that

$$
\begin{aligned}
& d_{\text{var}}(\mathcal{N}(\mu_1,\sigma^2),\mathcal{N}(\mu_2,\sigma^2)) \\
= \;& \frac{1}{2}\int dx_{2:n}\int_{x_1=-\infty}^{x_1=+\infty}\big|f(x;\mu_1)-f(x;\mu_2)\big|dx_1 \\
= \;& \frac{1}{2}\int dx_{2:n}\int_{x_1=-\infty}^{x_1=(a+b)/2}\big(f(x;\mu_1)-f(x;\mu_2)\big)dx_1 \\
& + \frac{1}{2}\int dx_{2:n}\int_{x_1=(a+b)/2}^{x_1=+\infty}\big(f(x;\mu_2)-f(x;\mu_1)\big)dx_1 \\
= \;& \frac{1}{2}\left(1-2\int dx_{2:n}\int_{x_1=(a+b)/2}^{x_1=+\infty}f(x;\mu_1)dx_1\right) \\
& + \frac{1}{2}\left(1-2\int dx_{2:n}\int_{x_1=-\infty}^{x_1=(a+b)/2}f(x;\mu_2)dx_1\right) \\
= \;& 1-2\frac{1}{\sqrt{2\pi}\sigma}\int_{z=|a-b|/2}^{\infty}\exp(\frac{-z^2}{2\sigma^2})dz \\
= \;& \frac{1}{\sqrt{2\pi}\sigma}\int_{z=-|a-b|/2}^{|a-b|/2}\exp(\frac{-z^2}{2\sigma^2})dx_1 \\
\leq \;& \frac{1}{\sqrt{2\pi}\sigma}|a-b| \\
= \;& \frac{1}{\sqrt{2\pi}\sigma}\|\mu_2-\mu_1\|_2.
\end{aligned}
$$

□

In our setting, we have Gaussian noise contributing additively to the next-state given current-state and action. As a consequence, the random variables (and their increments over time) are not bounded. The following proposition shows that we can "essentially" treat Gaussian random variables as bounded random variables with high probability by truncating the tails.

Proposition 21. *Let any $N>0,\delta>0,\sigma>0$ be given. Let $\{w^{(i)}\}_{i=1}^N$ be Gaussian random variables, namely $w^{(i)}\sim\mathcal{N}(0,\sigma^2)$. Then with probability $1-\delta$ we have*

$$\max_{i\in 1:N}|w^{(i)}|\leq\sigma\log\frac{4N}{\sqrt{2\pi}\delta}.$$

Proof.

$$\begin{aligned} \mathrm{P}(|w^{(i)}| \geq K) &= 2\frac{1}{\sqrt{2\pi}\sigma}\int_{z=K}^{\infty}\exp(-\frac{z^2}{2\sigma^2})dz \\ &\leq 2\frac{1}{\sqrt{2\pi}\sigma}\int_{z=K}^{\infty}2\exp(-\frac{z}{\sigma})dz \\ &= 4\frac{1}{\sqrt{2\pi}}\exp(-K/\sigma) \end{aligned}$$

So we have

$$\mathrm{P}(\max_{i\in 1:N}|w^{(i)}| \geq K) \leq N4\frac{1}{\sqrt{2\pi}}\exp(-K/\sigma),$$

which is equivalent to the proposition. □

Note the bound we use for the tail is fairly loose, but it's a simple form and enough for our purposes. Let $K > 0$, here is an another bound (based upon integration by parts):

$$\begin{aligned} \int_{z=K}^{\infty}\frac{1}{\sqrt{2\pi}\sigma}\frac{z}{z}\exp(-\frac{z^2}{2\sigma^2})dz &= \left[\frac{-\sigma}{z\sqrt{2\pi}}\exp(-\frac{z^2}{2\sigma^2})\right]_K^{\infty} \\ &\quad -\int_{z=K}^{\infty}\frac{\sigma}{\sqrt{2\pi}}\frac{1}{z^2}\exp(-\frac{z^2}{2\sigma^2})dz \\ &\leq \frac{\sigma}{\sqrt{2\pi}K}\exp(-\frac{K^2}{2\sigma^2}). \end{aligned}$$

B.5 Proofs for Section 3.7.2

In this section, let $n = n_S + n_{\mathcal{A}}$. In this section we establish several helper lemmas, and then prove Lemma 9. Proofs of helper lemmas are given as sketches or even left out in this section. The proofs are given in the following sections.

We will prove Lemma 9 in the following two steps.

- In Section B.5.1 we establish that for any inaccurate parameter θ, we have that $\mathrm{loss}^{(N_T H)}(\theta) > \mathrm{loss}^{(N_T H)}(\theta^*) + \Omega(N_T)$. I.e., the true parameter θ^* outperforms all inaccurate parameters by a margin of $\Omega(N_T)$.

- In Section B.5.2 we establish that no matter how the additional data $\{z^{(i)}\}_{i=N_TH+1}^{N_TH+k_1NH}$ are chosen, the probability that θ^* ever gets outperformed by an inaccurate parameter θ is exponentially small in the margin $\Omega(N_T)$. As a consequence, a "small" number of samples N_T from the teacher is sufficient to guarantee that throughout all iterations an accurate parameter has the smallest loss.

B.5.1 Model estimated from teacher's data

In this section we establish that for any inaccurate parameter θ, we have that

$$\text{loss}^{(N_TH)}(\theta) > \text{loss}^{(N_TH)}(\theta^*) + \Omega(N_T).$$

I.e., the true parameter θ^* outperforms all inaccurate parameters by a margin of $\Omega(N_T)$. A standard way to prove this is to prove it for one specific inaccurate parameter θ, and then use a cover of θ-space and a union bound to prove it for the whole space. The following lemma shows that it is sufficient to cover only a "small" ball around the origin. More specifically, it shows that with high probability all the solutions $\{\hat{\theta}^{(k)}\}_{k=N_TH}^{N_TH+k_1NH}$ to the regularized linear regression problem lie in a small ball S around the origin. Note that the solutions $\{\hat{\theta}^{(k)}\}_{k=N_TH}^{N_TH+k_1NH}$ to the regularized linear regression problems at steps $k = N_TH$ to $N_TH + k_1NH$ are by definition the only parameters we end up using in the algorithm. So it is sufficient to show $\text{loss}^{(N_TH)}(\theta) > \text{loss}^{(N_TH)}(\theta^*) + \Omega(N_T)$ for all inaccurate θ in the "small ball" S that contains all $\{\hat{\theta}^{(k)}\}_{k=N_TH}^{N_TH+k_1NH}$.

Lemma 22. *Let any $\delta > 0$ be given. Let $\{y^{(i)}, z^{(i)}\}_{i=1}^{N_TH+k_1NH}$ be generated as described in Eqn. (3.20). Let $\{\hat{\theta}^{(k)}\}_k$ be defined as in Eqn. (3.21). Let $\tilde{m} = N_TH + k_1NH$. Then we have with probability $1-\delta$ that $\forall k$ $(1 \le k \le \tilde{m})$*

$$\|\hat{\theta}^{(k)}\|_2 \;\le\; \kappa\sqrt{\tilde{m}}\left(\sqrt{2}\|\theta^*\|_2 + \sigma\log\frac{4\tilde{m}}{\sqrt{2\pi}\delta}\right)$$

and that

$$\max_{i=1:\tilde{m}}|y^{(i)}| \leq \sqrt{2}\|\theta^*\|_2 + \sigma \log \frac{4\tilde{m}}{\sqrt{2\pi}\delta}.$$

Note Lemma 22 would not hold if we used unregularized linear regression. Instead of regularizing with a quadratic penalty, one could regularize by explicitly constraining θ to be norm-bounded. This would directly result in a ball that is sufficient to be covered and thus simplify some of the proofs. However, in practice regularized linear regression with a quadratic penalty (as used in our algorithm) is much more commonly used than linear regression with a norm constraint on θ.

The following lemma establishes $\text{loss}^{(N_T H)}(\theta) > \text{loss}^{(N_T H)}(\theta^*) + \Omega(N_T)$ for all inaccurate $\theta \in S$.

Lemma 23. *Let any $\delta, \epsilon, \eta > 0$ be given. Let $\{y^{(i)}, z^{(i)}\}_{i=1}^{N_T H + k_1 NH}$ be generated as described in Eqn. (3.20). Let $\tilde{m} = N_T H + k_1 NH$. Let $y_{\max} = \max_{i=1:\tilde{m}}|y^{(i)}| \leq \sqrt{2}\|\theta^*\|_2 + \sigma \log \frac{8\tilde{m}}{\sqrt{2\pi}\delta}$ and let $R \leq \kappa\sqrt{\tilde{m}}\left(\sqrt{2}\|\theta^*\|_2 + \sigma \log \frac{8\tilde{m}}{\sqrt{2\pi}\delta}\right)$. Let $S = \{\theta : \|\theta\|_2 \leq R\}$. Then for all $\theta \in S$ that do not satisfy Eqn. (3.22) we have that for $\text{loss}^{(N_T H)}(\theta) - \text{loss}^{(N_T H)}(\theta^*) \geq N_T \epsilon^2 \eta/4$ to hold with probability $1 - \delta/4$, it suffices that*

$$N_T = \Omega(\text{poly}(\frac{1}{\epsilon}, \frac{1}{\eta}, \frac{1}{\delta}, H, \|\theta^*\|_2, n_S, n_{\mathcal{A}}, k_1, N)). \tag{B.26}$$

B.5.2 Influence of data from policies $\{\pi^{(i)}\}_i$

In this section we study the power of an adversary to favor one specific θ over θ^* by choosing the samples $\{z^{(i)}\}_{i=N_T H+1}^{N_T H + k_1 NH}$. More specifically the following lemma shows that the adversary's power is very limited. I.e., no matter what policy the adversary uses, the probability of ever making θ outperform θ^* by a margin $a \geq 0$ on this set of adversarially chosen samples is bounded by $\exp(-a/\sigma^2)$. Let

$$\text{loss}_{\text{adv}}^{(k)}(\theta) = \sum_{i=m+1}^{k} (y^{(i)} - \theta^\top z^{(i)})^2.$$

By convention let $\text{loss}_{\text{adv}}^{(k)} = 0$ for $k \leq m$.

Lemma 24. *Let any $a \geq 0$ be given. Let any $\theta \in \mathbb{R}^n$ be given. Let all else be as defined above. Then we have*

$$\mathrm{P}(\exists k > m : \mathrm{loss}_{\mathrm{adv}}^{(k)}(\theta) \leq \mathrm{loss}_{\mathrm{adv}}^{(k)}(\theta^*) - a) \leq \exp(\frac{-a}{2\sigma^2}).$$

The following lemma uses a covering argument to extend Lemma 24 to hold for the set $\{\theta : \|\theta\|_2 \leq R\}$.

Lemma 25. *Let any $\delta, \epsilon, \eta > 0$ be given. Let $\{y^{(i)}, z^{(i)}\}_{i=1}^{N_T H + k_1 NH}$ be generated as described in Eqn. (3.20). Let $\tilde{m} = N_T H + k_1 NH$. Let*

$$R = \kappa\sqrt{\tilde{m}}\left(\sqrt{2}\|\theta^*\|_2 + \sigma \log \frac{8\tilde{m}}{\sqrt{2\pi}\delta}\right).$$

Let $S = \{\theta : \|\theta\|_2 \leq R\}$. Then for all $\theta \in S$ that do not satisfy Eqn. (3.22) we have that for

$$\mathrm{loss}_{\mathrm{adv}}^{(k)}(\theta) - \mathrm{loss}_{\mathrm{adv}}^{(k)}(\theta^*) \geq -N_T \epsilon^2 \eta/8 > 0$$

to hold with probability $1 - \frac{\delta}{4}$ for all $k \in N_T H : k_1 NH$, it suffices that

$$N_T = \Omega\left(\mathrm{poly}(\frac{1}{\epsilon}, \frac{1}{\eta}, \frac{1}{\delta}, H, \|\theta^*\|_2, n_S, n_{\mathcal{A}}, k_1, N)\right). \tag{B.27}$$

B.5.3 Proof of Lemma 9

Proof of Lemma 9. From Lemma 22 we have with probability $1 - \frac{\delta}{2}$ that all the estimates $\{\hat{\theta}^{(k)}\}_{k=N_T H+1}^{N_T H + k_1 NH}$ lie in a bounded sphere

$$S = \{\theta : \|\theta\|_2 \leq \kappa\sqrt{\tilde{m}}\left(\sqrt{2}\|\theta^*\|_2 + \sigma \log \frac{8\tilde{m}}{\sqrt{2\pi}\delta}\right)\}$$

around the origin and that

$$\max_{i=1:\tilde{m}} |y^{(i)}| \leq \sqrt{2}\|\theta^*\|_2 + \sigma \log \frac{8\tilde{m}}{\sqrt{2\pi}\delta}.$$

Lemma 23 gives us that for all θ in this sphere S that do not satisfy the accuracy condition of Eqn. (3.22) we have that

$$\text{loss}^{(N_T H)}(\theta) - \text{loss}^{(N_T H)}(\theta^*) \geq N_T \epsilon^2 \eta / 4 \tag{B.28}$$

holds with probability $1 - \frac{\delta}{4}$ for large enough N_T as quantified in Eqn. (B.26), which corresponds to the condition on N_T in the lemma we are proving. From Lemma 25 we have that

$$\text{loss}_{\text{adv}}^{(k)}(\theta) - \text{loss}_{\text{adv}}^{(k)}(\theta^*) \geq -N_T \epsilon^2 \eta / 8 \tag{B.29}$$

holds with probability $1 - \frac{\delta}{4}$ under the same condition on N_T. Now since for any $k \geq N_T H$ we have by definition $\text{loss}^{(k)}(\theta) = \text{loss}^{(N_T H)}(\theta) + \text{loss}_{\text{adv}}^{(k)}(\theta)$. We can combine Eqn. (B.28) and Eqn. (B.29) to obtain that

$$\text{loss}^{(k)}(\theta) \geq \text{loss}^{(k)}(\theta^*) + N_T \epsilon^2 \eta / 8 \tag{B.30}$$

holds with probability $1 - \delta$. This means that all $\theta \in S$ that do not satisfy the accuracy condition of Eqn. (3.22) are outperformed by the true parameter θ^* by a margin of at least $N_T \epsilon^2 \eta / 8$ for all $k \geq N_T H$. As a consequence all the θ estimates $\{\hat{\theta}^{(k)}\}_{k=N_T H+1}^{N_T H + k_1 N H}$ must satisfy the accuracy condition of Eqn. (3.22). □

B.5.4 Proof of Lemma 22

Proof of Lemma 22. Let $y_{\max} = \max_{i=1:\tilde{m}} |y^{(i)}|$. For any $k : 1 \leq k \leq \tilde{m}$ we have $\text{loss}^{(k)}(\vec{0}) = \sum_{i=1}^{k} (y^{(i)})^2 \leq \tilde{m}(y_{\max})^2$. Since $\hat{\theta}^{(k)}$ achieves the minimal loss, we must have $\text{loss}^{(k)}(\hat{\theta}^{(k)}) \leq \tilde{m}(y_{\max})^2$, and thus

$$\frac{\|\hat{\theta}^{(k)}\|_2^2}{\kappa^2} \leq \tilde{m}(y_{\max})^2. \tag{B.31}$$

We also have (using $\|z^{(i)}\|_2 \leq \sqrt{2}$ and Proposition 21) that

$$\begin{aligned} y_{\max} &\leq \sqrt{2}\|\theta^*\|_2 + \max_{i=1:\tilde{m}}|w^{(i)}| \\ &\leq \sqrt{2}\|\theta^*\|_2 + \sigma \log \frac{8\tilde{m}}{\sqrt{2\pi}\delta} \text{w.p. } 1 - \frac{\delta}{2}, \end{aligned}$$

which combined with Eqn. (B.31) proves the lemma. □

B.5.5 Proof of Lemma 23

Let

$$\begin{aligned} \Delta^{(k)}(\theta_1, \theta_2) &= \frac{1}{\kappa^2}\left|\|\theta_1\|_2^2 - \|\theta_2\|_2^2\right| \\ &+ \max_{\{z^{(i)}, y^{(i)}\}_i} \sum_{i=1}^{k} \left|(y^{(i)} - \theta_1^\top z^{(i)})^2 - (y^{(i)} - \theta_2^\top z^{(i)})^2\right|. \end{aligned}$$

We first prove the following lemma.

Lemma 26. *Let any $\lambda > 0, \tilde{m} > 0$ be given. Let $y_{\max} = \max_{i=1:\tilde{m}}|y^{(i)}|$. There is a subset $\tilde{S}$ of the sphere $S = \{\theta : \|\theta\|_2 \leq R\}$ such that for all $\theta \in S$, there exists $\tilde{\theta} \in \tilde{S}$ such that for all $k(1 \leq k \leq \tilde{m})$ we have:*

$$|\text{loss}^{(k)}(\theta) - \text{loss}^{(k)}(\tilde{\theta})| \leq \Delta^{(k)}(\theta, \tilde{\theta}) \leq \lambda \tag{B.32}$$

And we have the following bound on the number of points in $\tilde{S}$

$$|\tilde{S}| \leq \left(\frac{2R\sqrt{n}(4\tilde{m}y_{\max} + (8\tilde{m} + 2\frac{1}{\kappa^2})R)}{\lambda}\right)^n.$$

In the special case where $R \leq \kappa\sqrt{\tilde{m}}(\sqrt{2}\|\theta^*\|_2 + \sigma \log \frac{8\tilde{m}}{\sqrt{2\pi}\delta})$ and $y_{\max} \leq \sqrt{2}\|\theta^*\|_2 + \sigma \log \frac{8\tilde{m}}{\sqrt{2\pi}\delta}$ we get

$$\log|\tilde{S}| = O\left(n \log \text{poly}(\frac{1}{\lambda}, \frac{1}{\delta}, \tilde{m}, \|\theta^*\|_2, n)\right).$$

Proof. Note that the first inequality in Eqn. (B.32) trivially holds by definition of Δ.

We now prove the second inequality. Consider the difference in loss for two weight vectors θ_1, θ_2. The contribution of one training sample is bounded by

$$\begin{aligned}
&\sup_{z:\|z\|\le\sqrt{2},y:|y|\le y_{\max}} \left|(y-\theta_1^T z)^2-(y-\theta_2^T z)^2\right| \\
&= \sup_{z:\|z\|\le\sqrt{2},y:|y|\le y_{\max}} \left|(y-\theta_1^T z)^2-(y-\theta_1^T z+(\theta_1-\theta_2)^T z)^2\right| \\
&= \sup_{z:\|z\|\le\sqrt{2},y:|y|\le y_{\max}} \left|-((\theta_1-\theta_2)^T z)^2-2(y-\theta_1^T z)(\theta_1-\theta_2)^T z\right| \\
&\le 2\|\theta_1-\theta_2\|_2^2+2\sqrt{2}(y_{\max}+\sqrt{2}R)\|\theta_1-\theta_2\|_2 \\
&\le 4R\|\theta_1-\theta_2\|_2+2\sqrt{2}(y_{\max}+\sqrt{2}R)\|\theta_1-\theta_2\|_2 \\
&\le (4y_{\max}+8R)\|\theta_1-\theta_2\|_2.
\end{aligned}$$

So we have

$$\begin{aligned}
&\Delta^{(k)}(\theta_1,\theta_2) \\
&\le k(4y_{\max}+8R)\|\theta_1-\theta_2\|_2+\frac{1}{\kappa^2}\left|\|\theta_1\|_2^2-\|\theta_2\|_2^2\right| \\
&= k(4y_{\max}+8R)\|\theta_1-\theta_2\|_2+\frac{1}{\kappa^2}\left|(\theta_1+\theta_2)^\top(\theta_1-\theta_2)\right| \\
&\le k(4y_{\max}+8R)\|\theta_1-\theta_2\|_2+\frac{1}{\kappa^2}2R\|\theta_1-\theta_2\|_2 \\
&\le (4ky_{\max}+(8k+2\frac{1}{\kappa^2})R)\|\theta_1-\theta_2\|_2.
\end{aligned}$$

To cover a sphere of radius R up to $\|\cdot\|_2 \le \gamma$, it is sufficient to have (cover the enclosing cube with side of $2R$ regularly) $(2R\sqrt{n}/\gamma)^n$ points. In our case we want to cover the set of considered θ's up to loss accuracy λ, so we have $\gamma = \frac{\lambda}{4ky_{\max}+(8k+2\frac{1}{\kappa^2})R}$, resulting in a number of points

$$\left(\frac{2R\sqrt{n}(4ky_{\max}+(8k+2\frac{1}{\kappa^2})R)}{\lambda}\right)^n.$$

This establishes the lemma for one specific k. It is easily seen that the cover used for $k=\tilde{m}$ can be used for all $k \le \tilde{m}$. This proves the theorem. □

We will use $\tilde{S}(n, \frac{1}{\lambda}, \frac{1}{\delta}, \tilde{m}, \|\theta^*\|_2)$ if we want to explicitly show the dependence on

the parameters.

Proof of Lemma 23. Let $\theta \in \mathbb{R}^n$. Let $e_\theta^{(i)} = \theta^{*\top} z^{(i)} - \theta^\top z^{(i)}$, and let $\tilde{e}_\theta^{(i)}$ be $e_\theta^{(i)}$ clipped to the interval $[-K_w, K_w]$. Let K_w be such that $\max_{i \in 1:N_T H} |w^{(i)}| \leq K_w$. Then we have that

$$\begin{aligned} \text{loss}^{(N_T H)}(\theta) &= \sum_{t=1}^{N_T H} (w^{(i)} + e_\theta^{(i)})^2 + \frac{1}{\kappa^2}\|\theta\|_2^2 \\ &\geq \sum_{t=1}^{N_T H} (w^{(i)} + \tilde{e}_\theta^{(i)})^2 + \frac{1}{\kappa^2}\|\theta\|_2^2. \end{aligned}$$

Now consider the difference in loss for θ and the optimal θ^*:

$$\begin{aligned} &\text{loss}^{(N_T H)}(\theta) - \text{loss}^{(N_T H)}(\theta^*) \\ &= \sum_{i=1}^{N_T H} (w^{(i)} + e_\theta^{(i)})^2 + \frac{1}{\kappa^2}\|\theta\|_2^2 - \sum_{i=1}^{N_T H} (w^{(i)})^2 - \frac{1}{\kappa^2}\|\theta^*\|_2^2 \\ &\geq \sum_{i=1}^{N_T H} (w^{(i)} + \tilde{e}_\theta^{(i)})^2 + \frac{1}{\kappa^2}\|\theta\|_2^2 - \sum_{i=1}^{N_T H} (w^{(i)})^2 - \frac{1}{\kappa^2}\|\theta^*\|_2^2 \\ &= \sum_{i=1}^{N_T H} (\tilde{e}_\theta^{(i)})^2 + 2w^{(i)}\tilde{e}_\theta^{(i)} + \frac{1}{\kappa^2}(\|\theta\|_2^2 - \|\theta^*\|_2^2) \end{aligned} \tag{B.33}$$

Let $Z_t = \sum_{i=1}^t (\tilde{e}_\theta^{(i)})^2 + 2w^{(i)}\tilde{e}_\theta^{(i)} - \text{E}(\tilde{e}_\theta^{(i)})^2$. Note that $\forall t, |Z_t - Z_{t-1}| \leq 4K_w^2$. Then applying Azuma's inequality to the martingale $\{Z_t\}_t$ gives us that

$$\sum_{i=1}^{N_T H} (\tilde{e}_\theta^{(i)})^2 + 2w^{(i)}\tilde{e}_\theta^{(i)} - \text{E}(\tilde{e}_\theta^{(i)})^2 \geq -\lambda \tag{B.34}$$

holds with probability $1 - \exp(-\lambda^2/(2N_T H 4K_w^2))$. Combining Eqn. (B.33) and (B.34) gives that

$$\begin{aligned} &\text{loss}^{(N_T H)}(\theta) - \text{loss}^{(N_T H)}(\theta^*) \geq \\ &\quad \sum_{i=1}^{N_T H} \text{E}(\tilde{e}_\theta^{(i)})^2 - \lambda + \frac{1}{\kappa^2}(\|\theta\|_2^2 - \|\theta^*\|_2^2) \end{aligned}$$

holds with probability $1-\exp(-\lambda^2/(8N_THK_w^2))$. Now using Lemma 26 we get that with probability $1-|\tilde{S}|\exp(-\lambda^2/(8N_THK_w^2))$ the following holds for all $\theta(\|\theta\|_2 \leq R) : \exists\tilde{\theta}\in\tilde{S}$ s.t. $\Delta^{(N_TH)}(\theta,\tilde{\theta})\leq\lambda$ and

$$\begin{aligned}
&\text{loss}^{(N_TH)}(\theta)-\text{loss}^{(N_TH)}(\theta^*)\\
&= \text{loss}^{(N_TH)}(\theta)-\text{loss}^{(N_TH)}(\tilde{\theta})\\
&\quad +\text{loss}^{(N_TH)}(\tilde{\theta})-\text{loss}^{(N_TH)}(\theta^*)\\
&\geq -\lambda+\text{loss}^{(N_TH)}(\tilde{\theta})-\text{loss}^{(N_TH)}(\theta^*)\\
&\geq \sum_{i=1}^{N_TH}\text{E}(\tilde{e}_{\tilde{\theta}}^{(i)})^2-2\lambda+\frac{1}{\kappa^2}(\|\tilde{\theta}\|_2^2-\|\theta^*\|_2^2)\\
&= \sum_{i=1}^{N_TH}\text{E}(\tilde{e}_{\theta}^{(i)})^2-2\lambda+\frac{1}{\kappa^2}(\|\theta\|_2^2-\|\theta^*\|_2^2)\\
&+\sum_{i=1}^{N_TH}\text{E}(\tilde{e}_{\tilde{\theta}}^{(i)})^2-\text{E}(\tilde{e}_{\theta}^{(i)})^2+\frac{1}{\kappa^2}(\|\tilde{\theta}\|_2^2-\|\theta\|_2^2). \qquad \text{(B.35)}
\end{aligned}$$

We also have that

$$
\begin{aligned}
&\Big| \sum_{i=1}^{N_T H} \mathrm{E}[(\tilde{e}_{\tilde{\theta}}^{(i)})^2] - \mathrm{E}[(\tilde{e}_{\theta}^{(i)})^2] \Big| \\
\leq \;& \sum_{i=1}^{N_T H} \Big| \mathrm{E}[(\tilde{e}_{\tilde{\theta}}^{(i)})^2] - \mathrm{E}[(\tilde{e}_{\theta}^{(i)})^2] \Big| \\
\leq \;& \sum_{i=1}^{N_T H} \Big| \mathrm{E}[(e_{\tilde{\theta}}^{(i)})^2] - \mathrm{E}[(e_{\theta}^{(i)})^2] \Big| \\
= \;& \sum_{i=1}^{N_T H} \Big| \mathrm{E}[(\theta^{*\top} z^{(i)} - \tilde{\theta}^{\top} z^{(i)})^2] \\
& - \mathrm{E}[(\theta^{*\top} z^{(i)} - \theta^{\top} z^{(i)})^2] \Big| \\
= \;& \sum_{i=1}^{N_T H} \Big| \mathrm{E}[(\theta^{*\top} z^{(i)} + w^{(i)} - \tilde{\theta}^{\top} z^{(i)})^2] \\
& - \mathrm{E}[(\theta^{*\top} z^{(i)} + w^{(i)} - \theta^{\top} z^{(i)})^2] \Big| \\
\leq \;& \sum_{i=1}^{N_T H} \max_{z^{(i)}, y^{(i)}} \Big| [(y^{(i)} - \tilde{\theta}^{\top} z^{(i)})^2] \\
& - [(y^{(i)} - \theta^{\top} z^{(i)})^2] \Big|.
\end{aligned}
$$

The second inequality uses the fact that for any $z^{(i)}$, we have $|(\tilde{e}_{\tilde{\theta}}^{(i)})^2 - (\tilde{e}_{\theta}^{(i)})^2| \leq |(e_{\tilde{\theta}}^{(i)})^2 - (e_{\theta}^{(i)})^2|$.

And thus we have

$$
\begin{aligned}
&\Big| \sum_{i=1}^{N_T H} \mathrm{E}(\tilde{e}_{\tilde{\theta}}^{(i)})^2 - \mathrm{E}(\tilde{e}_{\theta}^{(i)})^2 \Big| + \frac{1}{\kappa^2} \Big| \|\tilde{\theta}\|_2^2 - \|\theta\|_2^2 \Big| \\
&\quad \leq \; \Delta^{(N_T H)}(\theta, \tilde{\theta}) \\
&\quad \leq \; \lambda. \qquad \text{(B.36)}
\end{aligned}
$$

Combining Eqn. (B.35) and (B.36) gives us

$$\begin{aligned}&\text{loss}^{(N_T H)}(\theta) - \text{loss}^{(N_T H)}(\theta^*) \\ &\geq \sum_{i=1}^{N_T H} \mathrm{E}(\tilde{e}_\theta^{(i)})^2 - 3\lambda + \frac{1}{\kappa^2}(\|\theta\|_2^2 - \|\theta^*\|_2^2).\end{aligned}$$

Now for any $\epsilon > 0, \eta > 0$, if θ satisfies

$$P(\max_{i \in 1:H}(e_\theta^{(i)}) > \epsilon) > \eta \tag{B.37}$$

(note this corresponds to θ not satisfying Eqn. (3.22)) then we have that (let $\bar{\epsilon} = \min\{K_w, \epsilon\}$)

$$\begin{aligned}&\text{loss}^{(N_T H)}(\theta) - \text{loss}^{(N_T H)}(\theta^*) \\ &\geq N_T \bar{\epsilon}^2 \eta - 3\lambda + \frac{1}{\kappa^2}(\|\theta\|_2^2 - \|\theta^*\|_2^2)\end{aligned}$$

holds w.p. $1 - |\tilde{S}| \exp(-\lambda^2/(8 N_T H K_w^2))$.

Now choosing $\lambda = \frac{N_T \bar{\epsilon}^2 \eta}{6}$ gives us that

$$\begin{aligned}&\text{loss}^{(N_T H)}(\theta) - \text{loss}^{(N_T H)}(\theta^*) \\ &\geq \frac{N_T \bar{\epsilon}^2 \eta}{2} + \frac{1}{\kappa^2}(\|\theta\|_2^2 - \|\theta^*\|_2^2).\end{aligned}$$

So if

$$N_T \geq \frac{4}{\bar{\epsilon}^2 \eta \kappa^2} \|\theta^*\|_2^2 \tag{B.38}$$

then we have that

$$\text{loss}^{(N_T H)}(\theta) - \text{loss}^{(N_T H)}(\theta^*) \geq \frac{N_T \bar{\epsilon}^2 \eta}{4}$$

holds w.p. $1 - |\tilde{S}| \exp(-(N_T \bar{\epsilon}^4 \eta^2/(288 H K_w^2))$.

We have from Prop. 21 that $\max_{i \in 1:N_T H} w^{(i)} \leq \sigma \log \frac{4 N_T H}{\sqrt{2\pi}\delta'}$ with probability $1 - \delta'$. So we can choose $K_w = \sigma \log \frac{4 N_T H}{\sqrt{2\pi}\delta'}$ (and add in a failure probability of δ'). Making the dependencies in $\tilde{S}$ (and recall we chose $\lambda = \frac{N_T \bar{\epsilon}^2 \eta}{6}$) explicit, we have here

$\tilde{S}(n, \frac{6}{N_T\bar{\epsilon}^2\eta}, \frac{1}{\delta}, \tilde{m}, \|\theta^*\|_2)$ and thus

$$\log|\tilde{S}| = O\left(n \log \text{poly}(n, \frac{1}{N_T\bar{\epsilon}^2\eta}, \frac{1}{\delta'}, \tilde{m}, \|\theta^*\|_2)\right),$$

Now we choose

$$\begin{aligned}\frac{\delta}{8} &= \delta' \\ &= |\tilde{S}| \exp(-(N_T\bar{\epsilon}^4\eta^2/(288H\sigma^2 \log^2 \frac{32N_TH}{\sqrt{2\pi}\delta})).\end{aligned}$$

This gives us the following conditions on N_T.

$$\begin{aligned}&\text{(i)} \quad \text{Eqn. (B.38)}, \\ &\text{(ii)} \quad N_T \geq \frac{288H\sigma^2 \log^2 \frac{32N_TH}{\sqrt{2\pi}\delta}}{\bar{\epsilon}^4\eta^2} \log \frac{|\tilde{S}|}{\delta/8}.\end{aligned}$$

Recall $\bar{\epsilon} = \min\{K_w, \epsilon\}$. So we can replace conditions (i) and (ii) by the following conditions on N_T (recall $K_w = \sigma \log \frac{4N_TH}{\sqrt{2\pi}\delta/8}$):

$$\begin{aligned}&\text{(ia)} \quad N_T \geq \frac{4}{\epsilon^2\eta\kappa^2}\|\theta^*\|_2^2 \\ &\text{(ib)} \quad N_T \geq \frac{4}{(\sigma \log \frac{4N_TH}{\sqrt{2\pi}\delta/8})^2\eta\kappa^2}\|\theta^*\|_2^2 \\ &\text{(iia)} \quad N_T \geq \frac{288H\sigma^2 \log^2 \frac{32N_TH}{\sqrt{2\pi}\delta}}{\epsilon^4\eta^2} \log \frac{|\tilde{S}|}{\delta/8}, \\ &\text{(iib)} \quad N_T \geq \frac{288H\sigma^2 \log^2 \frac{32N_TH}{\sqrt{2\pi}\delta}}{(\sigma \log \frac{4N_TH}{\sqrt{2\pi}\delta/8})^4\eta^2} \log \frac{|\tilde{S}|}{\delta/8}.\end{aligned}$$

Combining the four conditions on N_T with the expression for $|\tilde{S}|$ gives us the following condition on N_T suffices:

$$N_T = \Omega(\text{poly}(\frac{1}{\epsilon}, \frac{1}{\eta}, \frac{1}{\delta}, H, \|\theta^*\|_2, n_S, n_{\mathcal{A}}, k_1, \log N)).$$

This proves the lemma. Note that in the statement of the lemma we have slightly weaker result, namely a polynomial dependence on N rather than a polynomial dependence on $\log N$, which we proved here. □

B.5.6 Proofs of Lemmas 24 and 25

We first prove the following lemma about a (possibly adversarial) biased random walk. We refer the reader to, e.g., (Durrett, 1995; Billingsley, 1995; Williams, 1991), for more details on martingales and stopping times.

Lemma 27. *Let $\{w^{(i)}\}_{i=1}^{\infty}$ be IID random variables with $w^{(i)} \sim \mathcal{N}(0, \tau^2)$. Let $\mathcal{F}_n = \sigma(w^{(1)}, \ldots, w^{(n)})$, the sigma algebra induced by these random variables. Let $\forall i, e^{(i)} \in \mathcal{F}_{i-1}$, i.e., $e^{(i)}$ has to be chosen based upon the past. Let $\forall n$,*

$$Z_n = \sum_{i=1}^{n} (e^{(i)})^2 + 2w^{(i)}e^{(i)}.$$

Let any $a > 0$ be given. Let $T_a = \inf\{n : Z_n \leq -a\}$.

Then we have

$$P(T_a < \infty) \leq \exp(\frac{-a}{2\sigma^2}).$$

Proof. Let the martingale sequence $\{Y_n\}_n$ over the filtration $\{\mathcal{F}_n\}_n$ be defined as follows

$$\begin{aligned} Y_n &= \exp\left(\frac{-1}{2\sigma^2} Z_n\right) \\ &= \exp\left(\frac{-1}{2\sigma^2} \sum_{i=1}^{n} (2w^{(i)}e^{(i)} + (e^{(i)})^2)\right). \end{aligned}$$

It is easily verified that $\{Y_n\}_n$ is adapted to $\{\mathcal{F}_n\}$, that $E|Y_n| < \infty$ and that $E(Y_{n+1}|\mathcal{F}_n) = Y_n$ for all n. Thus Y_n is indeed a martingale with respect to $\{\mathcal{F}_n\}_n$. (Note this is true no matter what the adversary's policy is for choosing the $\mathcal{F}_{i-1}$-measurable functions $e^{(i)}$.)

Let any integer $K > 0$ be fixed. Let

$$\begin{aligned} T_b &= \inf\{n : Z_n \geq b\}, \\ N &= \min\{T_a, T_b, K\}. \end{aligned}$$

Then N is a finite stopping time. Thus we can apply the Optional Stopping Theorem[2] and get

$$\begin{aligned} 1 &= \mathrm{E}Y_0 = \mathrm{E}Y_N \\ &= P(T_a < T_b, T_a < K)\mathrm{E}[Y_N | T_a < T_b, T_a < K] \\ &\quad + P(T_b < T_a, T_b < K)\mathrm{E}[Y_N | T_b < T_a, T_b < K] \\ &\quad + P(K \leq T_b, K \leq T_a)\mathrm{E}[Y_N | K < T_b, K < T_a]. \end{aligned}$$

Now since $\forall n, Y_n \geq 0$ we have

$$1 \geq P(T_a < T_b, T_a < K)\mathrm{E}[Y_N | T_a < T_b, T_a < K].$$

Using $\mathrm{E}[Y_N | T_a < T_b, T_a < K] \geq \exp(\frac{a}{2\sigma^2})$ we get

$$P(T_a < T_b, T_a < K) \leq \exp(\frac{-a}{2\sigma^2}).$$

Taking the limit for $K \to \infty$ (and using the monotone convergence theorem which allows us to interchange limit and expectation (probability)) gives us

$$P(T_a < T_b, T_a < \infty) \leq \exp(\frac{-a}{2\sigma^2}).$$

Since $b > 0$ was arbitrary, we get for $b \to \infty$ (and using the monotone convergence theorem which allows us to interchange limit and expectation (probability)) that

$$P(T_a < \infty) \leq \exp(\frac{-a}{2\sigma^2}).$$

[2]See, e.g., Durrett, 1995.

□

Proof of Lemma 24. We have $y^{(i)} = \theta^{*\top} z^{(i)} + w^{(i)}$, where $w^{(i)} \sim \mathcal{N}(0, \sigma^2)$. Let $e^{(i)} = \theta^{*\top} z^{(i)} - \theta^\top z^{(i)}$. Then

$$\begin{aligned}
&{\text{loss}_{\text{adv}}}^{(k)}(\theta) - {\text{loss}_{\text{adv}}}^{(k)}(\theta^*) \\
&= \sum_{i=m+1}^{k} (\theta^{*\top} z^{(i)} + w^{(i)} - \theta^\top z^{(i)})^2 - (w^{(i)})^2 \\
&= \sum_{i=m+1}^{k} (w^{(i)} + e^{(i)})^2 - (w^{(i)})^2 \\
&= \sum_{i=m+1}^{k} (e^{(i)})^2 + 2 w^{(i)} e^{(i)}.
\end{aligned}$$

So we can apply Lemma 27 with $Z_n = {\text{loss}_{\text{adv}}}^{(n+m)}(\theta) - {\text{loss}_{\text{adv}}}^{(n+m)}(\theta^*)$, which proves the lemma. □

Proof of Lemma 25. Using a $\lambda = N_T \epsilon^2 \eta / 16$ cover $\tilde{S}$ for S and Lemma 24 gives us that $\forall k \geq 1$ and for all $\theta \in S$ that

$$\text{loss}_{\text{adv}}^{(k)}(\theta) - \text{loss}_{\text{adv}}^{(k)}(\theta^*) \geq -\lambda - N_T \epsilon^2 \eta / 16 = -N_T \epsilon^2 \eta / 8$$

holds w.p.

$$1 - |\tilde{S}| \exp(-N_T \epsilon^2 \eta / (32 \sigma^2)).$$

The last term corresponds to the probability of the biased random walk reaching $-N_T \epsilon^2 \eta / 16$. Now requiring that the last term is smaller than $\frac{\delta}{4}$ gives us the following requirement:

$$N_T \geq \frac{32 \sigma^2}{\epsilon^2 \eta} \log \frac{|\tilde{S}(n, \frac{16}{N_T \epsilon^2 \eta}, \frac{2}{\delta}, \tilde{m}, \|\theta^*\|_2)|}{\delta / 4},$$

which is satisfied when N_T satisfies Eqn. (B.27). □

B.5.7 Proof of Theorem 10

Proof of Theorem 10. From Lemma 9 we have that for a trial under the teacher's policy $(\{(x_t, u_t)\}_{t=1}^{H})$ for

$$P\Big(\max_{t\in 1:H}\|A^{(i)}\phi(x_t) + B^{(i)}u_t - (A\phi(x_t) + Bu_t)\|_2 > \epsilon\Big) \leq \eta \tag{B.39}$$

to hold with probability $1-\delta$ for all $i \in 1:N$, it suffices that

$$N_T = \Omega\Big(\text{poly}(\frac{1}{\epsilon}, \frac{1}{\eta}, \frac{1}{\delta}, H, \|A\|_{\text{F}}, \|B\|_{\text{F}}, n_S, n_{\mathcal{A}}, k_1, N)\Big). \tag{B.40}$$

From Prop. 8 and Eqn. (B.39) we have that for

$$P\ \Big(\max_{t\in 1:H} d_{\text{var}}(P(\cdot\,|x_t, u_t), P^{(i)}(\cdot\,|x_t, u_t)) > \frac{1}{\sqrt{2\pi}\sigma}\epsilon\Big) \leq \eta \tag{B.41}$$

to hold for all $i \in 1:N$ with probability $1-\delta$ it is sufficient that N_T satisfies Eqn. (B.40). Let $\overline{SA}_\eta = \{(x,u) : \forall i \;\; d_{\text{var}}(P(\cdot\,|x,u), P^{(i)}(\cdot\,|x,u)) \leq \frac{\epsilon}{\sqrt{2\pi}\sigma}\}$. Then using the Simulation Lemma combined with Eqn. (B.41) we obtain that that for

$$|U_M(\pi_T) - U_{M^{(i)}}(\pi_T)| \leq H^2\frac{1}{\sqrt{2\pi}\sigma}\epsilon R_{\max} + \eta H R_{\max}$$

to hold for all $i \in 1:N$ with probability $1-\delta$, it suffices that N_T satisfies Eqn. (B.40). Choosing $\epsilon = \frac{\sqrt{2\pi}\sigma}{2H^2 R_{\max}}\alpha$ and $\eta = \frac{1}{2HR_{\max}}\alpha$ proves the theorem. □

B.6 More detailed version of Section 3.7.3

B.6.1 A result for Bayesian model averaging

Consider an adversary generating a data sequence $\{z^{(t)}\}_{t=1}^T$. For every time step t, $w^{(t)} \sim \mathcal{N}(0, \sigma^2)$, and $y^{(t)} = \theta^{*\top} z^{(t)}$. We assume $\|z^{(t)}\|_2^2 \leq 2$.[3] Now define a sequence $\{\hat{\theta}^{(t)}\}_{t=1}^T$ of estimates of θ^*

$$\begin{aligned} \hat{\theta}^{(t)} &= \arg\min_{\theta \in \mathbb{R}^n} \sum_{i=1}^{t} (y^{(i)} - \theta^\top z^{(i)})^2 + \frac{1}{\kappa^2}\|\theta\|_2^2 \\ &= \arg\min_{\theta \in \mathbb{R}^n} \sum_{i=1}^{t} \frac{1}{\sigma^2}(y^{(i)} - \theta^\top z^{(i)})^2 + \frac{1}{\nu^2}\|\theta\|_2^2, \end{aligned}$$

here $\nu^2 = \kappa^2\sigma^2$. Let $e^{(t)} = \theta^{*\top} z^{(t)} - \hat{\theta}^{(t)\top} z^{(t)}$. In this section, we will prove the following theorem

Theorem 28. *Let everything be as above defined. Then no matter how the adversary chooses each $z^{(t)}$ (possibly based on everything seen up to time $t-1$), we have that with probability $1-\delta$*

$$\begin{aligned} N_\mu &= \sum_{t=1}^{T} \mathbf{1}\{e^{(t)} \geq \mu\} \\ &\leq O(\sqrt{T}(\log T)^3 \mathrm{poly}(\|\theta^*\|_2, n, \log\frac{1}{\delta}, \frac{1}{\mu})). \end{aligned}$$

This result will be obtained in the following three steps:

- prove an online log-loss bound for Bayesian model averaging (BMA),
- prove a bound on the variances s_t^2 used at every step in the BMA algorithm, in particular prove a bound on how often these variances can be 'large',
- prove a bound on the squared loss incurred for the time-steps when the variances s_t^2 are 'small'.

[3] Since $z^{(t)}$ later corresponds to the concatenation of $\phi(x^{(t)})$ and $u^{(t)}$, which both have norm smaller than one, this is the right choice. Kakade and Ng (2005) use $\|z^{(t)}\|_2 \leq 1$, which makes their results slightly different from the way we state their results in this paper.

We now consider the Bayesian model averaging (BMA) algorithm, and give a bound on its worst-case online loss. In particular we consider the case of linear least squares regression. We have

$$p(y|z,\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(\theta^T z - y)^2}{2\sigma^2}\right), \tag{B.42}$$

where σ^2 is a fixed, *known* constant that is not a parameter of our model. Note that Kakade and Ng (2005) give results for generalized linear models. This section reviews a subset of their results, where notation is specialized to the linear regression case.

Let $S = \{(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \ldots, (z^{(T)}, y^{(T)})\}$ be an arbitrary sequence of examples, possibly chosen by an adversary. We also use S_t to denote the subsequence consisting of only the first t examples. Unless otherwise stated, we will assume throughout this section that $||z^{(i)}||_2 \leq \sqrt{2}$ for all i.

Assume that we are going to use a Bayesian algorithm to make our online predictions. Specifically, assume that we have a Gaussian prior on the parameters:

$$p(\theta) = \mathcal{N}(\theta; \vec{0}, \nu^2 I_n),$$

where I_n is the n-by-n identity matrix, $\mathcal{N}(\cdot; \mu, \Sigma)$ is the density of a Gaussian with mean μ and covariance Σ, and $\nu^2 > 0$ is some fixed constant governing the variance in our prior. Also, let

$$p_t(\theta) = p(\theta|S_t) = \frac{\left(\prod_{i=1}^t p(y^{(i)}|x^{(i)}, \theta)\right) p(\theta)}{\int_\theta \left(\prod_{i=1}^t p(y^{(i)}|x^{(i)}, \theta)\right) p(\theta) d\theta}$$

be the posterior distribution over θ given the first t training examples. We also have that $p_0(\theta) = p(\theta)$ is just the prior distribution.

On iteration t, we are given the input $x^{(t)}$, and the BMA algorithm makes a prediction using the posterior distribution over the outputs:

$$p(y|z^{(t)}, S_{t-1}) = \int_\theta p(y|z^{(t)}, \theta) p(\theta|S_{t-1}) d\theta.$$

We are then given the true label $y^{(t)}$, and we suffer logloss $-\log p(y^{(t)}|z^{(t)}, S_{t-1})$. We define the cumulative loss of the BMA algorithm after T rounds to be

$$L_{\text{BMA}}(S) = \sum_{t=1}^{T} -\log p(y^{(t)}|z^{(t)}, S_{t-1}).$$

We will be interested in comparing against the loss of any expert that uses some fixed parameters $\theta \in \mathbb{R}^n$. Define $\ell_\theta(t) = -\log p(y^{(t)}|x^{(t)}, \theta)$, and let

$$L_\theta(S) = \sum_{t=1}^{T} \ell_\theta(t) = \sum_{t=1}^{T} -\log p(y^{(t)}|z^{(t)}, \theta).$$

A more general form of the following theorem has been proved in Kakade and Ng (2005), we specialized it to the linear regression case.

Theorem 29. *[(Kakade & Ng, 2005) Theorem 2.2, for $c = 1/\sigma^2$.] For all sequences S of length T and for all θ^**

$$L_{BMA}(S) \leq L_{\theta^*}(S) + \frac{1}{2\nu^2}\|\theta^*\|^2 + \frac{n}{2}\log\left(1 + \frac{2T\nu^2}{n\sigma^2}\right).$$

Now we'll take a closer look at the predictions done by the BMA algorithm. Define $A_t = \frac{1}{\nu^2} I_n + \frac{1}{\sigma^2}\sum_{i=1}^{t} z^{(i)} z^{(i)^T}$, and $b_t = \frac{1}{\sigma^2}\sum_{i=1}^{t} z^{(i)} y^{(i)}$. We have that

$$p_t(\theta) = p(\theta|S_t) = \mathcal{N}\left(\theta; \hat{\theta}_t, \hat{\Sigma}_t\right), \tag{B.43}$$

where $\hat{\theta}_t = A_t^{-1} b_t$, and $\hat{\Sigma}_t = A_t^{-1}$. Also, the predictions are given by

$$p(y^{(t+1)}|z^{(t+1)}, S_t) = \mathcal{N}\left(y^{(t+1)}; \hat{y}_{t+1}, s_{t+1}^2\right) \tag{B.44}$$

where $\hat{y}_{t+1} = \hat{\theta}_t^T z^{(t+1)}$, $s_{t+1}^2 = z^{(t+1)^T} \hat{\Sigma}_t z^{(t+1)} + \sigma^2$. In contrast, the prediction of a fixed expert using parameter θ^* would be

$$p(y^{(t)}|z^{(t)}, \theta^*) = \mathcal{N}\left(y^{(t)}; y_t^*, \sigma^2\right), \tag{B.45}$$

where $y_t^* = \theta^{*T} z^{(t)}$.

Note that $s_t^2 \geq \sigma^2$, i.e., the BMA algorithm always predicts with a larger variance than a single expert.

Lemma 30. *The terms s_t^2 satisfy the following:*

$$\frac{s_t^2 - \sigma^2}{\sigma^2} \leq \frac{2\nu^2}{\sigma^2}, \tag{B.46}$$

$$\sum_{t=1}^{T} \frac{s_t^2 - \sigma^2}{\sigma^2} \leq \frac{2\nu^2}{\sigma^2 \log(1 + \frac{2\nu^2}{\sigma^2})} n \log\left(1 + \frac{2T\nu^2}{n\sigma^2}\right). \tag{B.47}$$

Proof. Let $m_t = \frac{s_t^2 - \sigma^2}{\sigma^2}$. Consider a sequence of examples $\{(z^{(1)}, y^{(i)}), \ldots, (z^{(T)}, y^{(T)})\}$, where all the outputs $y^{(1)} = \cdots = y^{(T)} = 0$. Given this sequence, the BMA algorithm's predictions will also all be $\hat{y}_t = 0$. Thus, we have

$$L_{\text{BMA}}(T) = \sum_{t=1}^{T} -\log \mathcal{N}(0; 0, s_t^2) \tag{B.48}$$

$$= \sum_{t=1}^{T} \left[-\log \frac{1}{\sqrt{2\pi}} + \frac{1}{2} \log s_t^2 \right]. \tag{B.49}$$

Now, consider the loss of an expert using the zero parameter vector $\theta = \vec{0}$. We have

$$L_{\theta^*}(T) = \sum_{t=1}^{T} -\log \mathcal{N}(0; 0, \sigma^2) \tag{B.50}$$

$$= \sum_{t=1}^{T} \left[-\log \frac{1}{\sqrt{2\pi}} + \frac{1}{2} \log \sigma^2 \right]. \tag{B.51}$$

Substituting Equations (B.49) and (B.51) into the main conclusion of Theorem 29, we get

$$\sum_{t=1}^{T} \frac{1}{2} \log s_t^2 \leq \sum_{t=1}^{T} \frac{1}{2} \log \sigma^2 + \frac{n}{2} \log\left(1 + \frac{2T\nu^2}{n\sigma^2}\right). \tag{B.52}$$

Using the definition $m_t = s_t^2/\sigma^2 - 1$, we get

$$\sum_{t=1}^{T} \log(1 + m_t) \leq n \log\left(1 + \frac{2T\nu^2}{n\sigma^2}\right). \tag{B.53}$$

Finally, observe that for all $0 \leq \epsilon \leq K$, we have

$$\log(1 + \epsilon) \geq \frac{\log(1 + K)}{K} \cdot \epsilon \tag{B.54}$$

Also, we can bound m_t as follows:

$$\begin{aligned} m_t &= \frac{s_t^2}{\sigma^2} - 1 \\ &= \frac{z^{(t+1)^T} \hat{\Sigma}_t z^{(t+1)} + \sigma^2}{\sigma^2} - 1 \\ &= \frac{1}{\sigma^2} z^{(t+1)^T} \left(\frac{1}{\nu^2} I_n + \frac{1}{\sigma^2} \sum_{i=1}^{t} z^{(t)} z^{(t)^T}\right)^{-1} z^{(t+1)} \\ &\leq \frac{1}{\sigma^2} z^{(t+1)^T} \left(\frac{1}{\nu^2} I_n\right)^{-1} z^{(t+1)} \\ &\leq \frac{2\nu^2}{\sigma^2}. \end{aligned} \tag{B.55}$$

For the last step, we used the fact that $\|z^{(t+1)}\|_2 \leq \sqrt{2}$. This shows (B.46).

Putting together (B.55) and (B.54) with $\epsilon = m_t$ and $K = 2\nu^2/\sigma^2$, we find that

$$m_t \leq \frac{2\nu^2}{\sigma^2 \log(1 + \frac{2\nu^2}{\sigma^2})} \log(1 + m_t). \tag{B.56}$$

Finally, Equations (B.56) and (B.53) together imply (B.47).

□

Proof of Theorem 28. As a direct consequence of Eqn. (B.47) of Lemma 30, we have the following bound for the number of times $N_{s_t^2 > \sigma^2(1+\epsilon^2)}$ that $s_t^2 > \sigma^2 + \epsilon^2\sigma^2$:

$$N_{s_t^2 > \sigma^2(1+\epsilon^2)} \leq \frac{1}{\epsilon^2} \frac{2\nu^2}{\sigma^2 \log(1 + \frac{2\nu^2}{\sigma^2})} n \log(1 + \frac{2T\nu^2}{n\sigma^2}). \tag{B.57}$$

If we let θ^* denote the "true" underlying parameter, and $w^{(t)}$ the noise at time t, then we have

$$y^{(t)} = \theta^{*\top} z^{(t)} + w^{(t)}.$$

Using the notation we just introduced we can then rewrite Theorem 29 as

$$\begin{aligned}
&\sum_{t=1}^{T} \left(\frac{1}{2s_t^2} \left(w^{(t)} + \theta^{*\top} z^{(t)} - \hat{\theta}_{t-1}^{\top} z^{(t)} \right)^2 + \log \sqrt{2\pi} s_t \right) \leq \\
&\quad \sum_{t=1}^{T} \left(\frac{1}{2\sigma^2} \left(w^{(t)} \right)^2 + \log \sqrt{2\pi}\sigma \right) \\
&\quad + \frac{1}{2\nu^2} \|\theta^*\|^2 + \frac{n}{2} \log \left(1 + \frac{2T\nu^2}{n\sigma^2} \right).
\end{aligned}$$

From now on let $e^{(t)} = \theta^{*\top} z^{(t)} - \hat{\theta}_{t-1}^{\top} z^{(t)}$.

Splitting up the summation into case where $s_t^2 > \sigma^2(1+\epsilon^2)$ and $s_t^2 \leq \sigma^2(1+\epsilon^2)$, and leaving out some positive terms from the left-handside, and using the fact that $s_t \geq \sigma$ for all t we get:

$$\begin{aligned}
&\sum_{t: s_t^2 \leq \sigma^2(1+\epsilon^2)} \frac{1}{2s_t^2} \left(w^{(t)} + e^{(t)} \right)^2 \leq \\
&\sum_{t=1}^{T} \frac{1}{2\sigma^2} \left(w^{(t)} \right)^2 + \frac{1}{2\nu^2} \|\theta^*\|^2 + \frac{n}{2} \log \left(1 + \frac{2T\nu^2}{n\sigma^2} \right),
\end{aligned}$$

which implies

$$\begin{aligned}
&\sum_{t: s_t^2 \leq \sigma^2(1+\epsilon^2)} \frac{1}{2\sigma^2(1+\epsilon^2)} \left(w^{(t)} + e^{(t)} \right)^2 \leq \\
&\sum_{t=1}^{T} \frac{1}{2\sigma^2} \left(w^{(t)} \right)^2 + \frac{1}{2\nu^2} \|\theta^*\|^2 + \frac{n}{2} \log \left(1 + \frac{2T\nu^2}{n\sigma^2} \right).
\end{aligned} \tag{B.58}$$

Now we would like to bound the number of times we can have that $e^{(t)} > \mu$ as a function of μ. Since $e^{(t)}$ could depend on the whole history $\{y^{(i)}, z^{(i)}\}_{i=1}^{t-1}$, we use a martingale argument.

Let K_w be such that for all t we have $|w^{(t)}| \leq K_w$ and such that $\mu \leq K_w$. (The latter condition ensures that when clipping the errors, errors do not get clipped below μ.) Let $\tilde{e}^{(t)}$ be defined as $e^{(t)}$ clipped to the interval $[-K_w, K_w]$. I.e., we define $\tilde{e}^{(t)} = \min\{K_w, \max\{-K_w, e^{(t)}\}\}$. Then we have

$$(w^{(t)} + \tilde{e}^{(t)})^2 \leq (w^{(t)} + e^{(t)})^2.$$

For the left-hand side of Eqn. (B.58) we apply Azuma's inequality to the martingale $Z_t = \sum_{i=1}^{t} (w^{(i)} + \tilde{e}^{(i)})^2 - \sigma^2 - (\tilde{e}^{(i)})^2$, which gives (the increment/decrement in one time step is at most $3K_w^2 + \sigma^2$)

$$P\left(\sum_{t:s_t^2 \leq \sigma^2(1+\epsilon^2)} (w^{(t)} + \tilde{e}^{(t)})^2 - (\tilde{e}^{(t)})^2 - \bar{T}\sigma^2 < -\bar{T}\lambda\right) \leq \exp\left(-\bar{T}\lambda^2/2(3K_w^2 + \sigma^2)^2\right), \tag{B.59}$$

where $\bar{T} = T - N_{s_t^2 > \sigma^2(1+\epsilon^2)}$.

For the right-hand side of Eqn. (B.58) we apply Azuma's inequality to the martingale $Z_t = \sum_{i=1}^{t} (w^{(i)})^2 - \sigma^2$, which gives us (the increment/decrement in one time step is bounded by $K_w^2 + \sigma^2$)

$$P\left(\sum_t (w^{(t)})^2 - T\sigma^2 > T\lambda\right) \leq \exp(-T\lambda^2/2(K_w^2 + \sigma^2)^2). \tag{B.60}$$

Combining the concentration results of Eqn. (B.59,B.60) with Eqn. (B.58) gives that with probability $1 - \exp(-\bar{T}\lambda^2/2(3K_w^2 + \sigma^2)^2) - \exp(-T\lambda^2/2(K_w^2 + \sigma^2)^2)$ the

following holds:

$$\sum_{t:s_t^2 \leq \sigma^2(1+\epsilon^2)} \frac{1}{2\sigma^2(1+\epsilon^2)}(\tilde{e}^{(t)})^2 + \bar{T}\frac{\sigma^2-\lambda}{2\sigma^2(1+\epsilon^2)} \leq$$
$$T\frac{\sigma^2+\lambda}{2\sigma^2} + \frac{1}{2\nu^2}\|\theta^*\|^2 + \frac{n}{2}\log\left(1+\frac{2T\nu^2}{n\sigma^2}\right).$$

Which is equivalent to

$$\sum_{t:s_t^2 \leq \sigma^2(1+\epsilon^2)} \frac{1}{2\sigma^2(1+\epsilon^2)}(\tilde{e}^{(t)})^2 \leq T\frac{\sigma^2+\lambda}{2\sigma^2}$$
$$-\bar{T}\frac{\sigma^2-\lambda}{2\sigma^2(1+\epsilon^2)} + \frac{1}{2\nu^2}\|\theta^*\|^2 + \frac{n}{2}\log\left(1+\frac{2T\nu^2}{n\sigma^2}\right).$$

As a consequence we have

$$N_{s_t^2 \leq \sigma^2(1+\epsilon^2), e^{(t)} \geq \mu}\frac{\mu^2}{2\sigma^2(1+\epsilon^2)} =$$
$$\sum_{t:s_t^2 \leq \sigma^2(1+\epsilon^2), e^{(t)} \geq \mu} \frac{1}{2\sigma^2(1+\epsilon^2)}\mu^2 \leq T\frac{\sigma^2+\lambda}{2\sigma^2}$$
$$-\bar{T}\frac{\sigma^2-\lambda}{2\sigma^2(1+\epsilon^2)} + \frac{1}{2\nu^2}\|\theta^*\|^2 + \frac{n}{2}\log\left(1+\frac{2T\nu^2}{n\sigma^2}\right).$$

Substituting $\bar{T} = T - N_{s_t^2 > \sigma^2(1+\epsilon^2)}$ (and using $-\lambda < +\lambda$) we get

$$N_{s_t^2 \leq \sigma^2(1+\epsilon^2), e^{(t)} \geq \mu}\frac{\mu^2}{2\sigma^2(1+\epsilon^2)} \leq$$
$$T\frac{\epsilon^2\sigma^2+\epsilon^2\lambda+2\lambda}{2\sigma^2(1+\epsilon^2)} + N_{s_t^2 > \sigma^2(1+\epsilon^2)}\frac{\sigma^2+\lambda}{2\sigma^2(1+\epsilon^2)}$$
$$+\frac{1}{2\nu^2}\|\theta^*\|^2 + \frac{n}{2}\log\left(1+\frac{2T\nu^2}{n\sigma^2}\right).$$

Substituting the bound for $N_{s_t^2>\sigma^2(1+\epsilon^2)}$ from Eqn. (B.57) we get

$$\begin{aligned}N_{s_t^2\leq\sigma^2(1+\epsilon^2),e^{(t)}\geq\mu}\frac{\mu^2}{2\sigma^2(1+\epsilon^2)} &\leq T\frac{\epsilon^2\sigma^2+\epsilon^2\lambda+2\lambda}{2\sigma^2(1+\epsilon^2)}\\ &+\frac{\sigma^2+\lambda}{2\sigma^2(1+\epsilon^2)}\frac{1}{\epsilon^2}\frac{2\nu^2}{\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}n\log(1+\frac{2T\nu^2}{n\sigma^2})\\ &+\frac{1}{2\nu^2}\|\theta^*\|^2+\frac{n}{2}\log\left(1+\frac{2T\nu^2}{n\sigma^2}\right).\end{aligned}$$

Multiplying both sides with $2\sigma^2(1+\epsilon^2)$ gives

$$\begin{aligned}N_{s_t^2\leq\sigma^2(1+\epsilon^2),e^{(t)}\geq\mu}\mu^2 &\leq T(\epsilon^2\sigma^2+\epsilon^2\lambda+2\lambda)\\ &+(\sigma^2+\lambda)\frac{1}{\epsilon^2}\frac{2\nu^2}{\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}n\log(1+\frac{2T\nu^2}{n\sigma^2})\\ &+\frac{\sigma^2(1+\epsilon^2)}{\nu^2}\|\theta^*\|^2+n\sigma^2(1+\epsilon^2)\log\left(1+\frac{2T\nu^2}{n\sigma^2}\right).\end{aligned}$$

This all holds w.p.

$$1-\exp(-\bar{T}\lambda^2/2(3K_w^2+\sigma^2)^2)-\exp(-T\lambda^2/2(K_w^2+\sigma^2)^2).$$

Now choose

$$\epsilon^2=\sqrt{\frac{2\nu^2n\log(1+\frac{2T\nu^2}{n\sigma^2})}{T\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}}, \tag{B.61}$$

then we get

$$\begin{aligned}N_{s_t^2\leq\sigma^2(1+\epsilon^2),e^{(t)}\geq\mu}\mu^2 &\leq 2\lambda T\\ &+2(\sigma^2+\lambda)\sqrt{\frac{2T\nu^2n\log(1+\frac{2T\nu^2}{n\sigma^2})}{\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}}\\ &+\left(1+\sqrt{\frac{2\nu^2n\log(1+\frac{2T\nu^2}{n\sigma^2})}{T\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}}\right)\frac{\sigma^2}{\nu^2}\|\theta^*\|^2\\ &+\left(1+\sqrt{\frac{2\nu^2n\log(1+\frac{2T\nu^2}{n\sigma^2})}{T\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}}\right)n\sigma^2\log\left(1+\frac{2T\nu^2}{n\sigma^2}\right).\end{aligned}$$

Substituting Eqn. (B.61) into Eqn. (B.57) gives

$$N_{s_t^2>\sigma^2(1+\epsilon^2)} \leq \sqrt{T\frac{2\nu^2}{\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}n\log(1+\frac{2T\nu^2}{n\sigma^2})}.$$

So we have that there exists $T^* = O(\text{poly}(n))$ (note we assume ν, σ are fixed in our analysis) such that for all $T > T^*$ we have that $N_{s_t^2>\sigma^2(1+\epsilon^2)} \leq \frac{1}{2}T$. Thus (for $T > T^*$) all the above holds w.p.

$$1 - 2\exp(-T\lambda^2/4(3K_w^2+\sigma^2)^2).$$

Now setting $\frac{\delta}{2} = 2\exp(-T\lambda^2/4(3K_w^2+\sigma^2)^2)$, or equivalently $\lambda^2 = \frac{4(3K_w^2+\sigma^2)^2}{T}\log\frac{4}{\delta}$ we get that

$$\begin{aligned}
N_{s_t^2\leq\sigma^2(1+\epsilon^2),e^{(t)}\geq\mu}\mu^2 &\leq 2T\sqrt{\frac{4(3K_w^2+\sigma^2)^2}{T}\log\frac{4}{\delta}} \\
&+ 2(\sigma^2 + \sqrt{\frac{4(3K_w^2+\sigma^2)^2}{T}\log\frac{4}{\delta}})\sqrt{\frac{2T\nu^2 n\log(1+\frac{2T\nu^2}{n\sigma^2})}{\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}} \\
&+ \left(1+\sqrt{\frac{2\nu^2 n\log(1+\frac{2T\nu^2}{n\sigma^2})}{T\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}}\right)\frac{\sigma^2}{\nu^2}\|\theta^*\|^2 \\
&+ \left(1+\sqrt{\frac{2\nu^2 n\log(1+\frac{2T\nu^2}{n\sigma^2})}{T\sigma^2\log(1+\frac{2\nu^2}{\sigma^2})}}\right)n\sigma^2\log\left(1+\frac{2T\nu^2}{n\sigma^2}\right).
\end{aligned} \tag{B.62}$$

holds with probability $1 - \frac{\delta}{2}$. Now recall that $K_w = \max\{\mu, \max_{t\in 1:T}|w^{(t)}|\}$. Thus from Prop. 21 we have that $K_w \leq \max\{\mu, \sigma\log\frac{8T}{\sqrt{2\pi}\delta}\}$ with probability $1 - \frac{\delta}{2}$. Thus

we have that

$$
\begin{aligned}
&N_{s_t^2 \leq \sigma^2(1+\epsilon^2), e^{(t)} \geq \mu} \mu^2 \leq \\
&4(3(\max\{\mu, \sigma \log \frac{8T}{\sqrt{2\pi}\delta}\})^2 + \sigma^2)\sqrt{T \log \frac{4}{\delta}} \\
&+ 2\left(\sigma^2 + 2(3(\max\{\mu, \sigma \log \frac{8T}{\sqrt{2\pi}\delta}\})^2 + \sigma^2)\sqrt{\frac{1}{T}\log\frac{4}{\delta}}\right) \\
&\sqrt{\frac{2T\nu^2 n \log(1+\frac{2T\nu^2}{n\sigma^2})}{\sigma^2 \log(1+\frac{2\nu^2}{\sigma^2})}} \\
&+ \left(1 + \sqrt{\frac{2\nu^2 n \log(1+\frac{2T\nu^2}{n\sigma^2})}{T\sigma^2 \log(1+\frac{2\nu^2}{\sigma^2})}}\right) \frac{\sigma^2}{\nu^2}\|\theta^*\|^2 \\
&+ \left(1 + \sqrt{\frac{2\nu^2 n \log(1+\frac{2T\nu^2}{n\sigma^2})}{T\sigma^2 \log(1+\frac{2\nu^2}{\sigma^2})}}\right) n\sigma^2 \log\left(1 + \frac{2T\nu^2}{n\sigma^2}\right).
\end{aligned}
$$

holds with probability $1 - \delta$.

After simplification we get that for any $T > T^* = O(\text{poly}(n))$ that

$$
\begin{aligned}
&N_{s_t^2 \leq \sigma^2(1+\epsilon^2), e^{(t)} \geq \mu} = \\
&\qquad O(\sqrt{T}(\log T)^3 \text{poly}(\|\theta^*\|_2, n, \log\frac{1}{\delta}, \frac{1}{\mu}))
\end{aligned} \tag{B.63}
$$

holds with probability $1 - \delta$. The condition that $T > T^* = O(\text{poly}(n))$ is readily incorporated by adjusting the polynomial in Eqn. (B.63), and can thus be omitted.

Taking into account that we might have $e^{(t)} \geq \mu$ when $s_t^2 \geq (1+\epsilon^2)\sigma^2$, we get that

$$
\begin{aligned}
N_{e^{(t)} \geq \mu} &\leq N_{s_t^2 \leq \sigma^2(1+\epsilon^2), e^{(t)} \geq \mu} \\
&\quad + N_{s_t^2 \leq \sigma^2(1+\epsilon^2)} \\
&= O(\sqrt{T}(\log T)^3 \text{poly}(\|\theta^*\|_2, n, \log\frac{1}{\delta}, \frac{1}{\mu})).
\end{aligned}
$$

Which proves the theorem. □

B.6.2 Proof of Lemma 11

Lemma 11 considers the setting where the model is not updated for $k_1 H$ steps. And then updated for all these steps at once. The result we have from Theorem 28 applies only to the setting where the updates are done between every datapoint. Moreover Lemma 11 considers n linear regression problems simultaneously.

Lemma 31. *Let any $l \in 1 : n$ be fixed. For the algorithm described in Section 3.4 we have that the number N_μ of times a state is encountered such that*

$$|(A_{l,:}\phi(x_t) + B_{l,:}u_t) - (\hat{A}^{(i)}_{l,:}\phi(x_t) - \hat{B}^{(i)}_{l,:}u_t)| > \mu \tag{B.64}$$

satisfies

$$\begin{aligned} N_\mu &= O(k_1 H\sqrt{N k_1 H}(\log N k_1 H)^3 \\ &\quad \text{poly}(\|A_{l,:}\|_F, \|B_{l,:}\|_F, n_S, n_{\mathcal{A}}, \log\frac{1}{\delta}, \frac{1}{\mu})) \end{aligned}$$

Proof. Consider $k_1 H$ versions of the data, permuted such that within each subsequence of length $k_1 H$ obtained under one policy, in every permutation a different data point comes first, but no data points are permutated across trials under different policies. Then every prediction done in our algorithm is also done for (at least) one permutation in this new setup (the new setup includes more predictions than just these). Thus bounding the number of large errors in this new setup gives us a bound on the number of large errors encountered in our algorithm. This new setup consists of $k_1 H$ data sequences of length $N k_1 H$ each. Using Theorem 28 we get that

$$\begin{aligned} N_\mu &= O(k_1 H\sqrt{N k_1 H}(\log N k_1 H)^3 \\ &\quad \text{poly}(\sqrt{\|A_{l,:}\|_F^2, \|B_{l,:}\|_F^2}, n_S + n_{\mathcal{A}}, \log\frac{1}{\delta}, \frac{1}{\mu})) \end{aligned}$$

which can be simplified (and be made less tight) to the statement of the lemma. □

Proof of Lemma 11. When x_t, u_t satisfy $\|(A\phi(x) + Bu) - (\hat{A}^{(i)}\phi(x) - \hat{B}^{(i)}u)\|_2 > \mu$,

then there must be $l \in 1 : n_S$ such that

$$|(A_{l,:}\phi(x_t) + B_{l,:}u_t) - (\hat{A}^{(i)}_{l,:}\phi(x_t) - \hat{B}^{(i)}_{l,:}u_t)| > \mu/\sqrt{n_S}$$

is satisfied. From Lemma 31 we have that this can happen at most

$$N' = O(k_1 H\sqrt{Nk_1H}(\log Nk_1H)^3 \text{poly}(\|A_{l,:}\|_F, \|B_{l,:}\|_F, n_S, n_{\mathcal{A}}, \log\frac{1}{\delta}, \frac{1}{\mu}))$$

times for each l w.p. $1-\delta$. So it can happen at most $n_S N'$ times w.p. $1 - n_S\delta$. This proves the lemma. □

B.6.3 Proof of Theorem 4 for linearly parameterized dynamics

Proof of Theorem 4. Assume the algorithm runs for N iterations. Using the Hoeffding inequality we have that for

$$\forall i = 1 : N \quad |\hat{U}_M(\pi^{(i)}) - U_M(\pi^{(i)})| \leq \frac{\alpha}{16} \tag{B.65}$$

to hold with probability $1 - \delta'$, it suffices that

$$k_1 \geq \frac{16^2 H^2 R^2_{\max}}{2\alpha^2} \log\frac{2N}{\delta'}. \tag{B.66}$$

Using the Hoeffding inequality, we also have that for

$$|\hat{U}_M(\pi_T) - U_M(\pi_T)| \leq \frac{\alpha}{16} \tag{B.67}$$

to hold with probability $1 - \delta''$, it suffices that

$$N_T \geq \frac{16^2 H^2 R^2_{\max}}{2\alpha^2} \log\frac{2}{\delta''}. \tag{B.68}$$

From Theorem 10 we have that for

$$|U_{\hat{M}^{(i)}}(\pi_T) - U_M(\pi_T)| \leq \frac{\alpha}{8} \tag{B.69}$$

to hold with probability $1 - \delta'''$, it suffices that $N_T =$

$$\Omega\left(\text{poly}(\frac{1}{\alpha}, \frac{1}{\delta'''}, H, R_{\max}, \|A\|_{\text{F}}, \|B\|_{\text{F}}, n_S, n_{\mathcal{A}}, k_1, N)\right). \tag{B.70}$$

Since the algorithm only exits in iteration N, we must have for all $i = 1 : N-1$ that

$$\hat{U}_M(\pi^{(i)}) < \hat{U}_M(\pi_T) - \alpha/2. \tag{B.71}$$

Combining Eqn. (B.71, B.65, B.67, B.69) and the fact that $\pi^{(i)}$ is $\alpha/8$-optimal for $\hat{M}^{(i)}$ we get

$$\forall i (1 \leq i \leq N-1) U_{\hat{M}^{(i)}}(\pi^{(i)}) \geq U_M(\pi^{(i)}) + \alpha/8. \tag{B.72}$$

Eqn. (B.72) states that for every iteration i that the algorithm continues, the model is inaccurate in evaluating the utility of the policy $\pi^{(i)}$. Now using the contrapositive of the Simulation Lemma (choosing $\epsilon = \frac{1}{2}\frac{\alpha/8}{H^2 R_{\max}}$) we get from Eqn. (B.72) that the policy $\pi^{(i)}$ must be visiting a state-action pair (x, u) that satisfies

$$d_{\text{var}}(P(\cdot|x,u), \hat{P}^{(i)}(\cdot|x,u)) > \frac{\alpha}{16H^2 R_{\max}} \tag{B.73}$$

with probability at least $\frac{\alpha}{16HR_{\max}}$ in every trial of horizon H. If (x, u) satisfies Eqn. (B.73) then we must have (using Prop. 8) that

$$\|(A\phi(x) + Bu) - (\hat{A}^{(i)}\phi(x) - \hat{B}^{(i)}u)\|_2 > \frac{\sqrt{2\pi}\sigma\alpha}{16H^2 R_{\max}}.$$

From Lemma 11 we have that with probability $1 - \delta''''$ this can happen only

$$\begin{aligned} N_\mu &= O(k_1\sqrt{k_1 N}(\log k_1 N)^3 \text{poly}(\|A\|_F, \\ &\quad \|B\|_F, n_S, n_{\mathcal{A}}, \log\frac{1}{\delta''''}, \frac{1}{\alpha}, H, \frac{16H^2 R_{\max}}{\sqrt{2\pi}\sigma\alpha})) \end{aligned}$$

times in N iterations of the algorithm. Substituting in the expression for k_1 from Eqn. (B.66) and simplifying gives us

$$\begin{aligned} N_\mu &= O(\sqrt{N}(\log N)^5 \mathrm{poly}(\|A\|_F, \|B\|_F, n_S, n_{\mathcal{A}}, \\ &\qquad \log\frac{1}{\delta''''}, \frac{1}{\alpha}, H, R_{\max}, \log\frac{1}{\delta'})). \end{aligned} \tag{B.74}$$

On the other hand, if the algorithm continues, we have from Eqn. (B.73) and Lemma 3 (choose $a = \frac{\alpha}{16HR_{\max}}N/2 - \log\frac{1}{\delta''''}$) that such an error must be encountered with probability $1 - \delta'''''$ at least

$$\frac{\alpha}{32HR_{\max}}N - \log\frac{1}{\delta'''''} \tag{B.75}$$

times. From Eqn. (B.74) and Eqn. (B.75) we have that after a number of iterations

$$\begin{aligned} O(\mathrm{poly}(\|A\|_F, \|B\|_F, n_S, n_{\mathcal{A}}, \log\frac{1}{\delta''''}, \frac{1}{\alpha}, H, \\ R_{\max}, \log\frac{1}{\delta'}, \log\frac{1}{\delta'''''})) \end{aligned}$$

the algorithm must have terminated with probability $1 - \delta' - \delta'' - \delta''' - \delta'''' - \delta'''''$. Now choose $\delta' = \delta'' = \delta''' = \delta'''' = \delta''''' = \frac{\delta}{5}$, to obtain the bound on the number of iterations of Eqn. (3.3). Given this bound on the number of iterations N, it is easily verified that the conditions of Eqn. (B.66, B.68, B.70) on N_T and k_1 are met by Eqn. (3.4) and Eqn. (3.5) of Theorem 4. Also, since we chose N_T, k_1 such that $\hat{U}_M(\pi_T)$ and $\{\hat{U}_M(\pi^{(i)}\}_i$ are accurately evaluated (as specified in Eqn. (B.65,B.67)), we have that Eqn. (3.2) holds when the algorithm terminates.

□

Appendix C

Derivation of EM Algorithm of Chapter 4

This Appendix derives the EM algorithm that optimizes Eqn. (4.7). The derivation is based on the presentation of the EM algorithm in Neal and Hinton (1999). Note that because of discounting, the objective is slightly different from the standard setting of learning the parameters of a Markov chain with unobserved variables in the training data.

Since we are using a first-order model, we have $P_{\hat{\theta}}(s_{t+k}|s_t, a_{t:t+k-1}) =$

$$\sum_{S_{t+1:t+k-1}} P_{\hat{\theta}}(s_{t+k}|S_{t+k-1}, a_{t+k-1}) P_{\hat{\theta}}(S_{t+k-1}|S_{t+k-2}, a_{t+k-2}) \dots P_{\hat{\theta}}(S_{t+1}|s_t, a_t).$$

Here, the summation is over all possible state sequences $S_{t+1:t+k-1}$. So we have

$$\begin{aligned}
&\sum_{t=0}^{T-1}\sum_{k=1}^{T-t}\gamma^k \log P_{\hat{\theta}}(s_{t+k}|s_t, a_{t:t+k-1}) \\
&= \sum_{t=0}^{T-1}\gamma \log P_{\hat{\theta}}(s_{t+1}|s_t, a_t) + \sum_{t=0}^{T-1}\sum_{k=2}^{T-t}\gamma^k \log \sum_{S_{t+1:t+k-1}} \frac{Q_{t,k}(S_{t+1:t+k-1})}{Q_{t,k}(S_{t+1:t+k-1})} \\
&\quad P_{\hat{\theta}}(s_{t+k}|S_{t+k-1}, a_{t+k-1}) P_{\hat{\theta}}(S_{t+k-1}|S_{t+k-2}, a_{t+k-2}) \dots P_{\hat{\theta}}(S_{t+1}|s_t, a_t) \\
&\geq \sum_{t=0}^{T-1}\gamma \log P_{\hat{\theta}}(s_{t+1}|s_t, a_t) + \sum_{t=0}^{T-1}\sum_{k=2}^{T-t}\gamma^k Q_{t,k}(S_{t+1:t+k-1}) \\
&\quad \log \frac{P_{\hat{\theta}}(s_{t+k}|S_{t+k-1}, a_{t+k-1}) P_{\hat{\theta}}(S_{t+k-1}|S_{t+k-2}, a_{t+k-2}) \dots P_{\hat{\theta}}(S_{t+1}|s_t, a_t)}{Q_{t,k}(S_{t+1:t+k-1})}. \qquad \text{(C.1)}
\end{aligned}$$

Here, $Q_{t,k}$ is a probability distribution, and the inequality follows from Jensen's inequality and the concavity of $\log(\cdot)$. Similar to Neal and Hinton (1999), the EM algorithm optimizes Eqn. (C.1) by alternately optimizing with respect to the distributions $Q_{t,k}$ (E-step), and the transition probabilities $P_{\hat{\theta}}(\cdot|\cdot,\cdot)$ (M-step). Optimizing with respect to the $Q_{t,k}$ variables (E-step) is achieved by setting

$$Q_{t,k}(S_{t+1:t+k-1}) = P_{\hat{\theta}}(S_{t+1}, \dots, S_{t+k-1}|S_t = s_t, S_{t+k} = s_{t+k}, A_{t:t+k-1} = a_{t:t+k-1}). \qquad \text{(C.2)}$$

Optimizing with respect to the transition probabilities $P_{\hat{\theta}}(\cdot|\cdot,\cdot)$ (M-step) for $Q_{t,k}$ fixed as in Eqn. (C.2) is done by updating $\hat{\theta}$ to $\hat{\theta}_{\text{new}}$ such that $\forall\, i, j \in S, \forall\, a \in A$ we have that $P_{\hat{\theta}_{\text{new}}}(j|i,a) = stats(j,i,a)/\sum_{k\in S} stats(k,i,a)$, where $stats(j,i,a) = \sum_{t=0}^{T-1}\sum_{k=1}^{T-t}\sum_{l=0}^{k-1}\gamma^k P_{\hat{\theta}}(S_{t+l+1} = j, S_{t+l} = i|S_t = s_t, S_{t+k} = s_{t+k}, A_{t:t+k-1} = a_{t:t+k-1})$ $\mathbf{1}\{a_{t+l} = a\}$. Note that only the pairwise marginals $P_{\hat{\theta}}(S_{t+l+1}, S_{t+l}|S_t, S_{t+k}, A_{t:t+k-1})$ are needed in the M-step, and so it is sufficient to compute only these when optimizing with respect to the $Q_{t,k}$ variables in the E-step.

Appendix D

Proofs for Chapter 6

D.1 Proof of Theorem 12

We first prove two propositions that will help us to prove Theorem 12. Proposition 32 shows our algorithm converges under certain assumptions on the utility function U. Proposition 33 shows these assumptions on the utility function U are true so long as certain assumptions on the transition functions $\{f_t\}_{t=0}^{H}$, the approximate transition functions $\{\hat{f}_t\}_{t=0}^{H}$, the reward function R, and the policy class π_θ are true.

Proposition 32. *Let U be bounded from above. Let $K, \epsilon > 0$ be such that $\|\nabla_\theta U(\theta) - \nabla_\theta \hat{U}(\theta)\|_2 \leq K\epsilon$. Let $M > 0$ be such that $\|\nabla_\theta^2 U(\theta)\|_2 \leq M$. Let the line search in our algorithm be η-optimal (i.e., it finds a point that is within η of the optimal utility along the line). Let the local policy improvement algorithm be policy gradient. Then our algorithm converges to a locally optimal region, in the following sense: it converges to a region where the following holds:*

$$\|\nabla_\theta U(\theta)\|_2^2 \leq 2K^2\epsilon^2 + 2M\eta. \tag{D.1}$$

For the case of exact line search ($\eta = 0$) we obtain:

$$\|\nabla_\theta U(\theta)\|_2^2 \leq 2K^2\epsilon^2. \tag{D.2}$$

Proof of Theorem 12. For any x, y, the following holds (we use the first order Taylor expansion of U at x, and $\|\nabla_\theta^2 U(\theta)\|_2 \leq M$ to bound the second order error term):

$$U(y) \geq U(x) + \nabla_\theta U(x)^\top (y - x) - \frac{M}{2}\|y - x\|_2^2.$$

For $x = \theta^{(i)}$ and $y = \theta^{(i)} + \alpha d^{(i)}$ we get that:

$$U(\theta^{(i)} + \alpha d^{(i)}) \geq U(\theta^{(i)}) + \alpha \nabla_\theta U(\theta^{(i)})^\top d^{(i)} - \frac{\alpha^2 M}{2}\|d^{(i)}\|_2^2.$$

Let $\alpha = \alpha^* = \frac{\nabla_\theta U(\theta^{(i)})^\top d^{(i)}}{M\|d^{(i)}\|_2^2}$, then we have that:

$$U(\theta^{(i)} + \alpha^* d^{(i)}) \geq U(\theta^{(i)}) + \frac{(\nabla_\theta U(\theta^{(i)})^\top d^{(i)})^2}{2M\|d^{(i)}\|_2^2}. \tag{D.3}$$

Now, since $\|\nabla_\theta U(\theta^{(i)}) - d^{(i)}\|_2 \leq K\epsilon$,[1] we have that $\frac{(\nabla_\theta U(\theta^{(i)})^\top d^{(i)})^2}{\|d^{(i)}\|_2^2} \geq \|\nabla_\theta U(\theta^{(i)})\|_2^2 - K^2\epsilon^2$.[2] Substituting this result into Eqn. (D.3) we obtain

$$U(\theta^{(i)} + \alpha^* d^{(i)}) \geq U(\theta^{(i)}) + \frac{\|\nabla_\theta U(\theta^{(i)})\|_2^2 - K^2\epsilon^2}{2M}.$$

Our algorithm uses an η-optimal line search to choose the step-size α. In case the line search is successful (i.e., our η-optimal line-search found a policy better than the

[1]Recall that we are considering the case of policy gradient. Thus we have that the descent directions $d^{(i)}$ are equal to the estimated gradients $\nabla_\theta \hat{U}(\theta^{(i)})$.

[2]We use the fact that for any vectors a, b if $\|a - b\|_2 \leq K\epsilon$, then we have $a^\top a - \frac{(a^\top b)^2}{b^\top b} \leq K^2\epsilon^2$. Proof:

$$\begin{aligned}
& a^\top a - \frac{(a^\top b)^2}{b^\top b} \\
\leq\ & a^\top a - \frac{(a^\top b)^2}{b^\top b} + \|\frac{bb^\top}{b^\top b}a - b\|_2^2 \\
=\ & a^\top a - \frac{(a^\top b)^2}{b^\top b} + \frac{a^\top bb^\top a}{b^\top b} - 2\frac{b^\top bb^\top a}{b^\top b} + b^\top b \\
=\ & a^\top a - 2b^\top a + b^\top b \\
=\ & \|a - b\|_2^2 \\
\leq\ & K^2\epsilon^2.
\end{aligned}$$

current policy), we have that $U(\theta^{(i+1)}) \geq U(\theta^{(i)} + \alpha^* d^{(i)}) - \eta$, and thus:

$$U(\theta^{(i+1)}) \geq U(\theta^{(i)}) + \frac{\|\nabla_\theta U(\theta^{(i)})\|_2^2 - K^2\epsilon^2}{2M} - \eta. \tag{D.4}$$

Hence as long as $\|\nabla_\theta U(\theta)\|_2^2 > 2K^2\epsilon^2 + 2M\eta$ holds, a successful line search is guaranteed to improve the objective by a finite amount, namely $K^2\epsilon^2$. Since the objective is bounded from above, this can happen only a finite number of times. Thus—if the algorithm's line search is successful in every iteration—the algorithm converges to a region where Eqn. (D.1) holds.

On the other hand, if the line search is not successful, then the objective could not be improved by η along the line. This means $\frac{\|\nabla_\theta U(\theta^{(i)})\|_2^2 - K^2\epsilon^2}{2M} - \eta \leq 0$ and the current point $\theta^{(i)}$ already satisfies Eqn. (D.1). □

Proposition 33. *Let $\epsilon > 0$ be such that for all $t = 0, 1, \ldots, H$ we have $\|\frac{df_t}{ds} - \frac{d\hat{f}_t}{ds}\|_2 \leq \epsilon$ and $\|\frac{df_t}{da} - \frac{d\hat{f}_t}{da}\|_2 \leq \epsilon$. Let all first and second order derivatives of $\{f_t(\cdot,\cdot)\}_{t=0}^H$, $\{\hat{f}_t(\cdot,\cdot)\}_{t=0}^H$, $\pi_\theta(\cdot)$, $R(\cdot)$ be bounded by a constant C. Let n, p, q denote the dimensionality of the state space, action space and policy parameterization space respectively. Then there exist constants K and M—which are expressible as a function only of n, p, q, the constant C, and the MDP's horizon H—such that the following holds:*

$$\|\nabla_\theta U(\theta) - \nabla_\theta \hat{U}(\theta)\|_2 \leq K\epsilon, \tag{D.5}$$

$$\|\nabla_\theta^2 U(\theta)\|_2 \leq M. \tag{D.6}$$

Proof of Proposition 33. To derive Eqn. (D.5), we use the chain rule and triangle

inequality to obtain:

$$\begin{aligned}
&\left\| \frac{dh_t}{d\theta} - \frac{d\hat{h}_t}{d\theta} \right\|_2 \leq \\
&\left\| \frac{df_{t-1}}{ds} + \frac{df_{t-1}}{da}\frac{d\pi_\theta}{ds} \right\|_2 \left\| \frac{dh_{t-1}}{d\theta} - \frac{d\hat{h}_{t-1}}{d\theta} \right\|_2 \\
&+ \left\| \frac{df_{t-1}}{da} - \frac{d\hat{f}_{t-1}}{da} \right\|_2 \left\| \frac{d\pi_\theta}{d\theta} \right\|_2 \\
&+ \left\| \frac{df_{t-1}}{ds} - \frac{d\hat{f}_{t-1}}{ds} \right\|_2 \left\| \frac{d\hat{h}_{t-1}}{d\theta} \right\|_2 \\
&+ \left\| \frac{df_{t-1}}{da} - \frac{d\hat{f}_{t-1}}{da} \right\|_2 \left\| \frac{d\pi_\theta}{ds} \right\|_2 \left\| \frac{d\hat{h}_{t-1}}{d\theta} \right\|_2 .
\end{aligned}$$

Using the boundedness assumptions, we have that there exist constants C_1, C_2, C_3, which depend only on C, H, n, p and q, such that:

$$\begin{aligned}
\left\| \frac{dh_t}{d\theta} - \frac{d\hat{h}_t}{d\theta} \right\|_2 \leq\ & C_1 \left\| \frac{dh_{t-1}}{d\theta} - \frac{d\hat{h}_{t-1}}{d\theta} \right\|_2 + C_2\epsilon \\
& + (\epsilon + C_3\epsilon) \left\| \frac{d\hat{h}_{t-1}}{d\theta} \right\|_2 . \qquad \text{(D.7)}
\end{aligned}$$

Since $\left\| \frac{d\hat{h}_{t-1}}{d\theta} \right\|_2$ is a finite sum and product of derivatives of $\hat{f}$ and π_θ (with number of terms bounded as a function of H, n, p and q only), $\left\| \frac{d\hat{h}_{t-1}}{d\theta} \right\|_2$ is bounded by a function of C, H, n, p and q. Expanding the recursion of Eqn. (D.7) (with base case $\left\| \frac{dh_1}{d\theta} - \frac{d\hat{h}_1}{d\theta} \right\|_2 \leq \left\| \frac{df_{t-1}}{da} - \frac{d\hat{f}_{t-1}}{da} \right\|_2 \left\| \frac{d\pi_\theta}{d\theta} \right\|_2 \leq pqC\epsilon$) then gives us that there exists a constant K_1, which depends only on C, H, n, p and q, such that

$$\left\| \frac{dh_t}{d\theta} - \frac{d\hat{h}_t}{d\theta} \right\|_2 \leq K_1\epsilon. \qquad \text{(D.8)}$$

From the chain rule for derivatives we have that $\nabla_\theta U(\theta) = \sum_{t=0}^{H} \nabla_s R \frac{dh_t}{d\theta}$, and

$\nabla_\theta \hat{U}(\theta) = \sum_{t=0}^{H} \nabla_s R \frac{d\hat{h}_t}{d\theta}$. Thus we have that

$$\left\| \nabla_\theta U(\theta) - \nabla_\theta \hat{U}(\theta) \right\|_2 \leq \sum_{t=0}^{H} \|\nabla_s R\|_2 \left\| \frac{dh_t}{d\theta} - \frac{d\hat{h}_t}{d\theta} \right\|_2 . \tag{D.9}$$

Substituting Eqn. (D.8) into Eqn. (D.9) proves Eqn. (D.5). Eqn. (D.6) follows directly from the fact that the entries in $\nabla^2_\theta U(\theta)$ are a finite sum and product of first and second derivatives of $\{f_t\}_{t=0}^{H}, \pi_\theta$ and R; and the number of terms in the sum is a function of H, n, p and q only. □

Theorem 12 is readily proved using Propositions 32 and 33. For completeness we give the proof below.

Proof of Theorem 12. The conditions of Proposition 33 are all satisfied by the assumptions in Theorem 12. Thus we can apply Proposition 33 and obtain that there exist constants K and M—which are a function of C, the MDP's horizon H, the dimensionalities of the state space n, of the action space p and of the policy parameterization space q only—such that the following holds:

$$\|\nabla_\theta U(\theta) - \nabla_\theta \hat{U}(\theta)\|_2 \leq K\epsilon, \tag{D.10}$$

$$\|\nabla^2_\theta U(\theta)\|_2 \leq M. \tag{D.11}$$

Since $R \leq R_{\max}$, we have that $U(\theta) \leq HR_{\max}$, and thus U is bounded from above. So all conditions of Proposition 32 are satisfied. Applying Proposition 32 proves the theorem. □

Appendix E

Trajectory Learning Algorithm

As described in Section 7.3, our algorithm (approximately) solves

$$\max_{\boldsymbol{\tau},\Sigma^{(\cdot)},\mathbf{d}} \log \mathbb{P}(\mathbf{y}, \boldsymbol{\rho}, \boldsymbol{\tau} \; ; \; \Sigma^{(\cdot)}, \mathbf{d}). \tag{E.1}$$

Then, once our algorithm has found $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$, it finds the most likely hidden trajectory, namely the trajectory $\mathbf{z}$ that maximizes the joint likelihood of the observed trajectories $\mathbf{y}$ and the observed prior knowledge about the ideal trajectory $\boldsymbol{\rho}$ for the learned parameters $\boldsymbol{\tau}, \mathbf{d}, \Sigma^{(\cdot)}$.

To optimize Eq. (E.1), we alternatingly optimize over $\Sigma^{(\cdot)}$, $\mathbf{d}$ and $\boldsymbol{\tau}$. Section 7.3 provides the high-level description, below we provide the detailed description of our algorithm.

1. Initialize the parameters to hand-chosen defaults. A typical choice: $\Sigma^{(\cdot)} = I$, $d_i^k = \frac{1}{3}$, $\tau_j^k = \lceil j \frac{T-1}{N^k-1} \rceil$.
2. E-step for latent trajectory: For the current setting of $\boldsymbol{\tau}, \Sigma^{(\cdot)}$ run a (extended) Kalman smoother to find the distributions for the latent states, $\mathcal{N}(\mu_{t|T-1}, \Sigma_{t|T-1})$.
3. M-step for latent trajectory: Update the covariances $\Sigma^{(\cdot)}$ using the standard EM update.
4. E-step for the time indexing (using hard assignments): run dynamic time warping to find $\boldsymbol{\tau}$ that maximizes the joint probability $\mathbb{P}(\bar{\mathbf{z}}, \mathbf{y}, \boldsymbol{\rho}, \boldsymbol{\tau})$, where $\bar{\mathbf{z}}$ is

fixed to $\mu_{t|T-1}$, namely the mode of the distribution obtained from the Kalman smoother.

5. M-step for the time indexing: estimate $\mathbf{d}$ from $\boldsymbol{\tau}$.
6. Repeat steps 2-5 until convergence.

E.1 Steps 2 and 3 details—EM for non-linear dynamical systems

Steps 2 and 3 in our algorithm correspond to the standard E and M steps of the EM algorithm applied to a non-linear dynamical system with Gaussian noise. For completeness we provide the details below.

In particular, we have:

$$z_{t+1} = f(z_t) + \omega_t, \quad \omega_t \sim \mathcal{N}(0, Q),$$
$$y_{t+1} = h(z_t) + \nu_t, \quad \nu_t \sim \mathcal{N}(0, R).$$

In the E-step, for $t = 0..T-1$, the Kalman smoother computes the parameters $\mu_{t|t}$ and $\Sigma_{t|t}$ for the distribution $\mathcal{N}(\mu_{t|t}, \Sigma_{t|t})$, which is the distribution of z_t conditioned on all observations up to and including time t. Along the way, the smoother also computes $\mu_{t+1|t}$ and $\Sigma_{t+1|t}$. These are the parameters for the distribution of z_{t+1} given only the measurements up to time t. Finally, during the backward pass, the parameters $\mu_{t|T-1}$ and $\Sigma_{t|T-1}$ are computed, which give the distribution for z_t given all measurements.

After running the Kalman smoother (for the E-step), we can use the computed quantities to update Q and R in the M-step. In particular, we can compute[1]:

[1]The notation $\mathcal{D}f(z)$ is the Jacobian of f evaluated at z.

$$
\begin{aligned}
\delta\mu_t &= \mu_{t+1|T-1} - f(\mu_{t|T-1}), \\
A_t &= \mathcal{D}f(\mu_{t|T-1}), \\
L_t &= \Sigma_{t|t} A_t^\top \Sigma_{t+1|t}^{-1}, \\
P_t &= \Sigma_{t+1|T-1} - \Sigma_{t+1|T-1} L_t^\top A_t^\top - A_t L_t \Sigma_{t+1|T-1}, \\
Q &= \frac{1}{T}\sum_{t=0}^{T-1} \delta\mu_t \delta\mu_t^\top + A_t \Sigma_{t|T-1} A_t^\top + P_t, \\
\delta y_t &= y_t - h(\mu_{t|T-1}), \\
C_t &= \mathcal{D}h(\mu_{t|T-1}), \\
R &= \frac{1}{T}\sum_{t=0}^{T-1} \delta y_t \delta y_t^\top + C_t \Sigma_{t|T-1} C_t^\top.
\end{aligned}
$$

E.2 Steps 4 and 5 details—Dynamic time warping

In Step 4 our goal is to compute $\bar{\boldsymbol{\tau}}$ as:

$$
\begin{aligned}
\bar{\boldsymbol{\tau}} &= \arg\max_{\boldsymbol{\tau}} \log \mathbb{P}(\bar{\mathbf{z}}, \mathbf{y}, \boldsymbol{\rho}, \boldsymbol{\tau}; \Sigma^{(\cdot)}, \mathbf{d}) \\
&= \arg\max_{\boldsymbol{\tau}} \log \mathbb{P}(\mathbf{y}|\bar{\mathbf{z}}, \boldsymbol{\tau}) \mathbb{P}(\boldsymbol{\rho}|\bar{\mathbf{z}}) \mathbb{P}(\bar{\mathbf{z}}) \mathbb{P}(\boldsymbol{\tau}) \qquad \text{(E.2)} \\
&= \arg\max_{\boldsymbol{\tau}} \log \mathbb{P}(\mathbf{y}|\bar{\mathbf{z}}, \boldsymbol{\tau}) \mathbb{P}(\boldsymbol{\tau})
\end{aligned}
$$

where $\bar{\mathbf{z}}$ is the mode of the distribution computed by the Kalman smoother (namely, $\bar{z}_t = \mu_{t|T-1}$) and Eq. (E.2) made use of independence assumptions implied by our model (see Figure 7.2). Again, using independence properties, the log likelihood above can be expanded to:

$$
\begin{aligned}
&\bar{\boldsymbol{\tau}} = \\
&\arg\max_{\boldsymbol{\tau}} \sum_{k=0}^{M-1} \sum_{j=0}^{N^k-1} \left[\ell(y_j^k | \bar{z}_{\tau_j^k}, \tau_j^k) + \ell(\tau_j^k | \tau_{j-1}^k) \right]
\end{aligned} \qquad \text{(E.3)}
$$

Note that the inner summations in the above expression are independent—the

likelihoods for each of the M observation sequences can be maximized separately. Hence, in the following, we will omit the k superscript, as the algorithm can be applied separately for each sequence of observations and indices.

At this point, we can solve the maximization over $\boldsymbol{\tau}$ using a dynamic programming algorithm known in the speech recognition literature as dynamic time warping (Sakoe & Chiba, 1978) and in the biological sequence alignment literature as the Needleman-Wunsch algorithm (Needleman & Wunsch, 1970). For completeness, we provide the details for our setting below.

We define the quantity $\mathcal{Q}(s,t)$ to be the maximum obtainable value of the first $s+1$ terms of the inner summation if we choose $\tau_s = t$.

For $s = 0$ we have:

$$\mathcal{Q}(0,t) = \ell(y_0|\bar{z}_{\tau_0}, \tau_0 = t) + \ell(\tau_0 = t) \tag{E.4}$$

And for $s > 0$:

$$\begin{aligned}\mathcal{Q}(s,t) = \ell(y_s|\bar{z}_{\tau_s}, \tau_s = t) \\ + \max_{\tau_1,\ldots,\tau_{s-1}} \left[\ell(\tau_s = t|\tau_{s-1}) \right. \\ \left. + \sum_{j=0}^{s-1} \left[\ell(y_j|\bar{z}_{\tau_j}, \tau_j) + \ell(\tau_j|\tau_{j-1})\right]\right]\end{aligned}$$

The latter equation can be written recursively as:

$$\begin{aligned}&\mathcal{Q}(s,t) = \ell(y_s|\bar{z}_{\tau_s}, \tau_s = t) + \\ &\max_{t'} \left[\ell(\tau_s = t|\tau_{s-1} = t') + \mathcal{Q}(s-1,t')\right]\end{aligned} \tag{E.5}$$

The equations (E.4) and (E.5) can be used to compute $\max_t \mathcal{Q}(N^k - 1, t)$ for each observation sequence (and the maximizing solution, $\boldsymbol{\tau}$), which is exactly the maximizing value of the inner summation in Eq. (E.3). The maximization in Eq. (E.5) can be restricted to the relevant values of t'. In our application, we only allow $t' \in \{t-3, t-2, t-1\}$. As is common practice, we typically restrict the time-index assignments to a fixed-width band around the default, equally-spaced alignment. In

our case, we only compute $\mathcal{Q}(s,t)$ if $2s - C \leq t \leq 2s + C$, for fixed C.

In Step 5 we compute the parameters $\mathbf{d}$ using standard maximum likelihood estimates for multinomial distributions.

Bibliography

Abbeel, P., Coates, A., Quigley, M., & Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. *NIPS 19*.

Abbeel, P., Dolgov, D., Ng, A. Y., & Thrun, S. (2008). Apprenticeship learning for motion planning with application to parking lot navigation. *Proc. IROS*.

Abbeel, P., Ganapathi, V., & Ng, A. Y. (2006a). Learning vehicular dynamics with application to modeling helicopters. *NIPS 18*.

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. *Proc. ICML*.

Abbeel, P., & Ng, A. Y. (2005a). Exploration and apprenticeship learning in reinforcement learning. *Proc. ICML*.

Abbeel, P., & Ng, A. Y. (2005b). Learning first order Markov models for control. *NIPS 18*.

Abbeel, P., Quigley, M., & Ng, A. Y. (2006b). Using inaccurate models in reinforcement learning. *Proc. ICML*.

Amit, R., & Mataric, M. (2002). Learning movement sequences from demonstration. *Proc. ICDL*.

An, C. H., Atkeson, C. G., & Hollerbach, J. M. (1988). *Model-based control of a robot manipulator*. MIT Press.

Anderson, B., & Moore, J. (1989). *Optimal control: Linear quadratic methods*. Prentice-Hall.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997a). Locally weighted learning for control. *AI Review*.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997b). Locally weighted learning for control. *Artificial Intelligence Review*, *11*.

Atkeson, C. G., & Schaal, S. (1997). Robot learning from demonstration. *Proc. 14th International Conference on Machine Learning* (pp. 12–20). Morgan Kaufmann.

Bagnell, J., Ng, A. Y., & Schneider, J. (2001). *Solving uncertain Markov decision problems* (Technical Report). Robotics Institute, Carnegie Mellon University.

Bagnell, J., & Schneider, J. (2001). Autonomous helicopter control using reinforcement learning policy search methods. *International Conference on Robotics and Automation*. IEEE.

Bertsekas, D. P. (2001). *Dynamic programming and optimal control*, vol. 1. Athena Scientific. 2nd edition.

Bertsekas, D. P., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Athena Scientific.

Billingsley, P. (1995). *Probability and Measure*. Wiley Interscience.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. *Proc. UAI*.

Brafman, R. I., & Tennenholtz, M. (2002). R-max, a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*.

Calinon, S., Guenter, F., & Billard, A. (2007). On learning, representing and generalizing a task in a humanoid robot. *IEEE Trans. on Systems, Man and Cybernetics, Part B*.

Coates, A., Abbeel, P., & Ng, A. Y. (2008). Learning for control from multiple demonstrations (Full version). http://heli.stanford.edu/icml2008.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory*. Wiley.

Demiris, J., & Hayes, G. (1994). A robot controller using learning by imitation.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Statistical Society*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *JAIR*.

Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2008). Path planning for autonomous driving in unknown environments. *Proceedings of the Eleventh International Symposium on Experimental Robotics(ISER-08)*. Athens, Greece.

Dullerud, G. E., & Paganini, F. (2000). *A course in robust control theory: A convex approach*, vol. 36 of *Texts in Applied Mathematics*. Springer - New York.

Durrett, R. (1995). *Probability:Theory and Examples*. Duxbury Press.

Gahinet, P., Nemirovski, A., Laub, A., & Chilali, M. (1995). *Lmi control toolbox*. Natick, MA.

Gavrilets, V., Martinos, I., Mettler, B., & Feron, E. (2002a). Control logic for automated aerobatic flight of miniature helicopter. *AIAA Guidance, Navigation and Control Conference*.

Gavrilets, V., Martinos, I., Mettler, B., & Feron, E. (2002b). Flight test and simulation results for an autonomous aerobatic helicopter. *AIAA/IEEE Digital Avionics Systems Conference*.

Gavrilets, V., Mettler, B., & Feron, E. (2004). Human-inspired control logic for automated maneuvering of miniature helicopter. *Journal of Guidance, Control, and Dynamics*, *27*, 752–759.

Gelb, A. (Ed.). (1974). *Applied optimal estimation*. MIT Press.

Ghahramani, Z. (1998). Learning dynamic Bayesian networks. In *Adaptive processing of sequences and data structures*, 168–197. Springer-Verlag.

Gillespie, T. (1992). *Fundamentals of vehicle dynamics*. SAE.

Hogan, N. (1984). An organizing principle for a class of voluntary movements. *J. of Neuroscience*, *4*, 2745–2754.

Intel (2001). Opencv libraries for computer vision. http://www.intel.com/research/mrl/research/opencv/.

J., R., & P., A. (1998). Learning to drive a bicycle using reinforcement learning and shaping. *Proc. ICML*.

Jacobson, D. H., & Mayne, D. Q. (1970). *Differential dynamic programming*. Elsevier.

Jones, L. K. (1992). A simple lemma on greedy approximation in hilbert space and convergence rates for projection pursuit regresson and neural network training. *The Annals of Statistics*.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *JAIR*.

Kakade, S., Kearns, M., & Langford, J. (2003). Exploration in metric state spaces. *Proc. ICML*.

Kakade, S., & Ng, A. Y. (2005). Online bounds for Bayesian algorithms. *NIPS 17*.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, *82*, 35–45.

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. *Proc. IJCAI*.

Kearns, M., Mansour, Y., & Ng, A. Y. (1999). Approximate planning in large POMDPs via reusable trajectories. *NIPS 12*.

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning journal*.

Kohl, N., & Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion.

Kolter, J. Z., Abbeel, P., & Ng, A. Y. (2008a). Hierarchical apprenticeship learning with application to quadruped locomotion. *Neural Information Processing Systems 20*.

Kolter, J. Z., Rodgers, M. P., & Ng, A. Y. (2008b). A control architecture for quadruped locomotion over rough terrain. *Proceedings of the International Conference on Robotics and Automation*.

Kuniyoshi, Y., Inaba, M., & Inoue, H. (1994). Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *T-RA*, *10*, 799–822.

La Civita, M., Papageorgiou, G., Messner, W. C., & Kanade, T. (2006). Design and flight testing of a high-bandwidth $\mathcal{H}_\infty$ loop shaping controller for a robotic helicopter. *Journal of Guidance, Control, and Dynamics*, *29*, 485–494.

Lefferts, E., Markley, F. L., & Shuster, M. D. (1982). Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics*.

Leishman, J. (2000). *Principles of helicopter aerodynamics*. Cambridge University Press.

Listgarten, J. (2006). *Analysis of sibling time series data: alignment and difference detection*. Doctoral dissertation, University of Toronto.

Listgarten, J., Neal, R. M., Roweis, S. T., & Emili, A. (2005). Multiple alignment of continuous time series. *NIPS 17*.

Ljung, L. (1986). *System identification: theory for the user*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Manne, A. (1960). Linear programming and sequential decisions. *Management Science*, *6*.

Mettler, B., Tischler, M., & Kanade, T. (1999). System identification of small-size unmanned helicopter dynamics. *American Helicopter Society, 55th Forum*.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pueger, M., Stanek, G., Stavens, D., Vogt, A., & Thrun, S. (2008). The stanford entry in the urban challenge. *Journal of Field Robotics*.

Moore, K. L. (1998). Iterative learning control: An expository overview. *Applied and Computational Controls, Signal Processing, and Circuits*.

Morimoto, J., & Atkeson, C. G. (2002). Minimax differential dynamic programming: An application to robust biped walking. *NIPS 14*.

Morimoto, J., & Doya, K. (2001). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*.

Neal, R., & Hinton, G. (1999). A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, 355–368. MIT Press.

Needleman, S., & Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*

Neu, G., & Szepesvari, C. (2007). Apprenticeship learning using inverse reinforcement learning and gradient methods. *Proc. UAI*.

Ney, H., Essen, U., & Kneser, R. (1994). On structuring probabilistic dependencies in stochastic language modeling. *Computer Speech and Language*, *8*.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2004a). Autonomous inverted helicopter flight via reinforcement learning. *ISER*.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. *Proc. ICML*.

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. *NIPS 14*.

Ng, A. Y., Kim, H. J., Jordan, M., & Sastry, S. (2004b). Autnonomous helicopter flight via reinforcement learning. *NIPS 16*.

Ng, A. Y., & Russell, S. (2000). Algorithms for inverse reinforcement learning. *Proc. ICML*.

Nilim, A., & El Ghaoui, L. (2005). Robust solutions to Markov decision problems with uncertain transition matrices. *Operations Research*.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kauffman.

Pomerleau, D. (1989). Alvinn: An autonomous land vehicle in a neural network. *NIPS 1*.

Precup, D., Sutton, R. S., & Singh, S. (1998). Theoretical results on reinforcement learning with temporally abstract options. *Proc. ECML*.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*.

Ramachandran, D., & Amir, E. (2007). Bayesian inverse reinforcement learning. *Proc. IJCAI*.

Ratliff, N., Bagnell, J., & Zinkevich, M. (2006). Maximum margin planning. *Proc. ICML*.

Ratliff, N., Bradley, D., Bagnell, J. A., & Chestnutt, J. (2007). Boosting structured prediction for imitation learning. *Neural Information Processing Systems 19*.

Roberts, J. M., Corke, P. I., & Buskey, G. (2003). Low-cost flight control system for a small autonomous helicopter. *IEEE Int'l Conf. on Robotics and Automation*.

Rockafellar, R. T. (1970). *Convex analysis*. Princeton University Press.

Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*.

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. *Proc. ICML*.

Saripalli, S., Montgomery, J., & Sukhatme, G. (2003). Visually-guided landing of an unmanned aerial vehicle.

Satia, J. K., & Lave, R. L. (1973). Markov decision processes with uncertain transition probabilities. *Operations Research*.

Saul, L. K., & Jordan, M. I. (1999). Mixed memory Markov models: decomposing complex stochastic processes as mixtures of simpler ones. *Machine Learning*, *37*.

Schaal, S., & Atkeson, C. G. (1994). Robot learning by nonparametric regression. *Proc. IROS*.

Seddon, J. (1990). *Basic helicopter aerodynamics*. AIAA Education Series. America Institute of Aeronautics and Astronautics.

Smart, W. D., & Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. *Proc. ICML*.

Stevens, B. L., & Lewis, F. L. (2003). *Aircraft control and simulation*. Wiley and Sons. 2nd edition.

Sutton, R. S. (1995). TD models: Modeling the world at a mixture of time scales. *Proc. ICML*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Syed, U., & Schapire, R. E. (2008). A game-theoretic approach to apprenticeship learning. *NIPS 20*.

Tischler, M., & Cauffman, M. (1992). Frequency response method for rotorcraft system identification: Flight application to BO-105 couple rotor/fuselage dynamics. *Journal of the American Helicopter Society*.

Uno, Y., Kawato, M., & Suzuki, R. (1989). Formation and control of optimal trajectory in human multijoint arm movement. minimum torque-change model. *Biological Cybernetics*, *61*, 89–101.

Vapnik, V. N. (1998). *Statistical learning theory*. John Wiley & Sons.

White, C. C., & Eldeib, H. K. (1994). Markov decision processes with imprecise transition probabilities. *Operations Research*.

Williams, D. (1991). *Probability with Martingales*. Cambridge Mathematical Textbooks.

Zhou, K., Doyle, J., & Glover, K. (1995). *Robust and optimal control*. Prentice Hall.