MOTOR SKILL LEARNING WITH LOCAL TRAJECTORY METHODS

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

Sergey Levine
March 2014

This dissertation is online at: http://purl.stanford.edu/qg636tb5216

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.

Vladlen Koltun, Primary Adviser

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.

Patrick Hanrahan, Co-Adviser

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.

Percy Liang

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Motor or sensorimotor skills are behaviors that require close coordination of motor control with feedback from the environment. This includes a wide range of human and animal behaviors, such as locomotion and manipulation. Constructing effective and generalizable motor skills is crucial for creating naturalistic, versatile, and effective virtual characters and robots. However, constructing such motor skills manually requires extensive engineering and, quite often, nontrivial insights into the structure of the behavior. For a robot or virtual character to reproduce a motor skill repertoire as wide as that of a human being, the required engineering effort would be staggering. A more scalable approach is to acquire motor skills autonomously, by combining concepts from optimal control with machine learning. In this thesis, I discuss several algorithms based on local trajectory methods that can be used to construct motor skills for walking, running, swimming, traversal of uneven terrain, and recovery from strong perturbations. I show how example demonstrations can be used to automatically learn the objective or goal of the skill, and how local trajectory methods can be used to train general-purpose controllers, represented by large neural networks, without the need for extensive manual engineering or domain knowledge about the task at hand.

Acknowledgments

First and foremost, I would like to warmly thank my mentor and adviser Vladlen Koltun for his advice and encouragement during my time at Stanford. I would also like to thank my committee members for their feedback and appraisal of my work: Pat Hanrahan, Percy Liang, Noah Goodman, and Mark Cutkosky.

I would also like to thank my long time collaborator Zoran Popović for his support and advice, as well as the rest of my collaborators, in no particular order: Jovan Popović, Jack Wang, Philipp Krähenbühl, Alexis Haraux, Yongjoon Lee, Taesung Park, and Sebastian Thrun.

Many of the experiments presented in this thesis were made possible by the MuJoCo simulator, which was generously provided to me by Emanuel Todorov, Yuval Tassa, and Tom Erez.

The Stanford graphics lab, which has been my home during my graduate studies, has made that time intellectually stimulating and collegial, and I would like to acknowledge all of the students in the graphics lab for making Stanford computer science such a wonderful place to study and work.

Finally, I would like to thank my family, and especially my wife Kathy, for having patience, offering support, and serving as an undaunted source of encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> "We have a brain for one reason and one reason only: to produce adaptable and complex movements."
>
> DANIEL WOLPERT

Endowing robots and virtual characters with complex, responsive sensorimotor skills has been a persistent research goal for decades. However, despite impressive advances in optimal control, reinforcement learning, motion planning, and optimization, neither robots nor virtual characters posses behavioral repertoires that can compare to the breadth, flexibility, and responsiveness of human or animal motor control. Human motor control has been observed to achieve near-optimal performance for well-practiced skills (Woodworth, 1899; Nelson, 1983; Meyer et al., 1988; Pandy et al., 1990), exhibits very close, task-specific integration of perception and action (McLeod and Dienes, 1996), and can adapt quickly to changes in the task or environment (Thoroughman and Shadmehr, 2000).

Standard motor control architectures, particularly in robotics, typically reflect a modular decomposition of the task, as illustrated in Figure 1.1. One example decomposition might involve a perception module to analyze the environment, a motion planning module to choose a suitable path, and a control mechanism to choose motor commands to realize this trajectory (Brooks, 1986; Kalakrishnan et al., 2011a). On the other hand, human motion skills are often the product of habituation and learned or innate heuristics (Rosenbaum,

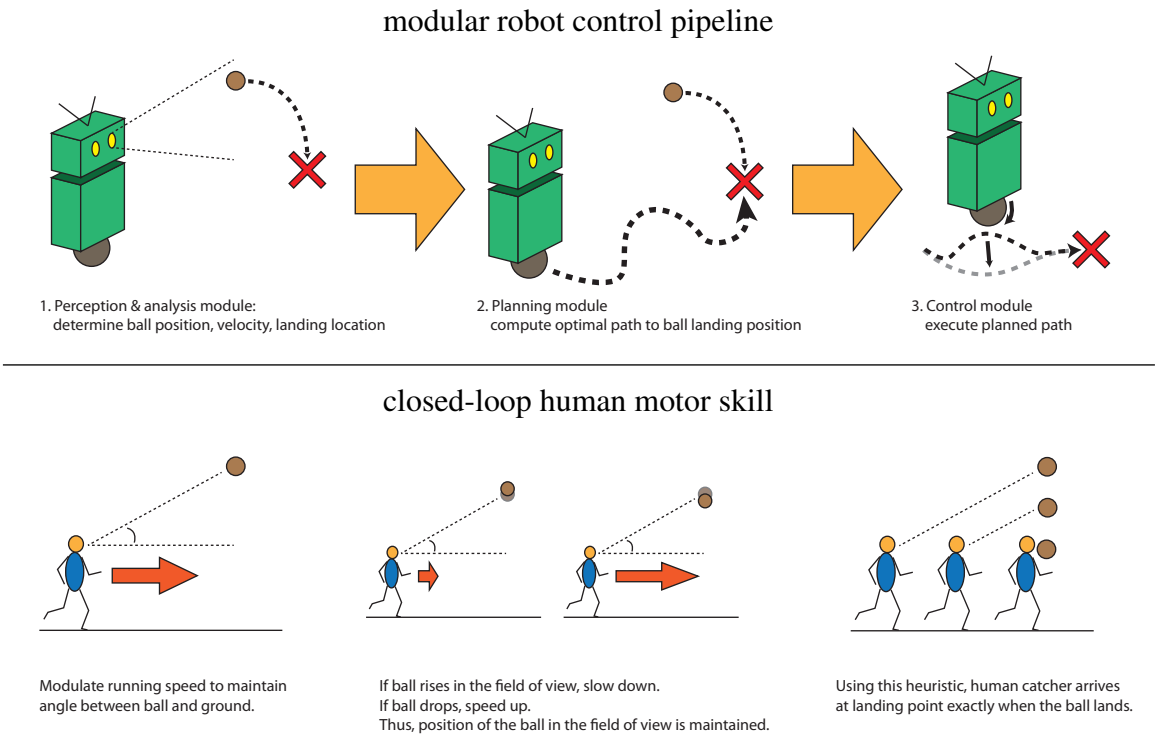


Figure 1.1: Example decomposition of the ball catching problem representative of standard techniques in robotics (top), compared to a closed-loop motor skill that closely integrates perception and action (bottom). When catching the ball, a human need only keep its position fixed in the field of view while running (McLeod and Dienes, 1996).

1991; Salmon, 1995). For example, when catching a ball flying through the air, human athletes do not attempt to predict the velocity, air resistance, and landing location of the ball, but instead employ a simple feedback rule illustrated in Figure 1.1. Although this feedback rule is specific to catching flying objects, it is a parsimonious solution to the catching problem that can generalize better to new situations. In contrast, the more standard pipeline is vulnerable to errors in the initial estimation of the ball's velocity and requires potentially expensive online planning.

As discussed in Chapter 2, some of the most successful controllers for robots and virtual characters have used hand-crafted control heuristics to similarly "short-circuit" the pipeline between perception and control, replacing the complex chain of perception and control modules with simple, task-specific feedback rules that map sensory signals (such as proprioception) directly to actions. The subsumption architecture proposed by Brooks

in 1986 in fact incorporates this idea as one of its key features, and offers a robust and practical alternative to the modular methods more commonly employed in robotics and AI (Brooks, 1986).

Unfortunately, the construction of effective and generalizable control laws, such as the ball-catching heuristic discussed above, poses a significant challenge. Engineering such controllers by hand takes significant effort, with entire research papers (Yin et al., 2007) and doctoral theses (Geyer, 2005) dedicated entirely to the subject of a single motor skill. Reproducing the entire repertoire of human or animal motion through such manual engineering is a staggering undertaking, and even then would fail to produce an adaptive system that can easily acquire new skills, which humans can do quite easily. To scale to the range of motor skills that might be required for a robot to act in the real world, or for a realistic virtual character to interact with simulated environments, we require methods that can acquire motor skills automatically.

In this thesis, I discuss algorithms for learning such motor skills automatically from example demonstrations and experience. One of the central themes is the development of methods that are general and make few assumptions about either the system being controlled or the form of the motor skill. Although the proposed algorithms can be used with any motor skill representation, I demonstrate their performance using general-purpose neural networks, which map the state of the system directly to the actions that should be taken. This representation allows learning a wide variety of motor skills with flexible and responsive strategies, including behaviors that switch between multiple strategies depending on the state. Although learning neural networks for motor control has long been regarded as a promising direction (Lewis et al., 1999), the high dimensionality and nonlinearity of such controllers has made them notoriously difficult to apply to larger problems and has restricted their use to small, simple domains (Miller et al., 1990). The proposed methods tackle these challenges by using trajectory optimization to guide motor skill learning, as summarized in the following section.

1.1 Motor Skills and Optimal Control

In order to learn a motor skill, we must first define its objective. This objective is usually defined by means of a reward or cost function that specifies the utility of each state-action pair, such that states and actions that accomplish the task successfully are given a low cost (or high reward), while those that are considered undesirable are given a high cost (or low reward). This function can range from a simple objective, such as a locomotion cost that merely rewards moving forward at the desired velocity, to a complex combination of numerous terms, such as a cost function for manipulating a cup of coffee that considers its distance from the desired position, its orientation (to avoid spillage), distance to obstacles for safety, and the total exerted effort.

The cost or reward function in general provides only delayed feedback about the utility of each action, and greedily minimizing the cost typically results in a poor control strategy. For example, the cost function for basketball might only be low when the ball is actually in the basket, but might otherwise be uninformative about how to place it there. This means that a learning or optimization algorithm must be able to consider the future consequences of current actions. Optimal control and reinforcement learning concern themselves primarily with this problem, and we will draw heavily on ideas from these fields in developing the motor skill learning algorithms. A review of related work in reinforcement learning and optimal control is provided in Chapter 2, while this section provides a high-level overview.

Motor skill learning is an instance of the policy search problem, where the goal is to optimize the parameters of a parametric policy with respect to a cost. Policy search methods have a long history in reinforcement learning, going back to early work on policy gradient and actor-critic methods (Williams, 1992; Sutton and Barto, 1998). Many reinforcement learning algorithms attempt to discover effective policies without knowledge of the system dynamics, in a manner analogous to trial and error (Deisenroth et al., 2013). Although such model-free methods have advanced considerably in recent years, their application is generally limited to policies with under a hundred parameters (Deisenroth et al., 2013). General-purpose controllers like neural networks can have thousands or even millions or parameters, making them exceptionally difficult to learn with model-free techniques. Instead, successful applications of model-free learning have focused on carefully designed

policy classes with a small number of parameters (Ijspeert et al., 2003; Kohl and Stone, 2004; Theodorou et al., 2010). Such policy classes necessarily trade off representational power and generality for computational tractability.

As an alternative to model-free learning, model-based policy search methods make use of a known or learned model of system dynamics. When the dynamics are known, the total policy cost can be optimized with gradient-based methods by directly differentiating the policy objective with respect to the parameters, for example by means of finite differences (Peters and Schaal, 2008). However, this requires differentiating through the dynamics function over the course of many time steps, which is known to produce catastrophically unstable gradients for non-toy problems. These difficulties are similar to those encountered when training recurrent neural networks, which is well known to be an exceptionally difficult problem (Pascanu and Bengio, 2012).

The algorithms presented in this thesis instead make use of trajectory optimization to avoid the need to differentiate the policy objective directly with respect to its parameters. Trajectory optimization addresses the simpler problem of minimizing the cost of individual trajectories from a single initial configuration, directly with respect to the actions taken at each point in time. Unlike policy search, the parameters of the optimization (the actions) decompose over time, allowing dynamic programming algorithms to be employed. However, the result of trajectory optimization is an open-loop, nonstationary control law, which does not generalize to new situations. For example, a trajectory optimized for walking on one terrain cannot easily be ported onto a slightly different terrain without rerunning the optimization.

The algorithms I will discuss make use of trajectory optimization as an intermediate step toward optimizing parameteric motor skills. By using trajectory optimization to handle the temporal structure of the problem, I show how the policy objective can be approximately factorized over time steps, avoiding the instabilities associated with direct differentiation of the policy objective through time. Trajectory optimization also provides an informed and efficient method for discovering low-cost executions of the task, removing the need to rely on the trial and error techniques of model-free reinforcement learning, though at the cost of requiring a model of the system dynamics. Finally, the use of trajectory optimization allows the proposed algorithms to be initialized with known good trajectories,

which can come from example demonstrations.

1.2 Learning Motor Skills from Demonstrations

As discussed previously, humans exhibit impressive versatility in acquiring and using sophisticated sensorimotor skills. However, even humans struggle to acquire complex motor skills on their own. Instead, we often benefit from the instruction of other humans in the form of coaching and example demonstrations, for example when we learn to play a sport. In similar fashion, the methods I will describe are all able to utilize successful example executions of the desired task, which can be supplied either by a human expert, or synthesized in an offline planning phase in a known, controlled environment.

Besides guiding the policy search, example demonstrations can be used to obviate the need to hand-specify the cost function for the desired behavior. While cost functions for some motor skills are simple and easy to design, other behaviors may require complex functions that trade off competing goals in non-intuitive ways. For example, the previously mentioned coffee cup manipulation cost requires trading off success at grasping the cup, prevention of spilling its contents, avoidance of obstacles, and the minimization of effort. The relative weighting of each term in the scalar cost function can have large effects on the resulting behavior, and choosing appropriate weights is not always intuitive. Other motor skills, such as complex acrobatics, may not even present obvious criteria to add to the cost without significant study and engineering effort.

I will show how trajectory optimization techniques similar to those used in the proposed policy search algorithms can also be adapted to learn cost or reward functions directly from examples, through a technique called inverse optimal control (IOC) or inverse reinforcement learning (IRL). These learned cost functions can then be used with any policy search method to recover the corresponding motor skill. This approach to motor skill learning is known to generalize much better than directly mimicking the example demonstration, a process known as behavior cloning. The process of learning the cost implicitly encodes the prior knowledge that the demonstration is *goal-directed* and near-optimal with respect to some unknown cost function (Abbeel and Ng, 2004). As discussed in previous work, this knowledge drastically affects the posterior belief about the demonstrated behavior, leading

to substantially better inference for goal-directed tasks (Ziebart, 2010).

Once a cost function has been either learned or specified, the example demonstrations can be employed to guide the policy search procedure and help discover the desired motor skill. Since the policy search methods discussed in this thesis all begin with a trajectory optimization step, the examples can serve to directly initialize this trajectory, immediately placing the policy search into the neighborhood of a good solution. This option is not generally available to standard policy search methods, which operate entirely in the space of policy parameters, and therefore cannot be initialized with an example trajectory.

1.3 Applications of Motor Skill Learning

Direct applications of motor skills and motor skill learning include robotics (Peters and Schaal, 2008), computer graphics (Treuille et al., 2007), and biomechanics (Geyer and Herr, 2010). More broadly, techniques developed for learning motor control policies can be applied to other domains that benefit from reinforcement learning and optimal control, including power grid management (Gayme and Topcu, 2012), energy generation (Kolter et al., 2012), autonomous vehicle control (Abbeel et al., 2006; Ross et al., 2013), and operations research (Mannor et al., 2003).

As discussed previously and shown in Figure 1.1, the classical control pipeline in robotics consists of a series of independent modules that use AI techniques to solve distinct subproblems with considerable accuracy. In contrast, motor skills tightly couple perception directly to control, taking a shortcut around the standard pipeline. This approach can relax the demands on accurately modeling the environment and, as in the case of the ball catching heuristic, can offer superior generalization, but only when the right control feedback law is learned. A practical application of learned motor skills in robotics might utilize a set of motor skills as primitives for a high-level planner, as suggested in some previous work (Stulp et al., 2012; Levine et al., 2011a).

In computer graphics, responsive and naturalistic motor skills are essential for creating believable virtual characters. Such motor skills may be kinematic, which means they set the configuration of the character's joints directly, or dynamic, which means they set forces and torques on the joints within a physical simulation. The methods described in

this thesis are applicable to both contexts, though results are presented only for dynamic controllers. Kinematic animation offers some unique challenges, since although the control task is simplified tremendously by the lack of inertia and gravity, it becomes more difficult to produce motions that appear natural and physically plausible. In previous work, I proposed statistical techniques for constructing dynamical systems that avoid the complexity of real physics, while still restricting the character to realistic and natural motions (Levine et al., 2012; Levine and Popovic, 2012). Such techniques can be combined with the algorithms in this thesis to learn realistic motor skills for virtual characters without the use of physics simulation.

Whether the animation is realized with or without physical simulation, most current graphics applications use hand-engineered control mechanisms. Due to a lower reliability burden and the availability of perfect perception and dynamics models, the graphics control pipeline is substantially simpler, and might consist of a state machine that switches between a number of prepared motion clips based on simple observations about the environment. However, as the fidelity of virtual worlds increases, accompanying improvements in graphics and displays (such as the Oculus Rift virtual reality device (Oculus VR, 2014)) as well as input technologies (such as the Microsoft Kinect (Zhang, 2012)), the demand for character fidelity will also grow. Current virtual worlds often use behaviors that are more symbolic than realistic. For example, opening a door is represented by simply touching the handle. With increased fidelity, users will expect more lifelike virtual characters. When a user can interact with the virtual world with his own hands, for example by means of the Kinect, he will expect a character to be able to shake his hand. The increased demand for fidelity in character control will in turn make hand-engineering every controller impractical, and learning methods, such as those presented in this thesis, will become necessary to endow virtual characters with the motor skill repertoire necessary to achieve the requisite level of fidelity.

Simulation of human-like motion skills is also an active area of research in biomechanics, where such simulations can help us understand the mechanisms of human motor control and predict the outcomes of treatments (Reinbolt et al., 2011; John et al., 2012; Webb et al., 2012). Predictive simulation of changes in human behavior in response to perturbations and surgical interventions have so far been limited by the lack of access to

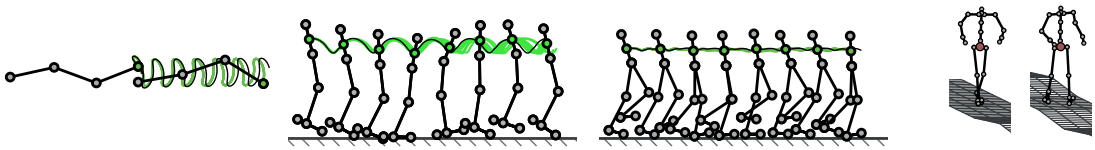


Figure 1.2: Visualization of the dynamical systems used in the experimental evaluation: the planar swimmer, hopper, bipedal walker, and 3D humanoid. The evaluation includes flat ground, uneven slopes, and recoveries from lateral pushes.

realistic human-like control mechanisms. Learning realistic motor skills from human example demonstrations could serve as the first step toward bridging this gap, and paving the way for powerful predictive simulations in biomechanics.

The algorithms presented in this thesis are evaluated on a number of simulated locomotion tasks, including planar swimming, hopping, bipedal walking, traversal of uneven terrain, and recovery from very strong lateral perturbations. In addition, a realistic evaluation on a 3D humanoid running task with motion captured example demonstrations is provided. These tasks present a number of challenges that are very difficult to overcome with previous methods, and are, at a high level, broadly representative of the applications described in this section. Each task is performed on an underactuated system, which means that there are more total degrees of freedom than there are actuators, since the global position and orientation cannot be controlled directly. Underactuation requires the controller to avoid dangerous states that can result in an unrecoverable failure, such as falling to the ground during running, and in practice results in a very narrow window of acceptable actions that both maintain the stability of the system and succeed at the desired task.

Example visualizations of these tasks are shown in Figure 1.2. All of the tasks exceed the dimensionality of feasible dynamic programming problems, with the simplest task (swimming) having 10 state dimensions, and the most complex (3D running) having 63. To avoid the need to engineer task-specific policy classes, I will employ general-purpose neural networks to represent the learned motor skill. However, this causes the dimensionality of the policy parameterization to also exceeds the feasible dimensionality for model-free reinforcement learning. The smallest policies used in the evaluation have 70 policy parameters, and the largest have as many as 25,000. In a series of evaluations, I will show that only the simplest of these tasks can be solved with previous policy search methods.

1.4 Contributions and Thesis Overview

In this thesis, I will present a family of guided policy search algorithms for learning motor skills, as well as a method for learning cost functions for motor skills from example demonstrations. All of the algorithms draw on a common framework of trajectory-based stochastic optimal control presented in Chapter 3, which also presents the notation used in the remainder of the thesis, technical background on optimal control and reinforcement learning, and a discussion of the connections between optimal control and probabilistic inference. The main contributions of the thesis are then presented in Chapters 4 and 5.[1]

Chapter 4 uses this stochastic optimal control framework to develop an efficient algorithm for learning cost functions for high-dimensional continuous tasks. The proposed algorithm uses a probabilistic generative model of expert behavior, which provides a principled and effective way to learn even from suboptimal demonstrations. This chapter also presents experimental evaluations of this inverse optimal control algorithm on a simulated driving task and on humanoid running.

Chapter 5 then presents three different algorithms for learning motor skills, centered on the common theme of using trajectory optimization as a tool for utilizing example demonstrations and discovering low-cost executions of the task, thus guiding the policy search process. Although the presented algorithms build on one another, each has its own strengths and weaknesses. Experimental evaluations are presented at the end of the chapter, and include bipedal walking on uneven terrain, push recovery, and 3D humanoid running.

[1] This work has previously appeared in the following publications: Levine and Koltun (2012); Park and Levine (2013); Levine and Koltun (2013a,b); Levine (2013)

Chapter 2

Related Work

In this chapter, I discuss previous work in machine learning, robotics, and computer graphics that relates to the execution and learning of motor skills. Since motor skill learning can be regarded as an instance of policy search, reinforcement learning techniques provide us with a logical starting point in Section 2.1. The algorithms presented in this thesis also extensively utilize techniques from trajectory optimization and optimal control. I therefore review relevant optimal control methods in Section 2.2. The proposed methods also make use of example executions of the desired task, a technique known as learning from demonstration. When example demonstrations are available, they can not only help learn the motor skill, but can also be used to infer the goal of the desired skill, by means of inverse optimal control (IOC) or inverse reinforcement learning (IRL). Previous learning from demonstration methods, including IOC and IRL techniques, are reviewed in Section 2.3. In additional to learning techniques, I discuss a variety of relevant motor skill representation in Section 2.4, followed by a discussion of previous applications in robotics and character animation.

2.1 Reinforcement Learning

Reinforcement learning is a powerful framework for controlling dynamical systems that can be used to learn control policies with minimal user intervention. A complete review of reinforcement learning is outside the scope of this thesis, but may be found across several

popular textbooks (Sutton and Barto, 1998; Bertsekas, 2001) and recent survey articles (Kaelbling et al., 1996; Kober et al., 2013). This section provides a brief overview, with particular emphasis on motor skills and techniques relevant to the proposed algorithms.

Modern reinforcement learning has its roots in the dynamic programming work of Richard Bellman (Bellman and Dreyfus, 1962) and Richard Sutton's work on temporal difference learning (Sutton, 1988), which represent the model-based and model-free branches of reinforcement learning, respectively. In model-based reinforcement learning, a model of the system dynamics is used together with dynamic programming to compute a control policy with minimal cost or maximum reward. This is usually done by estimating the value function (sometimes called a cost-to-go function), which specifies the minimum additional cost that an optimal policy will accrue when starting from a particular state. When the optimal value function is known, the optimal policy can simply choose the action that leads to the state with the best value (Bertsekas, 2001). In model-free temporal difference learning, as well as related algorithms such as Q-learning and SARSA (Watkins and Dayan, 1992; Singh and Sutton, 1996), the value function is instead estimated from repeated interactions with the environment by means of bootstrapping, where the observed reward and the next predicted value are used to estimate the total value of the visited states, and the values are adjusted online to drive the error of such predictions to zero (Sutton and Barto, 1998).

Both dynamic programming and temporal difference learning require the user to choose a suitable representation for the value function. In small, discrete spaces, the value function can be represented exactly by recording a distinct value for each state. Continuous state spaces, such as those encountered in motor control, must make use of function approximators. In general, function approximators cannot represent the value function exactly, and the resulting error can can both model-based and model-free methods to not converge, or converge to extremely poor policies (Boyan and Moore, 1995; Sutton, 1996). Recently, algorithms have been proposed to mitigate the convergence issues (Maei et al., 2010), but a good representation of the value function is still required to learn an effective policy. While the best features for value representation tend to be highly problem-specific, as illustrated in the seminal work on learning backgammon strategies (Tesauro, 1995), several authors have proposed general-purpose features that tend to achieve better results than naïve discretization (Sutton, 1996; Konidaris and Osentoski, 2008).

The root cause for the difficulty of representing the value function in high-dimensional, continuous state spaces is termed the curse of dimensionality (Bellman and Dreyfus, 1962), and refers to the fact that any uniform discretization schemes requires a number of discrete values that is exponential in the dimensionality of the space. The curse of dimensionality cannot be overcome in general, though it can be mitigated with domain-specific features that take advantage of the structure of the task. For more complex tasks, such knowledge is scarce, since the shape of the value function is rarely an intuitive and simple consequence of the structure of the task.

Direct policy search methods provide an alternative to value function estimation that sidesteps the curse of dimensionality, learning the control policy directly (Williams, 1992). Such methods can scale to tasks with high state space dimensionality (Peters and Schaal, 2008), and have been applied successfully to real robotic applications, as discussed in Section 2.5. Many policy search methods aim to estimate the gradient of the expected total reward of a policy with respect to its parameters, typically by using on-policy samples (Williams, 1992; Sutton et al., 1999; Baxter et al., 2001; Peters and Schaal, 2008), and then take a small step in the direction of the gradient to improve the policy. The central challenges of this approach are the choice of learning rate, which can be critical in the highly peaked landscape of the expected cost, the high variance of a gradient estimated from too few samples, and the problem of local optima.

More recent policy search methods have tried to mitigate these challenges by using importance sampling to include samples from previous policies, thus reducing variance (Tang and Abbeel, 2010), and by reframing the problem as a reward-weighted regression that can be solved in closed form, without the need to set a learning rate (Kober and Peters, 2009; Theodorou et al., 2010). However, these solutions do not significantly alleviate the challenge of local optima, which in practice limits such model-free techniques to small, task-specific policy classes with few parameters (Deisenroth et al., 2013). To train general-purpose controllers such as neural networks, more powerful algorithms are required that can overcome this limitation.

Local optima pose a serious problem for direct model-free policy search because all such methods rely on trial and error for improving the policy: improvement is achieved by increasing the probability of low-cost samples, and decreasing the probability of high-cost

samples. But if the initial policy only visits high cost regions, as is the case with a randomly initialized neural network controlling an underactuated physical system, the probability of randomly discovering a successful execution of the task can be very low. For example, it is profoundly unlikely for a random neural network to produce a stable walking gait. One way to overcome this challenge is to replace trial-and-error with model-based improvement. Early model-based policy search algorithms attempted to directly differentiate the policy objective through the dynamics (Ng and Jordan, 2000), but this approach is known to be exceedingly unstable for nonlinear, high dimensional dynamics, for the same reason that recurrent neural networks suffer from exploding or vanishing gradients (Pascanu and Bengio, 2012). More recent methods have attempted to mitigate this problem by learning models from a particular, well-behaved class, such as Gaussian processes (GPs), which then allow the gradient of the expected cost to be computed in closed form (Deisenroth and Rasmussen, 2011). But such methods require policy classes where the expectation under the GP can be evaluated in closed form, and generally cannot represent highly nonlinear, discontinuous dynamics, which are very common in locomotion tasks.

In contrast, the methods I will discuss in this thesis decompose the policy search problem across time steps by using trajectory optimization to handle the difficult problems of exploration and delayed rewards, while the policy optimization is reduced to reproducing the behavior of the optimized trajectories at each point in time. In the next section, I review some techniques from optimal control and trajectory optimization that are related to the methods presented in this thesis.

2.2 Optimal Control and Trajectory Optimization

Optimal control deals with the problem of choosing optimal actions in the context of a dynamical system, typically with known dynamics and a known cost function. Although the applications and techniques of optimal control have many parallels with reinforcement learning and in particular with dynamic programming, they carry an emphasis on optimization and continuous domains (Todorov, 2006b). A complete survey of optimal control is again outside the scope of this thesis, and this section instead discusses several recent methods that are relevant to motor control and motor skill learning.

A central problem in optimal control is the planning of an optimal trajectory. Foundational work in this field was initially carried out by Pontryagin (Pontryagin, 1964), whose maximum principle paralleled the development of dynamic programming by Richard Bellman. However, because the maximum principle concerns itself with individual trajectories, rather than a value function or policy defined on the entire state space, it does not suffer from the curse of dimensionality, and can be practically applied to problems in large, continuous state and action spaces. Practical dynamic programming methods can also be constructed so as to focus on only a single trajectory, by considering only its local neighborhood in a differential sense by means of a Taylor expansion. Differential dynamic programming (DDP) is a popular method that does precisely this, by employing a second order expansion of the dynamics and cost function, in a manner analogous to Newton's method (Jacobson and Mayne, 1970). Modern variants of DDP have proposed using only first order dynamics for efficiency (Li and Todorov, 2004), spatial rather than time indexing (Kolter et al., 2008), as well as generalization to receding horizons (Tassa et al., 2007).

Although such trajectory-centric optimal control methods can be used to efficiently optimize a sequence of actions in continuous domains, they do not produce a generalizable and portable policy. Typically, controllers that rely on trajectory optimization must reoptimize the current trajectory in real time as changes in the environment are observed. This technique is called model predictive control (MPC) (Diehl et al., 2009), and has experienced a resurgence in recent years as improvements in computational power have permitted larger and more realistic applications (Abbeel et al., 2006; Tassa et al., 2012). However, MPC does not take advantage of an offline training phase, and must instead "rediscover" the desired behavior via trajectory optimization at runtime. While this is effective for simple behaviors, the vulnerability of the method to local optima and the strict computational limitations of requiring real time performance limit the method's applicability. Furthermore, trajectory optimization requires the current state of the system to be fully observed, which is not always practical at test time. Instead, the methods proposed in this thesis only use trajectory optimization at training time, in a controlled environment that can be assumed to be fully modeled and observed. The algorithms then train a parametric motor skill that, through its representation, can accomodate any desired constraint on the observability of the environment.

Although there are no prior methods that use trajectory optimization to guide the search for a parametric policy, previous work has proposed to reuse trajectories optimized via DDP in a nonparametric fashion to create a stationary policy (Atkeson and Morimoto, 2002; Atkeson and Stephens, 2008). Such nonparametric methods can be viewed as a variant of nearest-neighbor lookup, and though they provide superior generalization compared to simply rerunning a previously optimized trajectory, as shown in the experiments in Section 4.6, parametric neural network policies trained by the proposed algorithms tend to generalize significantly better.

2.3 Learning from Demonstrations and Inverse Optimal Control

One of the advantages of using trajectory optimization to guide policy search is that example demonstrations can be used to initialize such trajectories, mitigating the challenges associated with local optima. Using example demonstrations to help learn effective controllers is an active area of research called learning from demonstration, imitiation learning, or apprenticeship learning (Argall et al., 2009).[1]

Although example demonstrations cannot in general be used to directly initialize arbitrary parametric policies, previous work has proposed to use specific trajectory-centric policy classes that can be initialized directly from examples via supervised learning (Ijspeert et al., 2002; Coates et al., 2008). Other authors have proposed methods that involve an interactive "teaching" procedure, where human operators correct the mistakes of a learned policy (Ross et al., 2011; Jain et al., 2013).

Methods that aim to mimic the example demonstrations directly are sometimes referred to as behavior cloning, to distinguish them from inverse reinforcement learning (IRL) or inverse optimal control (IOC) methods. IRL/IOC is the problem of learning the goal of the task, represented by a cost or reward function, directly from example demonstrations of an optimal or near-optimal behavior (Ng and Russell, 2000). IOC is known to offer superior

[1] Although there are subtle distinctions in the meaning of these terms, they are often used interchangeably.

generalization, due to the implicit integration of the prior belief that the example demonstration is a near-optimal goal-driven behavior, rather than an arbitrary mapping from states to actions (Ziebart, 2010). The choice of the structure of the reward or cost function, defined for example through a set of features, provides an additional avenue for incorporating intuitive domain knowledge about the task, and allows more portable cost or reward functions to be learned. Although most IOC algorithms use a linear parameterization, where the cost is linear parameterization of a set of features, more flexible methods with learned features (Levine et al., 2010b) or nonlinear function approximators (Levine et al., 2011b) have also been proposed.

Unfortunately, the IOC problem can be significantly more challenging in practice than the "forward" problem of recovering the optimal policy from a learned reward. Early IOC and IRL algorithms often required finding the optimal policy for the current cost function within the inner loop of an optimization procedure that iteratively modified the current cost, in order to make the resulting optimal policy resemble the example demonstrations (Ng and Russell, 2000; Abbeel and Ng, 2004; Ratliff et al., 2006; Ramachandran and Amir, 2007; Ziebart, 2010). Algorithms for learning cost functions in large or continuous spaces had previously been restricted either to recovering a cost matrix in the linear-quadratic regulator setting (Ziebart, 2010; Boyd et al., 1994), or recovering a value function only, and then computing the cost function (Dvijotham and Todorov, 2010). The problem with the latter approach is that it requires features for representing the value function, rather than the cost function. This requires specifying value function features which, as discussed in Section 2.1, can be exceptionally difficult, and does not permit the structure of the learned cost to be controlled by means of cost features, as is the case in standard IOC methods.

In Chapter 4, I present an IOC algorithm that overcomes these challenges by working exclusively in the neighborhood of the example trajectories. This removes the need to find an optimal policy on the entire state space, and in practice enables complex cost functions to be learned even in very high dimensional state spaces, such as humanoid locomotion. This algorithm can be used to learn a cost function for a desired behavior using only the example demonstration. Together, the learned cost and the example can then be used to recover the motor skill.

2.4 Motor Skill Representations

Regardless of how the motor skill is learned or whether example demonstrations are used, any algorithm that learns motor skills must choose a representation that can capture the desired behaviors. In this section, I will briefly review a few of the representations proposed in the robotics and computer graphics literature. These representations can broadly be categorized as task-specific, trajectory-centric, or general-purpose.

Task-specific representations are hand-engineered controllers for particular tasks, where the specific parameters of the controller are tuned to maximize performance. Locomotion controllers are most commonly constructed in this fashion, with a long history of various controller classes in both computer graphics (Raibert and Hodgins, 1991; Yin et al., 2007; Coros et al., 2010) and robotics (Morimoto et al., 2004; Kohl and Stone, 2004). Since such controllers typically have a small number of parameters, a growing trend in recent years is the use of evolutionary algorithms such as covariance matrix adaptation (CMA) (Hansen and Ostermeier, 1996) or the cross-entropy method (Rubinstein and Kroese, 2004) to optimize their parameters (Wampler and Popović, 2009; Wang et al., 2009; Stulp and Sigaud, 2012). However, while such controllers can be used to create effective motor skills, and can be relatively easily to train, they require extensive hand-engineering for each class of behaviors, and therefore do not scale to the vast motion repertoires displayed by humans and animals.

A more general representation for motor skills is a generalized notion of a trajectory. Splines are a popular kinematic trajectory representation in robotics (Sciavicco and Siciliano, 2007; Peters and Schaal, 2008). Dynamic movement primitives (DMPs) provide a more dynamic representation, where the parameters include both the shape of the trajectory, and the stiffness with which each waypoint in the trajectory is tracked (Ijspeert et al., 2003). Because of their ability to vary the gains, DMPs can represent more dynamic behaviors such as swinging a tennis raquet (Kober and Peters, 2009). More recently, a probabilistic variant of DMPs was proposed to allow for more principled compositionality and adaptation (Paraschos et al., 2013). Although this class of controllers offers greater generality than hand-engineered control laws, it is highly restrictive, since it cannot represent a control strategy that makes distinct decisions based on state. For example, while

DMPs can be used to represent a tennis swing, they cannot be used to represent a motor skill that chooses when to swing the raquet in response to the position of the ball, or a behavior that can choose between a backhand and forehand swing based on the situation. In practice, this relegates such policies to be the bottom layer in a hand-engineered control hierarchy.

Fully general policy classes that can represent any behavior have received comparatively little attention, due to their high dimensionality. PILCO, a model-based algorithm that uses Gaussian processes to model the system dynamics, operates exclusively on radial basis function networks, which are general function approximators (Deisenroth and Rasmussen, 2011). However, because this algorithm relies on representing the dynamics with a Gaussian process with squared exponential kernels, it is limited to tasks where the dynamics are highly smooth, and in practice has not been applied to locomotion or other behaviors that involve contact. Neural network controllers have long been viewed as a promising class of control policies (Lewis et al., 1999), but their high dimensionality and nonlinearity has made them exceptionally difficult to apply to larger problems (Miller et al., 1990). Real-world applications have largely been restricted to grasping and reaching motions (Lampe and Riedmiller, 2013), as well as simple experiments in computational neuroscience (Herbort et al., 2009).

In this work, I show how large neural network controllers can be trained to perform complex tasks like humanoid locomotion, by using trajectory optimization to reduce the problem of choosing the network weights to supervised regression. This alleviates many of the challenges traditionally associated with using neural networks in reinforcement learning, such as the difficulty of choosing the learning rate, local optima due to bad initial policies, and instability of gradients propagated through time, although the challenges of local optima and nonconvexity cannot be eliminated entirely. Furthermore, by using fully general policy classes, the proposed methods can also learn highly complex policies that choose between a variety of strategies depending on the situation, as demonstrated by the push recovery task in Chapter 5, where the policy must choose between different learned recovery strategies.

2.5 Applications in Robotics and Computer Graphics

Although the classic perception-control pipeline described in Chapter 1 remains the dominant paradigm for robot control, learned and hand-engineered motor skills have found a number of applications. A substantial body of work in robotics focuses on learned and hand-designed locomotion controllers (Morimoto et al., 2004; Kohl and Stone, 2004; Byl and Tedrake, 2008; Whitman and Atkeson, 2009), often utilizing a mixture of hand-designed policy classes, reinforcement learning, and simple heuristics like the zero momentum point. Dynamic movement primitives have also been applied extensively for a wide variety of motor skills, including games such as table tennis and ball-in-cup (Schaal, 2003; Kober and Peters, 2009) and reaching and grasping behaviors (Righetti et al., 2014).

In addition to one-shot or rhythmic motor skills, several previous works have examined the integration of multiple motor skills into more complex compound behaviors (Konidaris et al., 2010; Stulp et al., 2012; Daniel et al., 2012). However, although the advantages of hierarchical control have been recognized for a long time, effective methods for constructing hierarchical or compound skills automatically have yet to be developed, and the previously mentioned techniques in large part rely on the user to manually specify an appropriate decomposition of the task. While the algorithms described in this thesis are not explicitly designed for learning hierarchies, the experimental evaluation in Section 4.6 shows how a single neural network can be trained on the push recovery task to capture a variety of distinct recovery strategies, together with the rules necessary to choose an appropriate strategy depending on the current state. This suggests that learning neural network controllers may in fact be a promising direction for developing methods that acquire hierarchical or compound control policies naturally and automatically.

In the context of character animation, the branch of computer graphics concerned with the simulation of virtual humans and animals, methods for executing and learning motor skills can be broadly divided into kinematic and dynamic methods. Kinematics methods set the configuration of the character (such as joint angles) directly, often by using motion capture training data. The majority of characters seen in games and films are animated manually using kinematic methods, with simple hand-designed state machines to choose

the appropriate motion (Johansen, 2009). Learned kinematic controllers suitable for interactive applications were first popularized in the context of motion graphs. Motion graphs transform the continuous kinematic control problem into a discrete graph traversal problem, by arranging example motion clips into a graph structure, with nodes corresponding to short clips (such as individual steps in a gait), and edges corresponding to a transition between two clips, usually performed by means of linear blending (Kovar et al., 2002; Arikan and Forsyth, 2002; Lee et al., 2002). While traversal of motion graphs was initially performed by means of standard graph search algorithms (Safonova and Hodgins, 2007), real-time controllers were soon developed that used reinforcement learning on the discrete state space of the motion graph (Lee and Lee, 2006; Treuille et al., 2007; Lo and Zwicker, 2008), and were applied to tasks ranging from fencing (Wamplers et al., 2010) to animal locomotion (Wamplers et al., 2013) to generation of hand gestures in natural conversation (Levine et al., 2009, 2010a). Unlike robotic controllers, such techniques do not suffer from the curse of dimensionality, because the graph provides a small, discrete space on which the learning can be performed. The downside of such techniques is their extravagant data requirements, and the comparatively modest agility and responsiveness due to the rigid graph structure. More recently, continuous data-driven kinematic controllers have been developed by means of approximate dynamic programming (Lee et al., 2010b) and nonlinear dimensionality reduction (Levine et al., 2012). While kinematic character controllers tend to be significantly more expressive than inidividual motor skills in robotics, they are still limited in scope due to the curse of dimensionality, which limits their ability to process input from the environment. For example, a graph-based controller for walking around an obstacle must be parameterized by the obstacle's location, and the additional parameters are usually incorporated by means of discretization (Lee et al., 2009). Like in robotics, hierarchical methods have been proposed that use a high-level planner to schedule task-specific low-level kinematic controllers (Levine et al., 2011a).

In addition to kinematic control, extensive work has been conducted in character animation on controlling dynamic characters in the context of a physical simulation, starting with hand-engineered task-specific controllers (Raibert and Hodgins, 1991; Yin et al., 2007). In recent years, the use of optimizations methods such as CMA (Hansen and Ostermeier, 1996) has become a popular technique for adapting the parameters of such controllers to

maximize their performance on a particular task or in a particular environment (Wang et al., 2009, 2010). Other works proposed to construct dynamic character controllers by using standard control techniques to track motion capture data (Sok et al., 2007; Muico et al., 2009), and recent years have seen the convergence of such methods with stochastic optimization of controller parameters (Liu et al., 2010; Lee et al., 2010a).

Although character animation methods have many parallels with robotic control, they tend to place a heavier emphasis on naturalistic appearance and ease of use, and rely more heavily on example demonstrations. As shown in Section 4.6, the proposed algorithms can use example demonstrations to create highly naturalistic humanoid locomotion controllers. Together with automatic inference of behavioral goals from demonstration via IOC, these techniques could in the future be developed into a general, data-efficient, and user-friendly system for learning motor skills for virtual characters.

Chapter 3

Trajectory Optimization and Stochastic Optimal Control

The field of reinforcement learning has traditionally addressed the policy search task with trial-and-error reinforcement learning algorithms. Such methods adjust the policy to increase the probability of previously observed good outcomes, and decrease the probability of bad outcomes. However, effective application of reinforcement learning often requires the policy class to be chosen carefully, so that a good policy can be found without falling into poor local optima. The development of new, specialized policy classes is an active area that has provided substantial improvements on real-world systems (Ijspeert et al., 2003; Paraschos et al., 2013). This specialization is necessary because most model-free policy search methods can only feasibly be applied to policies with under a hundred parameters (Deisenroth et al., 2013). However, it restricts the types of behaviors that can be learned, and requires additional engineering.

On the other hand, model-based trajectory optimization offers a general method for planning actions, and can been used to generate a wide range of behaviors from high-level cost functions (Todorov and Li, 2005; Toussaint, 2009; Tassa et al., 2012; Schulman et al., 2013). However, such methods do not learn generalizable parametric policies, and therefore require expensive online replanning and cannot directly handle partially observed environments.

Later in this thesis, I will present algorithms that adapt trajectory optimization to the

task of policy search, overcoming some of the challenges of reinforcement learning while preserving the benefits of learning parametric policies. In this chapter, I will describe differential dynamic programming, a classical trajectory optimization method (Jacobson and Mayne, 1970), and describe a probabilistic variant that can be used to optimize trajectory distributions in the framework of stochastic optimal control. This probabilistic variant will be used in the following chapters to develop algorithms both for policy search and for learning cost functions from example demonstrations.

3.1 Preliminaries

Trajectory optimization is the task of optimizing a trajectory $\tau = \{(\hat{\mathbf{x}}_1, \hat{\mathbf{u}}_1), \ldots, (\hat{\mathbf{x}}_T, \hat{\mathbf{u}}_T)\}$ with respect to some cost function $\ell(\mathbf{x}_t, \mathbf{u}_t)$, where the total cost of the trajectory is given by $\ell(\tau) = \sum_{t=1}^{T} \ell(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$. For example, the cost function for a walking behavior might prescribe a high cost to states where the current velocity deviates from the desired velocity. The relationship between the current state and action and the next state is governed by the system dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, and the optimization is typically performed over the actions $\mathbf{u}_t$. The states $\mathbf{x}_t$ are vector-valued quantities that describe the state of the system in phase space – that is, they contain sufficient information to fully describe its configuration, so that any future behavior of the system is independent of past states when the current state is known. This is sometimes called the Markov property. In the case of a physical system such as a robot, the state typically includes the configuration of the body, as well as the corresponding velocities. The actions $\mathbf{u}_t$ are the decisions made at each point in time,[1] which may represent joint torques or muscle contractions. The dynamics specify how the state changes in response to the passage of time and the application of torques. In a physical system, this corresponds to integrating the equations of motion.

The trajectory emerges as a consequence of the chosen actions $\mathbf{u}_t$, and the states that then result from applying the system dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. In the case of continuous trajectory optimization, it is common to assume that the dynamics are approximately Gaussian, with the mean given by $f_t(\mathbf{x}_t, \mathbf{u}_t)$ and a fixed (potentially time-varying) covariance

[1] Discrete time steps are used for simplicity. In general, many of the same methods can be derived in continuous time, and then solved via discretization.

$\mathbf{F}_t$, such that $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_t(\mathbf{x}_t, \mathbf{u}_t), \mathbf{F}_t)$. As discussed in the following section, the symmetry of the Gaussian distribution makes it possible to ignore the covariance $\mathbf{F}_t$ when deriving the optimal actions under the Laplace approximation, which corresponds to a linear-quadratic expansion of the dynamics and cost function.

In dealing with sequential decision problems such as trajectory optimization, it is often useful to consider the value function $V_t(\mathbf{x}_t)$, which gives the minimum additional cost that will be accumulated in all time steps from 1 to T if starting from initial state $\mathbf{x}_t$. In the deterministic case, the value function can be defined as $V_t(\mathbf{x}_t) = \min_{\mathbf{u}_t,\ldots,\mathbf{u}_T} \ell(\mathbf{x}_t, \mathbf{u}_t)$, while in the case of stochastic dynamics, the cost is replaced by the expectation of the cost. The value function can be defined recursively as $V(\mathbf{x}_t) = \min_{\mathbf{u}_t} \ell(\mathbf{x}_t, \mathbf{u}_t) + E[V(\mathbf{x}_{t+1})]$, which yields a convenient dynamic programming algorithm for recursively computing the optimal value function. For convenience, we can define the Q-function as $Q_t(\mathbf{x}_t, \mathbf{u}_t) = \ell(\mathbf{x}_t, \mathbf{u}_t) + E[V(\mathbf{x}_{t+1})]$, so that the value function is given simply as $V(\mathbf{x}_t) = \min_{\mathbf{u}_t} Q_t(\mathbf{x}_t, \mathbf{u}_t)$.

While computing the value function with dynamic programming is an efficient method for obtaining the optimal trajectory, the value function cannot be represented exactly in continuous domains with general, nonlinear dynamics. Instead, we must approximate the value function, and the choice of approximation can have profound effects on the quality of the solution. In the following section, I describe a classic iterative algorithm that approximates the value function locally around a nominal trajectory.

3.2 Differential Dynamic Programming

Differential dynamic programming (DDP) is a local trajectory optimization algorithm analogous to Newton's method. The goal is to find a trajectory $\tau = \{(\hat{\mathbf{x}}_1, \hat{\mathbf{u}}_1), \ldots, (\hat{\mathbf{x}}_T, \hat{\mathbf{u}}_T)\}$ that is consistent with the deterministic dynamics $f_t(\mathbf{x}_t, \mathbf{u}_t)$ and locally optimal with respect to the cost function $\ell(\tau) = \sum_{t=1}^{T} \ell(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$. In addition to producing an optimized trajectory, DDP also produces a linear feedback policy at each time step that can be used to stabilize the trajectory under mild perturbations. It has been showed that this method produces a locally optimal linear policy even under Gaussian dynamics, so long as the dynamics mean $f_t(\mathbf{x}_t, \mathbf{u}_t)$ is used during optimization. As with Newton's method, the algorithm has quadratic convergence and is guaranteed to find a local optimum (Jacobson

and Mayne, 1970).

Starting from some nominal trajectory τ, DDP uses a second-order Taylor expansion of the value function around the trajectory at each time step to compute a new, improved trajectory, recomputes the Taylor expansion, and repeats the process (Jacobson and Mayne, 1970). Iterating this procedure to convergence produces a locally optimal trajectory. In the derivation below, we will initially assume that the dynamics are deterministic, allowing us to replace $E[V(\mathbf{x}_{t+1})]$ in the Q-function by $V(f_t(\mathbf{x}_t, \mathbf{u}_t))$. In the case of nondeterministic Gaussian dynamics, the equations can be shown to be identical due to the symmetry of the Gaussian distribution (Todorov and Li, 2005).

At each time step, we can use a second order Taylor expansion around $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$ to approximate the Q-function $Q_t(\mathbf{x}_t, \mathbf{u}_t) = \ell(\mathbf{x}_t, \mathbf{u}_t) + V_t(f_t(\mathbf{x}_t, \mathbf{u}_t))$ as following:

$$Q_t(\mathbf{x}_t, \mathbf{u}_t) \approx Q_t(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \frac{1}{2}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^{\mathrm{T}} \ell_{\mathbf{xu},\mathbf{xu}t}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^{\mathrm{T}} \ell_{\mathbf{xu}t} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^{\mathrm{T}} \left[f_{\mathbf{xu}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{xu}t} + V_{\mathbf{x}t+1} \cdot f_{\mathbf{xu},\mathbf{xu}t}\right]\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^{\mathrm{T}} f_{\mathbf{xu}t}^{\mathrm{T}} V_{\mathbf{x}t+1}, \quad (3.1)$$

where we assume without loss generality that $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) = 0$. If this is not the case, we simply replace $\mathbf{x}_t$ and $\mathbf{u}_t$ with $\mathbf{x}_t - \hat{\mathbf{x}}_t$ and $\mathbf{u}_t - \hat{\mathbf{u}}_t$ in the above equations. The subscripts in the equation denote differentiation. For example, $\ell_{\mathbf{xu},\mathbf{xu}t}$ is the second derivative of $\ell_t(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$ with respect to $(\mathbf{x}_t, \mathbf{u}_t)^{\mathrm{T}}$, $\ell_{\mathbf{xu}t}$ is the first derivative, and so forth. The second derivative of the dynamics, $f_{\mathbf{xu},\mathbf{xu}t}$, is a tensor, and $V_{\mathbf{x}t+1} \cdot f_{\mathbf{xu},\mathbf{xu}t}$ denotes the tensor contraction $\sum_i [V_{\mathbf{x}t+1}]_i \frac{\partial [f_t]_i}{\partial (\mathbf{x}_t, \mathbf{u}_t)^{\mathrm{T}}}$. Equivalently, we can express the first and second derivatives of the Q-function as

$$\begin{aligned}
Q_{\mathbf{x}t} &= \ell_{\mathbf{x}t} + f_{\mathbf{x}t}^{\mathrm{T}} V_{\mathbf{x}t+1} \\
Q_{\mathbf{u}t} &= \ell_{\mathbf{u}t} + f_{\mathbf{u}t}^{\mathrm{T}} V_{\mathbf{x}t+1} \\
Q_{\mathbf{x},\mathbf{x}t} &= \ell_{\mathbf{x},\mathbf{x}t} + f_{\mathbf{x}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{x}t} + V_{\mathbf{x}t} \cdot f_{\mathbf{x},\mathbf{x}t} \\
Q_{\mathbf{u},\mathbf{u}t} &= \ell_{\mathbf{u},\mathbf{u}t} + f_{\mathbf{u}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{u}t} + V_{\mathbf{x}t} \cdot f_{\mathbf{u},\mathbf{u}t} \\
Q_{\mathbf{u},\mathbf{x}t} &= \ell_{\mathbf{u},\mathbf{x}t} + f_{\mathbf{u}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{x}t} + V_{\mathbf{x}t} \cdot f_{\mathbf{u},\mathbf{x}t}.
\end{aligned}$$

Since the value function is the minimum of the Q-function with respect to $\mathbf{u}_t$, we can solve for the optimal action as

$$\mathbf{u}_t = \arg\min_{\mathbf{u}_t} Q_t(\mathbf{x}_t, \mathbf{u}_t) = -Q_{\mathbf{u},\mathbf{u}t}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{u},\mathbf{x}}\mathbf{x}_t) = \mathbf{K}_t\mathbf{x}_t + \mathbf{k}, \tag{3.2}$$

where $\mathbf{K}_t$ is referred to as a feedback matrix. Substituting this equation for $\mathbf{u}_t$ into Equation 3.1 yields the value function

$$\begin{aligned} V_t(\mathbf{x}_t, \mathbf{u}_t) \approx\ & Q_t(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) - \frac{1}{2}Q_{\mathbf{u}t}^{\mathrm{T}}Q_{\mathbf{u},\mathbf{u}t}^{-1}Q_{\mathbf{u}t} + \\ & \frac{1}{2}\mathbf{x}_t^{\mathrm{T}}[Q_{\mathbf{x},\mathbf{x}t} - Q_{\mathbf{x},\mathbf{u}t}Q_{\mathbf{u},\mathbf{u}t}^{-1}Q_{\mathbf{u},\mathbf{x}t}]\mathbf{x}_t + \mathbf{x}_t^{\mathrm{T}}[Q_{\mathbf{x}t} - Q_{\mathbf{x},\mathbf{u}t}Q_{\mathbf{u},\mathbf{u}t}^{-1}Q_{\mathbf{u}t}], \end{aligned}$$

with the corresponding value function gradients and Hessians given by

$$\begin{aligned} V_{\mathbf{x}t} &= Q_{\mathbf{x}t} - Q_{\mathbf{x},\mathbf{u}t}Q_{\mathbf{u},\mathbf{u}t}^{-1}Q_{\mathbf{u}t} \\ V_{\mathbf{x},\mathbf{x}t} &= Q_{\mathbf{x},\mathbf{x}t} - Q_{\mathbf{x},\mathbf{u}t}Q_{\mathbf{u},\mathbf{u}t}^{-1}Q_{\mathbf{u},\mathbf{x}t}. \end{aligned}$$

The resulting recursion allows us to compute the value function and feedback terms $\mathbf{K}_t$ and $\mathbf{k}_t$ by proceeding backwards, starting with the last time step T where the Q-function is simply the cost. The result of this backward dynamic programming pass is a linear feedback policy $\mathbf{u}_t = \mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t$ at every time step. By following this policy, we can obtain a new nominal trajectory with lower cost. Repeating the second order expansion and dynamic programming procedure around this new trajectory will further improve the trajectory cost, eventually leading to a local optimum.

Before applying DDP to realistic problems, a few additional changes must be made. First, the Taylor expansion of the dynamics and cost only gives us an accurate representation of the system in the local neighborhood of the nominal trajectory. If the feedback policy $\mathbf{u}_t = \mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t$ produces a trajectory that deviates drastically from the previous one, the cost may not in fact decrease. Therefore, a line search must be used. Following Tassa et al. (2012), we can perform a linesearch by using the policy $\mathbf{u}_t = \mathbf{K}_t\mathbf{x}_t + \alpha\mathbf{k}_t$ and varying α until the new trajectory achieves a sufficient improvement over the old one. The second detail is that the matrix $Q_{\mathbf{u},\mathbf{u}t}$ may not be positive definite, in which case the minimization

in Equation 3.2 is unbounded. The solution proposed by Tassa et al. (2012), which is also employed in the methods presented in this thesis, is to regularize the value function by adding a factor μ to its diagonal, resulting in a regularization of the form $\mu f_{\mathbf{u}t}^{\mathrm{T}} f_{\mathbf{u}t}$ on $Q_{\mathbf{u},\mathbf{u}t}$. Intuitively, regularizing the value function keeps the new trajectory closer to the old one, limiting the step length to mitigate the uncertainty induced by a non-positive-definite Hessian. Details about the schedule for increasing μ and varying α can be found in previous work (Tassa et al., 2012).

The third and final detail relates to the form of the dynamics. While the classic DDP algorithm employs a second order expansion of the dynamics, the second derivative tensor $f_{\mathbf{xu},\mathbf{xu}t}$ is in general very expensive to compute. Since we usually work with a black box simulator of the system dynamics, the dynamics must be differentiated with finite differences. Computing second derivatives in this way scales quadratically with the task dimensionality. For even moderately sized problems ($20 - 50$ dimensions), it is advantageous to simply ignore the second order terms $f_{\mathbf{xu},\mathbf{xu}t}$, insteading using only a linearization of the dynamics. The resulting method is known as iterative LQG (Li and Todorov, 2004). Although it no longer has the appealing second order convergence properties of DDP, in practice many more DDP iterations can be taken with first order dynamics in the same amount of time as would be required for a single second order DDP iteration. This often results in faster convergence, as well as simpler dynamic programming equations, since the Q-function now becomes

$$
\begin{aligned}
Q_{\mathbf{x}t} &= \ell_{\mathbf{x}t} + f_{\mathbf{x}t}^{\mathrm{T}} V_{\mathbf{x}t+1} \\
Q_{\mathbf{u}t} &= \ell_{\mathbf{u}t} + f_{\mathbf{u}t}^{\mathrm{T}} V_{\mathbf{x}t+1} \\
Q_{\mathbf{x},\mathbf{x}t} &= \ell_{\mathbf{x},\mathbf{x}t} + f_{\mathbf{x}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{x}t} \\
Q_{\mathbf{u},\mathbf{u}t} &= \ell_{\mathbf{u},\mathbf{u}t} + f_{\mathbf{u}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{u}t} \\
Q_{\mathbf{u},\mathbf{x}t} &= \ell_{\mathbf{u},\mathbf{x}t} + f_{\mathbf{u}t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{x}t}.
\end{aligned}
$$

DDP can be used to find a locally optimal trajectory. However, in the algorithms presented in this paper, it will be necessary to consider probability distributions over trajectories. In learning cost functions, this distribution over trajectories will provide us with a generative model of expert behavior. In then learning the actual motor skill, trajectory

distributions will provide us with a variety of different trajectories on which the controller may be trained, insteading of restricting us to just the single solution from DDP. In the next section, I introduce the concept of soft optimality or stochastic optimal control, which can be used to reason about distributions over trajectories.

3.3 Stochastic Optimal Control

While standard optimal control techniques like DDP can be used to produce a single locally optimal trajectory, inverse optimal control and policy search typically benefit from access to a number of trajectories that illustrate many possible executions of the task, typically in the form of a probability distribution over trajectories.

When performing probabilistic inverse optimal control, we require a probabilistic model of the expert's behavior. This model corresponds to some form of optimal control, but because human experts are unlikely to be perfectly optimal, standard trajectory optimization makes for a poor model, as it does not explain minor suboptimalities. A trajectory distribution, on the other hand, could contain a range of near-optimal executions, assigning lower probability to trajectories with higher cost.

Similarly, when optimizing a policy, it is difficult to learn an effective policy from a small number of trajectories, since the policy must act intelligently even when it deviates from the training states. As discussed in the previous chapter, most RL methods sample from the trajectory distribution of the current policy to address this issue. If we instead wish to make use of trajectory optimization for policy learning, we must similarly be able to generate a distribution over trajectories from which the policy can learn not just a single trajectory, but a collection of feedback rules that stabilize a good execution of the task.

To optimize a trajectory distribution $q(\tau)$, we must first formulate a suitable objective. The classic optimality principle suggests that the objective should be the expectation of the cost under $q(\tau)$, given by $E_q[\sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)] = E_q[\ell(\tau)]$. However optimizing such an objective under deterministic dynamics would inevitably produce a degenerate distribution containing the single optimal trajectory. Under non-deterministic dynamics, the distribution would be wider, but the action distribution at each state would still be degenerate.

In practice, we will often want trajectory distributions that are as wide as possible, while

only including trajectories with low cost. A better objective in this case can be obtained by augmenting the classical objective with an entropy term, to get

$$q(\tau) = \arg\min_q E_q[\ell(\tau)] - \mathcal{H}(q), \tag{3.3}$$

where $\mathcal{H}(q)$ is the differential entropy of $q(\tau)$, given by

$$\mathcal{H}(q) = -\int q(\tau) \log q(\tau) d\tau.$$

This type of control problem has been termed soft optimality or maximum entropy control (Ziebart, 2010), stochastic optimal control (Kappen et al., 2012), and linearly-solvable control (Todorov, 2006a), and is known to correspond closely to inference in probabilistic graphical models.

The objective in Equation 3.3 has a few interesting properties. First, we can rewrite the objective as the KL-divergence between two probability distributions: the trajectory distribution $q(\tau)$ and an exponential cost distribution $\rho(\tau) \propto \exp(-\ell(\tau))$,[2] since

$$E_q[\ell(\tau)] - \mathcal{H}(q) = -E_q[\log \rho(\tau)] - \mathcal{H}(q) = D_{\mathrm{KL}}(q(\tau) \| \rho(\tau)) + \text{const}, \tag{3.4}$$

where the constant is the partition function of $\rho(\tau)$. This means that when no constraints are placed on the form of $q(\tau)$, the optimal choice of $q(\tau)$ is $\rho(\tau) \propto \exp(-\ell(\tau))$. When $q(\tau)$ is constrained to be in some distribution class (such as the Gaussian case discussed in the following section), the optimal choice of $q(\tau)$ is the I-projection of $\rho(\tau)$ onto that distribution class. This is exactly the distribution from that class that minimizes the KL-divergence in Equation 3.4.

The Boltzmann-type distribution $\rho(\tau)$ also corresponds the posterior marginal in a particular graphical model, shown in Figure 3.1. This model is a dynamic Bayesian network that relates the states to the actions by means of an action prior $p(\mathbf{u}_t|\mathbf{x}_t)$, relates

[2]The notation $\rho(\tau) \propto \exp(-\ell(\tau))$ is somewhat incomplete, since $\rho(\tau)$ must be a distribution over dynamically consistent trajectories, rather than all state-action sequences. Formally, the distribution is given by $\rho(\tau) \propto p(\mathbf{x}_1) \prod_{t=1}^{T} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) \exp(-\ell(\mathbf{x}_t, \mathbf{u}_t))$. However, if we assume that a trajectory is parameterized only by its actions, while the states are simply a consequence of the (uncontrolled) dynamics, the notation $\rho(\tau) \propto \exp(-\ell(\tau))$ can also be used.

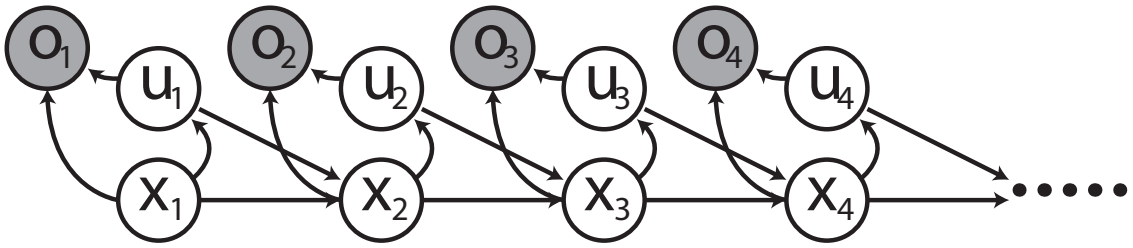


Figure 3.1: Graphical model with auxiliary optimality variables $\mathcal{O}_t$. The factors in the Bayesian network are the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, the action prior $p(\mathbf{u}_t|\mathbf{x}_t)$, and the binary optimality variables $P(\mathcal{O}_t = 1|\mathbf{x}_t, \mathbf{u}_t) = \exp(-\ell(\mathbf{x}_t, \mathbf{u}_t))$.

the next state to the preceding state and action via the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, and introduces the cost by means of binary optimality variables $\mathcal{O}_t$, where $P(\mathcal{O}_t = 1|\mathbf{x}_t, \mathbf{u}_t) = \exp(-\ell(\mathbf{x}_t, \mathbf{u}_t))$. If we condition on all optimality variables being set to 1, we obtain the trajectory distribution $P(\mathcal{O}_{1,\ldots,T} = 1|\tau) = \exp(-\ell(\tau))$. This means that $p(\tau|\mathcal{O}_{1,\ldots,T} = 1)$ is identical to $\rho(\tau)$, making $\rho(\tau)$ is the posterior marginal over trajectories in this graphical model conditioned on all of the optimality variables being 1.

This model can be seen as analogous to a hidden Markov model or Kalman filter, where the "observations" are the optimality variables. In fact, inference in this model, which corresponds to solving the stochastic optimal control problem, can proceed analogously to inference in a continuous state hidden Markov model. We can perform inference by means of a dynamic programming backward pass, where the backward probabilities are given by

$$\beta_t(\mathbf{x}_t, \mathbf{u}_t) = P(\mathcal{O}_{t,\ldots,T}|\mathbf{x}_t, \mathbf{u}_t) = \int \beta_{t+1}(\mathbf{x}_{t+1}) p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) P(\mathcal{O}_t|\mathbf{x}_t, \mathbf{u}_t) d\mathbf{x}_{t+1},$$

where $\beta_{t+1}(\mathbf{x}_{t+1})$ marginalizes out the action from $\beta_t(\mathbf{x}_t, \mathbf{u}_t)$:

$$\beta_{t+1}(\mathbf{x}_{t+1}) = \int \beta_{t+1}(\mathbf{x}_{t+1}, \mathbf{u}_{t+1}) p(\mathbf{u}_{t+1}|\mathbf{x}_{t+1}) d\mathbf{u}_{t+1}.$$

In general, the action prior corresponds to the notion of *passive dynamics* in linearly solvable control (Todorov, 2006a). If we are using maximum entropy, as in Equation 3.4, the corresponding action prior is a uniform distribution, which means that $\beta_{t+1}(\mathbf{x}_{t+1})$ can be obtained simply by integrating out the action from $\beta_{t+1}(\mathbf{x}_{t+1}, \mathbf{u}_{t+1})$.

From these equations, we can obtain the posterior distribution over actions conditioned on states, which corresponds to the optimal policy:

$$p(\mathbf{u}_t|\mathbf{x}_t, \mathcal{O}_{1,\dots,T}) = p(\mathbf{u}_t|\mathbf{x}_t, \mathcal{O}_{t,\dots,T}) = \frac{\beta_t(\mathbf{x}_t, \mathbf{u}_t)}{\beta_t(\mathbf{x}_t)},$$

where the first equality holds because of the conditional independencies in the Bayesian network. We can also define the Q-function and value function for the stochastic optimal control problem as $Q_t(\mathbf{x}_t, \mathbf{u}_t) = -\log \beta_t(\mathbf{x}_t, \mathbf{u}_t)$ and $V_t(\mathbf{x}_t) = -\log \beta_t(\mathbf{x}_t)$, yielding the log-space recurrence

$$\begin{aligned} Q_t(\mathbf{x}_t, \mathbf{u}_t) &= \ell(\mathbf{x}_t, \mathbf{u}_t) + V_t(f_t(\mathbf{x}_t, \mathbf{u}_t)) \\ V_t(\mathbf{x}_t) &= -\log \int \exp(-Q_t(\mathbf{x}_t, \mathbf{u}_t)) d\mathbf{u}_t \\ p(\mathbf{u}_t|\mathbf{x}_t, \mathcal{O}_{1,\dots,T}) &= \exp(-Q_t(\mathbf{x}_t, \mathbf{u}_t) + V_t(\mathbf{x}_t)) \end{aligned} \tag{3.5}$$

in the case of deterministic dynamics. In the case of stochastic dynamics, we can replace $V_t(f_t(\mathbf{x}_t, \mathbf{u}_t))$ with $E[V(\mathbf{x}_{t+1})]$ as before, but this does not correspond exactly to the posterior in the graphical model in Figure 3.1.[3] Instead, this produces the solution predicted under maximum *causal* entropy, as discussed by Ziebart (2010). In many cases, this solution is actually better suited for control problems, as it corresponds to the expected outcome under stochastic dynamics, rather than the best-case outcome. A full discussion of maximum causal entropy is outside the scope of this thesis, and we refer the reader to Ziebart (2010) for details.

Note that the equation for the value function can be interpreted as a softened minimum function, providing an intuitive link to the classical control setting, where the value function is a hard minimum over the Q-function with respect to the actions.

This derivation can now be used to compute the optimal trajectory distribution under the soft optimality objective. After computing the posteriors $p(\mathbf{u}_t|\mathbf{x}_t, \mathcal{O}_{1,\dots,T})$ at each time step, the trajectory distribution is recovered as a product of these posteriors, multiplied by

[3]To obtain the true posterior, we would instead need to take the expectation outside of the exponential. This corresponds to considering the next state as another entry in the action vector, with a prior that corresponds to the dynamics. The issue with this interpretation is that the agent does not get to "choose" the next state, only the action. For low-variance dynamics, this approximation is reasonable but somewhat optimistic.

the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ and initial state distribution $p(\mathbf{x}_1)$. This distribution minimizes the objective in Equation 3.4, capturing a wide variety of low-cost trajectories. However, as in the classical control setting, representing the value function exactly in large continuous spaces is impossible. In the following section, I derive a DDP-like algorithm for computing Gaussian trajectory distributions $q(\tau)$ that approximately minimize the objective in Equation 3.4 by using the same kind of linear-quadratic expansion as in the previous section.

3.4 Optimizing Trajectory Distributions

We can optimize a Gaussian trajectory distribution $q(\tau)$ with respect to the soft optimality objective in Equation 3.4 by using the same Taylor expansion we employed in Section 3.2. In the case of fitting trajectory distributions, these assumptions correspond to the Laplace approximation, which is used to fit a Gaussian to a complex distribution by using its Taylor expansion about some point (Tierney and Kadane, 1986). As in Section 3.2, we will use linearized dynamics, although second order dynamics may also be used in general. Starting from Equation 3.5, we have

$$
\begin{aligned}
&Q_t(\mathbf{x}_t, \mathbf{u}_t) = \ell(\mathbf{x}_t, \mathbf{u}_t) + V_t(f_t(\mathbf{x}_t, \mathbf{u}_t)) \\
&\approx \frac{1}{2}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^\mathrm{T} \ell_{\mathbf{xu},\mathbf{xu}t}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^\mathrm{T} \ell_{\mathbf{xu}t} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^\mathrm{T} f_{\mathbf{xu}t}^\mathrm{T} V_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{xu}t}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^\mathrm{T} f_{\mathbf{xu}t}^\mathrm{T} V_{\mathbf{x}t+1} \\
&= \frac{1}{2}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^\mathrm{T} Q_{\mathbf{xu},\mathbf{xu}t}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^\mathrm{T} Q_{\mathbf{xu}t},
\end{aligned}
$$

where we assume that the value function at step $t + 1$ is quadratic. As with standard DDP, we will show that the value function remains quadratic by induction. First, we rewrite the Q-function so that exponentiating it produces a function that is Gaussian in $\mathbf{u}_t$ up to a

normalization constant:

$$\begin{aligned}
Q_t(\mathbf{x}_t, \mathbf{u}_t) &= \frac{1}{2}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^{\mathrm{T}} Q_{\mathbf{xu},\mathbf{xu}t}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t \\ \mathbf{u}_t\end{bmatrix}^{\mathrm{T}} Q_{\mathbf{xu}t} \\
&= \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{x}t}\mathbf{x}_t + \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x}t} + \frac{1}{2}\mathbf{u}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}\mathbf{u}_t + \mathbf{u}_t^{\mathrm{T}} Q_{\mathbf{u}t} + \mathbf{u}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{x}t}\mathbf{x}_t \\
&= \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{x}t}\mathbf{x}_t + \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x}t} - \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}\mathbf{x}_t - \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} + \\
&\quad \frac{1}{2}(\mathbf{u}_t + Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}\mathbf{x}_t + Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t})^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}(\mathbf{u}_t + Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}\mathbf{x}_t + Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t}) \\
&= \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{x}t}\mathbf{x}_t + \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x}t} - \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}\mathbf{x}_t - \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} + \\
&\quad \frac{1}{2}(\mathbf{u}_t - \mathbf{K}_t\mathbf{x}_t - \mathbf{k}_t)^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}(\mathbf{u}_t - \mathbf{K}_t\mathbf{x}_t - \mathbf{k}_t) \qquad (3.6)
\end{aligned}$$

We can now solve for the conditional distribution $q(\mathbf{u}_t|\mathbf{x}_t)$ simply by exponentiating and normalizing the Q-function (omitting all terms that do not depend on $\mathbf{u}_t$):

$$\begin{aligned}
q(\mathbf{u}_t|\mathbf{x}_t) \propto \exp(-Q_t(\mathbf{x}_t, \mathbf{u}_t)) &\propto \exp(-\frac{1}{2}(\mathbf{u}_t - \mathbf{K}_t\mathbf{x}_t - \mathbf{k}_t)^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}(\mathbf{u}_t - \mathbf{K}_t\mathbf{x}_t - \mathbf{k}_t)) \\
&\propto \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, Q_{\mathbf{u},\mathbf{u}t}^{-1}) \qquad (3.7)
\end{aligned}$$

Completing the inductive proof, we can evaluate the soft minimum analytically, since the Gaussian component integrates to a constant that is independent of $\mathbf{x}_t$, while the remaining terms do not depend on $\mathbf{u}_t$ and can be taken outside of the integral:

$$\begin{aligned}
V_t(\mathbf{x}_t) = &-\log \int \exp\left(-\frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{x}t}\mathbf{x}_t - \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x}t} + \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}\mathbf{x}_t + \right. \\
&\left. \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} - \frac{1}{2}(\mathbf{u}_t - \mathbf{K}_t\mathbf{x}_t - \mathbf{k}_t)^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}(\mathbf{u}_t - \mathbf{K}_t\mathbf{x}_t - \mathbf{k}_t)\right) d\mathbf{u}_t \\
= &\frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{x}t}\mathbf{x}_t + \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x}t} - \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}\mathbf{x}_t - \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} + \text{const}
\end{aligned}$$

Note that distribution $q(\mathbf{u}_t|\mathbf{x}_t)$ is a conditional linear Gaussian, with a mean that is precisely identical to the DDP linear feedback policy, and a covariance that is equal to the inverse of the action-dependent quadratic component of the Q-function. This fact has a simple intuitive interpretation: when the quadratic component of the Q-function is highly peaked, only

a few actions are good, and the policy $q(\mathbf{u}_t|\mathbf{x}_t)$ must be more deterministic. When the Q-function is very broad, many actions are about equally good, and we can afford to act more randomly. Finally, note that the value function is identical to the value function in standard DDP. This is not generally the case when employing the soft optimality objective, but is known to hold in the linear-quadratic case. This intriguing relationship between inference and control is known as the Kalman duality, and was noted as far back as Kalman's original work on the Kalman filter, which has the same form as the graphical model in Figure 3.1, but with the optimality variables $\mathcal{O}_t$ replaced by observations (Kalman, 1960).

The implication of the above derivation is that an approximate Gaussian I-projection of $\rho(\tau)$ minimizing the soft optimality objective in Equation 3.4 can be optimizing by running the DDP algorithm described in Section 3.2, but replacing the deterministic linear feedback policy with the stochastic linear Gaussian policy in Equation 3.7. In the following two chapters, we will see how this procedure can be used to solve two challenging problems: the inverse optimal control problem, which addresses the estimation of a cost or reward function from example demonstrations, and the motor skill learning problem, which deals with optimizing policy parameters to minimize a known cost. In the case of inverse optimal control, the probabilistic model in Figure 3.1 is used as a generative model of the expert's behavior, explaining suboptimal demonstrations and making the method robust to noise. In the case of policy learning, trajectory distributions are used to explore the space of trajectories that have low cost and agree with the current policy, without committing to a single optimal trajectory or requiring the policy to replicate that trajectory exactly.

Chapter 4

Inverse Optimal Control

Inverse optimal control (IOC), also known as inverse reinforcement learning (IRL), refers to the task of recovering an unknown reward or cost function in a goal-driven task from expert demonstrations of the behavior. This cost function can then be used to perform apprenticeship learning, generalize the expert's behavior to new situations, or infer the expert's goals (Ng and Russell, 2000). While cost functions for some motor skills are simple and easy to design, other behaviors may require complex functions that trade off competing goals in non-intuitive ways. For example, the previously mentioned coffee cup manipulation cost requires trading off success at grasphing the cup, prevention of spilling its contents, avoidance of obstacles, and the minimization of effort. The relative weighting of each term in the scalar cost function can have large effects on the resulting behavior, and choosing appropriate weights is not always intuitive. Other motor skills, such as complex acrobatics, may not even present obvious criteria to add to the cost without significant study and engineering effort. Inverse optimal control can allow us to avoid the challenges of designing such cost functions by hand when example demonstrations are available.

Performing IOC in continuous, high-dimensional domains is challenging, because IOC algorithms are usually much more computationally demanding than the corresponding "forward" control methods that solve for an optimal policy or trajectory given a known cost. Many standard IOC algorithms require solving for the optimal policy under the current cost in the inner loop of a cost optimization procedure, which becomes intractable when solving the forward problem even once is exceptionally difficult. In this chapter, I present

a probabilistic IOC algorithm that efficiently handles control problems with large, continuous state and action spaces by considering only the shape of the learned cost function in the neighborhood of the expert's demonstrations.

Since this method only considers the shape of the cost around the expert's examples, it does not integrate global information about the cost along all alternative paths. This is analogous to the trajectory optimization methods described in the preceding section, which solve the forward control problem by finding a local optimum. However, while a local model is a disadvantage for solving the forward problem, it can actually be advantageous in IOC. In the inverse setting, a local model corresponds to the assumption that the example demonstration is the result of a local, rather than global, optimization. This removes the requirement for the expert demonstrations to be globally optimal, allowing the algorithm to use examples that only exhibit local optimality.[1] For complex tasks, human experts might find it easier to provide such locally optimal examples. For instance, a skilled driver might execute every turn perfectly, but still take a globally suboptimal route to the destination.

The algorithm in this chapter optimizes the approximate likelihood of the expert trajectories under a parameterized cost. The approximation assumes that the expert's trajectory lies near a peak of this likelihood, and the resulting optimization finds a cost function under which this peak is most prominent. Since this approach only considers the shape of the cost around the examples, it does not require the examples to be globally optimal, and remains efficient even in high dimensions. I present two variants of the algorithm that learn the cost either as a linear combination of the provided features, as is common in prior work, or as a nonlinear function of the features, as in a number of recent methods (Ratliff et al., 2009; Levine et al., 2010b, 2011b).

4.1 Probabilistic Inverse Optimal Control

I will consider deterministic, fixed-horizon control tasks with continuous states $\mathbf{x}_t$, continuous actions $\mathbf{u}_t$, and discrete-time dynamics $f_t(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_{t+1}$, where the cost function

[1] Note that "global optimality" in this case refers only to the optimality of the trajectories with respect to the (learned and unknown) cost function, not the local or global optimality of the learning algorithm itself. In the inverse setting, the trajectories are provided as data, and are not themselves optimized.

$\ell(\mathbf{x}_t, \mathbf{u}_t)$ is unknown. IOC aims to find a cost function ℓ under which the optimal actions resemble the expert's demonstrated trajectories $\mathcal{D} = \{\tau_1, \ldots, \tau_n\}$. To represent the cost function, the algorithm is presented with cost features $\mathbf{f}$, where each feature is a function $\mathbf{f}_i : (\mathbf{x}_t, \mathbf{u}_t) \rightarrow \mathbb{R}$. Real demonstrations are rarely perfectly optimal, so we require a model for the expert's behavior that can explain suboptimality or "noise." We employ the maximum entropy IRL (MaxEnt) model (Ziebart, 2010), which corresponds exactly to the soft optimality model described in the previous chapter. Under this model, the probability of a given example trajectory τ is proportional to the exponential of its total cost:[2]

$$p(\tau|\ell) = \frac{1}{Z} \exp(-\ell(\tau)), \tag{4.1}$$

where Z is the partition function. Under this model, the expert follows a stochastic policy that becomes more deterministic when the stakes are high, and more random when all choices have similar value. In prior work, the log likelihood derived from Equation 4.1 was maximized directly. However, computing the partition function Z requires finding the complete policy under the current reward, using a variant of value iteration (Ziebart, 2010). In high dimensional spaces, this becomes intractable, since this computation scales exponentially with the dimensionality of the state space.

We can avoid computing the partition function exactly by using the trajectory optimization procedure described in the preceding chapter in Section 3.4. This corresponds to using the Laplace approximation to approximate Equation 4.1, and can be done efficiently even in high dimensional, continuous domains. In addition to breaking the exponential dependence on dimensionality, this approximation removes the requirement that the example trajectories be globally near-optimal, and only requires approximate local optimality. An example of a locally optimal but globally suboptimal trajectory is shown in Figure 4.1: although

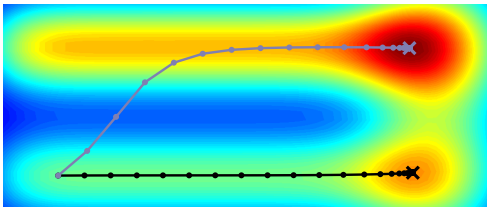


Figure 4.1: A trajectory that is locally optimal but globally suboptimal (black) and a globally optimal trajectory (grey). Warmer colors indicate lower cost.

[2] In the more common case of multiple example trajectories, their probabilities are simply multiplied together to obtain $p(\mathcal{D})$.

another path has a higher total reward, any local perturbation of the trajectory increases the total cost.

4.2 Local Inverse Optimal Control

To evaluate Equation 4.1 without computing the partition function Z, we apply the Laplace approximation, which locally models the distribution as a Gaussian (Tierney and Kadane, 1986). Note that this is not equivalent to modeling the cost function itself as a Gaussian, since Equation 4.1 uses the sum of the cost along a path. In the context of IOC, this corresponds to assuming that the expert performs a *local* optimization when choosing the actions $\mathbf{u}_t$, rather than global planning. This assumption is strictly less restrictive than the assumption of global optimality. As we will see, this results in an objective that matches the soft optimality objective in Section 3.4, which in this section will be derived directly from the Laplace approximation.

We can rewrite Equation 4.1 as

$$p(\tau|\ell) = e^{-\ell(\tau)} \left[\int e^{-\ell(\tilde{\tau})} d\tilde{\tau} \right]^{-1}.$$

We approximate this probability using a second order Taylor expansion of $\ell(\tau)$ around τ:

$$\ell(\tilde{\tau}) \approx \ell(\tau) + (\tilde{\tau} - \tau)^{\mathrm{T}} \frac{\partial \ell}{\partial \tau} + \frac{1}{2} (\tilde{\tau} - \tau)^{\mathrm{T}} \frac{\partial^2 \ell}{\partial \tau^2} (\tilde{\tau} - \tau).$$

Denoting the gradient $\frac{\partial \ell}{\partial \tau}$ as $\mathbf{g}$ and the Hessian $\frac{\partial^2 \ell}{\partial \tau^2}$ as $\mathbf{H}$, the approximation to Equation 4.1 is given by

$$\begin{aligned} p(\tau|\ell) &\approx e^{-\ell(-\tau)} \left[\int e^{-\ell(\tau) - (\tilde{\tau}-\tau)^{\mathrm{T}}\mathbf{g} - \frac{1}{2}(\tilde{\tau}-\tau)^{\mathrm{T}}\mathbf{H}(\tilde{\tau}-\tau)} d\tilde{\tau} \right]^{-1} \\ &= \left[\int e^{\frac{1}{2}\mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{g} - \frac{1}{2}(\mathbf{H}(\tilde{\tau}-\tau)+\mathbf{g})^{\mathrm{T}}\mathbf{H}^{-1}(\mathbf{H}(\tilde{\tau}-\tau)+\mathbf{g})} d\tilde{\tau} \right]^{-1} \\ &= e^{-\frac{1}{2}\mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{g}} |\mathbf{H}|^{\frac{1}{2}} (2\pi)^{-\frac{d_\tau}{2}}, \end{aligned}$$

from which we obtain the approximate log likelihood

$$\mathcal{L}(\ell) = -\frac{1}{2}\mathbf{g}^{\mathrm{T}}\mathbf{H}^{-1}\mathbf{g} + \frac{1}{2}\log|\mathbf{H}| - \frac{d_\tau}{2}\log 2\pi. \tag{4.2}$$

Intuitively, this likelihood indicates that cost functions under which the example paths have small gradients and large Hessians are more likely. The magnitude of the gradient corresponds to how close the example is to a local minimum in the (total) cost landscape, while the Hessian describes how deep that minimum is. For a given parameterization of the cost, we can learn the most likely parameters by maximizing Equation 4.2. In the next section, we discuss how this objective and its gradients can be computed efficiently by using a variant of the trajectory optimization algorithm from the previous chapter.

4.3 Efficient Optimization with Dynamic Programming

We can optimize Equation 4.2 directly with any optimization method, but naïvely evaluating the objective requires solving the linear system $\mathbf{H}^{-1}\mathbf{g}$, which has a cost that is cubic in the path length T. To derive an efficient approximation, we can parameterize the trajectory τ by the sequence of actions $\mathbf{u}_1, \ldots, \mathbf{u}_T$ and decompose the gradient and Hessian in terms of the derivatives of $\ell(\tau)$ with respect to $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ and $\mathbf{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_T\}$ individually:

$$\begin{aligned}
\mathbf{g} &= \underbrace{\frac{\partial \ell}{\partial \mathbf{U}}}_{\tilde{\mathbf{g}}} + \underbrace{\frac{\partial \mathbf{X}}{\partial \mathbf{U}}^{\mathrm{T}}}_{\mathbf{J}} \underbrace{\frac{\partial \ell}{\partial \mathbf{X}}}_{\hat{\mathbf{g}}} \\
\mathbf{H} &= \underbrace{\frac{\partial^2 \ell}{\partial \mathbf{U}^2}}_{\tilde{\mathbf{H}}} + \underbrace{\frac{\partial \mathbf{X}}{\partial \mathbf{U}}^{\mathrm{T}}}_{\mathbf{J}} \underbrace{\frac{\partial^2 \ell}{\partial \mathbf{X}^2}}_{\hat{\mathbf{H}}} \underbrace{\frac{\partial \mathbf{X}}{\partial \mathbf{U}}}_{\mathbf{J}^{\mathrm{T}}} + \underbrace{\frac{\partial^2 \mathbf{x}}{\partial \mathbf{U}^2}}_{\breve{\mathbf{H}}} \underbrace{\frac{\partial \ell}{\partial \mathbf{X}}}_{\hat{\mathbf{g}}}.
\end{aligned}$$

Since $\ell(\mathbf{x}_t, \mathbf{u}_t)$ depends only on the state and action at time t, $\hat{\mathbf{H}}$ and $\tilde{\mathbf{H}}$ are block diagonal, with T blocks. To build the Jacobian $\mathbf{J}$, we employ the dynamics derivatives $f_{\mathbf{x}t}$ and $f_{\mathbf{x}\mathbf{u}t}$, and note that, since future actions do not influence past states, $\mathbf{J}$ is block upper triangular,

with nonzero blocks given recursively as

$$\mathbf{J}_{t_1,t_2} = \frac{\partial \mathbf{x}_{t_2}}{\partial \mathbf{u}_{t_1}}^{\mathrm{T}} = \begin{cases} f_{\mathbf{u}t_1}^{\mathrm{T}}, & t_2 = t_1 + 1 \\ \mathbf{J}_{t_1,t_2-1} f_{\mathbf{x}t_2-1}^{\mathrm{T}}, & t_2 > t_1 + 1 \end{cases}$$

We can now write $\mathbf{g}$ and $\mathbf{H}$ almost entirely in terms of matrices that are block diagonal or block triangular. Unfortunately, the final second order term $\breve{\mathbf{H}}$ does not exhibit such convenient structure. In particular, the Hessian of the last state $\mathbf{x}_T$ with respect to the actions $\mathbf{U}$ can be arbitrarily dense. We will therefore disregard this term. Since $\breve{\mathbf{H}}$ is zero only when the dynamics are linear, this corresponds to linearizing the dynamics. Since we now have linear dynamics and quadratic rewards, we can use the simple dynamic programming algorithm in the preceding chapter to evaluate the log-likelihood in Equation 4.2. First, note that

$$\log p(\tau|\ell) = \sum_{t=1}^{T} \log p(\hat{\mathbf{u}}_t|\hat{\mathbf{x}}_t, \ell) + \text{const} = \sum_{t=1}^{T} Q_t(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) - V_t(\hat{\mathbf{x}}_t) + \text{const}, \tag{4.3}$$

where $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ are the states and actions in τ, the second equality comes from Equation 3.5 in the preceding chapter, and the Q-function and value function are computed (in terms of soft optimality) with respect to the current cost ℓ. Since we have assumed a quadratic cost and linear dynamics, we can directly apply one backward pass of the DDP algorithm described in Section 3.4 to compute the Q-function and value function. However, a slight change to the derivation is necessary to properly handle the constants, since, although they may be independent of the state and action and therefore irrelevant for trajectory optimization, they may still depend on the cost, and must be considered if the cost is modified. Specifically, recall the dynamic programming step

$$\begin{aligned} V_t(\mathbf{x}_t) = & -\log \int \exp\left(-\frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{x}t}\mathbf{x}_t - \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x}t} + \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}\mathbf{x}_t + \right. \\ & \left. \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} - \frac{1}{2}(\mathbf{u}_t - \mathbf{K}_t\mathbf{x}_t - \mathbf{k}_t)^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}^{-1}(\mathbf{u}_t - \mathbf{K}_t\mathbf{x}_t - \mathbf{k}_t) \right) d\mathbf{u}_t \\ = & \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{x}t}\mathbf{x}_t + \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x}t} - \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}\mathbf{x}_t - \mathbf{x}_t^{\mathrm{T}} Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} + \text{const}. \end{aligned}$$

The constant const is a consquence of integrating the Gaussian term on the second line, and is equal to $\frac{1}{2}\log|Q_{\mathbf{u},\mathbf{u}t}| - \frac{1}{2}\log 2\pi$. Using Equation 3.6 for $Q_t(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$, we can evaluate the likelihood in Equation 4.3 as

$$\log p(\tau|\ell) = \sum_{t=1}^{T} -\frac{1}{2}Q_{\mathbf{u}t}^{\mathrm{T}}Q_{\mathbf{u},\mathbf{u}t}^{-1}Q_{\mathbf{u}t} + \frac{1}{2}\log|Q_{\mathbf{u},\mathbf{u}t}| - \frac{1}{2}\log 2\pi, \tag{4.4}$$

where we use the fact that $\mathbf{u}_t$ and $\mathbf{x}_t$ are zero to eliminate all terms that depend on $\mathbf{u}_t$ and $\mathbf{x}_t$. Recall from the previous chapter that this assumption can be made without loss of generalization, since $\mathbf{u}_t$ and $\mathbf{x}_t$ represents the perturbation from the nominal trajectory, which in this case is the example itself.

In order to learn a cost function that maximizes the approximate likelihood in equations 4.2 and 4.3, we must also evaluate its gradients with respect to the cost parameterization. This can be done as part of the dynamic programming backward pass by means of the chain rule, as described in Appendix A. An alternative dynamic programming approach is also described by Levine and Koltun (2012). The alternative approach is in general more complex to implement and somewhat less stable numerically, but tends to be more efficient when the number of cost function parameters is large, as in the case of the Gaussian process cost function described later. In the next two sections, I will discuss two options for parameterizing cost functions and some additional details of the maximum likelihood optimization for each parameterization.

4.4 Optimizing Linear Cost Functions

The first cost parameterization we consider is the linear parameterization, where the cost is simply a linear combination of the provided features, given by $\ell(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{w}^{\mathrm{T}}\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$, and the weights $\mathbf{w}$ are learned. In order to compute the approximate log-likelihood in Equation 4.2 using the dynamic programming algorithm, we require the gradients and Hessians of the cost with respect to the states and actions. For a linear parameterization, these are

simply linear combinations of the gradients and Hessians of the features:

$$\ell_{\mathbf{x}\mathbf{u}t} = \sum_{k=1}^{K} w_k \mathbf{f}_{\mathbf{x}\mathbf{u}tk} \qquad \ell_{\mathbf{x}\mathbf{u},\mathbf{x}\mathbf{u}t} = \sum_{k=1}^{K} w_k \mathbf{f}_{\mathbf{x}\mathbf{u},\mathbf{x}\mathbf{u}tk},$$

and the correspond derivatives with respect to each weight w_k, which are necessary to differentiate the log-likelihood, are simply $\mathbf{f}_{\mathbf{x}\mathbf{u}tk}$ and $\mathbf{f}_{\mathbf{x}\mathbf{u},\mathbf{x}\mathbf{u}tk}$.

When evaluating the log-likelihood by means of Equation 4.4, we may encounter non-positive-definite Q-function matrices, which can make the log-determinant term undefined and may cause the dynamic programming procedure to fail. This corresponds to the example path lying in a valley rather than on a peak of the log-probability landscape. A high-probability cost function will avoid such cases, but it is nontrivial to find an initial point for which the objective can be evaluated. We therefore add a dummy regularizer feature that ensures that the Q-function always has positive eigenvalues. This feature has a gradient that is zero at each time step, and a Hessian equal to the identity matrix.

The initial weight w_r on this feature must be set such that the matrices $Q_{\mathbf{u},\mathbf{u}t}$ of all example paths at all time steps are positive definite. We can find a suitable weight simply by doubling w_r until this requirement is met. During the optimization, we must drive w_r to zero in order to solve the original problem. In this way, w_r has the role of a relaxation, allowing the algorithm to explore the parameter space without requiring all $Q_{\mathbf{u},\mathbf{u}t}$ matrices to always be positive definite. Unfortunately, driving w_r to zero too quickly can create numerical instability, as the $Q_{\mathbf{u},\mathbf{u}t}$ matrices become ill-conditioned or singular. Rather than simply penalizing the regularizing weight, we found that we can maintain numerical stability and still obtain a solution with $w_r = 0$ by using the Augmented Lagrangian method (Birgin and Martínez, 2009). This method solves a sequence of maximization problems that are augmented by a penalty term of the form

$$\phi^{(j)}(w_r) = -\frac{1}{2}\mu^{(j)} w_r^2 + \lambda^{(j)} w_r,$$

where $\mu^{(j)}$ is a penalty weight, and $\lambda^{(j)}$ is an estimate of the Lagrange multiplier for the constraint $w_r = 0$. After each optimization, $\mu^{(j+1)}$ is increased by a factor of 10 if w_r has

not decreased, and $\lambda^{(j+1)}$ is set to

$$\lambda^{(j+1)} \leftarrow \lambda^{(j)} - \mu^{(j)} w_r.$$

This approach allows w_r to decrease gradually with each optimization without using large penalty terms.

4.5 Optimizing Nonlinear Cost Functions

In the nonlinear variant of this algorithm, the cost function is represented as a Gaussian process (GP) that maps from feature values to costs. The inputs of the Gaussian process are a set of inducing feature points $\mathbf{F} = [\mathbf{f}^1 \dots \mathbf{f}^n]^{\mathrm{T}}$, and the noiseless outputs $\mathbf{y}$ at these points are learned. The location of the inducing points can be chosen in a variety of ways. We simply choose the points that lie on the example paths, which concentrates the learning on the regions where the examples are most informative. Further discussion on the choice of inducing points may be found in previous work (Levine et al., 2011b). In addition to the outputs $\mathbf{y}$, we must also learn the hyperparameters λ and β that describe the GP kernel function, given by

$$k(\mathbf{f}^i, \mathbf{f}^j) = \beta \exp\left(-\frac{1}{2}\sum_k \lambda_k \left[(\mathbf{f}_k^i - \mathbf{f}_k^j)^2 + 1_{i \neq j}\mathbf{F}\right]\right).$$

This kernel is a variant of the radial basis function kernel, regularized by input noise $\mathbf{F}$. This input noise is necessary to avoid degeneracies that may occur when the outputs of a Gaussian process are optimized together with its hyperparameters. The GP covariance is then defined as $\mathbf{K}_{ij} = k(\mathbf{f}^i, \mathbf{f}^j)$, producing the following GP likelihood:

$$\log p(\mathbf{y}, \lambda, \beta | \mathbf{F}) = -\frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}| + \log p(\lambda, \beta | \mathbf{F}).$$

The last term in the likelihood is the prior on the hyperparameters. This prior encourages the feature weights λ to be sparse, and prevents degeneracies that occur as $\mathbf{y} \to 0$. The latter is accomplished with a prior that encodes the belief that no two inducing points are

deterministically dependent, as captured by their partial correlation:

$$\log p(\lambda, \beta|\mathbf{F}) = -\frac{1}{2}\text{tr}\left(\mathbf{K}^{-2}\right) - \sum_{k=1}^{K} \log\left(\lambda_k + 1\right)$$

This prior is discussed in more detail in previous work, along with the rationale behind including input noise within the kernel function (Levine et al., 2011b). The cost at a feature point $\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$ is given by the GP posterior mean, which we can additionally augment with a set of linear features $\mathbf{f}_l$ and corresponding weights $\mathbf{w}$:

$$\ell(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{k}_t\alpha + \mathbf{w}^{\mathrm{T}}\mathbf{f}_l(\mathbf{x}_t, \mathbf{u}_t),$$

where $\alpha = \mathbf{K}^{-1}\mathbf{y}$, and $\mathbf{k}_t$ is a row vector corresponding to the covariance between $\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$ and each inducing point $\mathbf{f}^i$, given by $\mathbf{k}_{ti} = k(\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t), \mathbf{f}^i)$. The linear features can be used to include a dummy regularizer feature, as described in the preceding section.

The exact log likelihood, before the proposed approximation, is obtained by using the GP likelihood as a prior on the IOC likelihood in Equation 4.1:

$$\log p(\tau, \mathbf{y}, \lambda, \beta, \mathbf{w}|\mathbf{F}) = -\sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t) - \log Z + \log p(\mathbf{y}, \lambda, \beta|\mathbf{F}).$$

Only the IOC likelihood is altered by the proposed approximation.

Using nonlinear cost representations allows us to learn more expressive costs in domains where a linear cost basis is not known, but with the usual bias and variance tradeoff that comes with increased model complexity. As shown in the evaluation in the following section, the linear method requires fewer examples when a linear basis is available, while the nonlinear variant can work with much less expressive features.

4.6 Experimental Evaluation

In this section, I will present experimental evaluations of the proposed IOC algorithm on simple planar robot arm tasks, planar navigation, simulated driving, and a complex humanoid locomotion task. In the robot arm task, the expert sets continuous torques on each

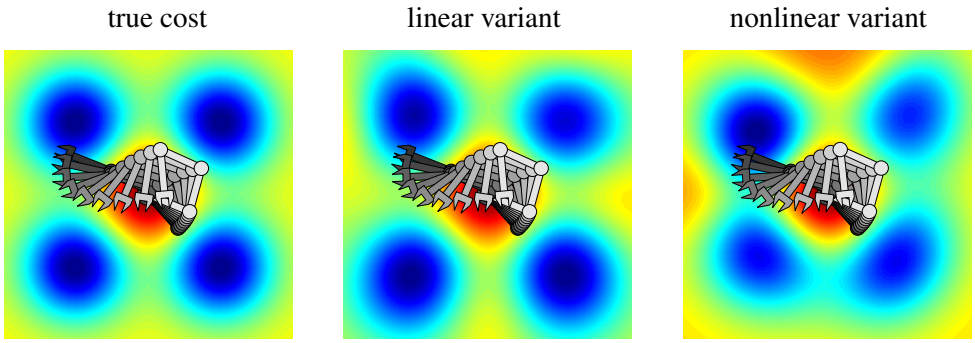


Figure 4.2: Robot arm costs learned by the proposed algorithms for a 4-link arm. One of eight examples is shown. Warner colors indicate low cost.

joint of an n-link planar robot arm. The cost function depends on the position of the end-effector. Each link has an angle and a velocity, producing a state space with $2n$ dimensions. By changing the number of links, we can vary the dimensionality of the task. An example of a 4-link arm is shown in Figure 4.2. Although this task is relatively simple, its high dimensionality makes comparisons to prior work impractical, so an even simpler planar navigation task is also included. In this task, the expert takes continuous steps on a plane, as shown in Figure 4.3. Two more realistic tasks are provided to exhibit the capacity of the algorithm to scale up to more practical problems: a simulated driving task, where example demonstrations are provided by human operators, and a humanoid locomotion task, where the example demonstration is motion capture of a real human run.

The cost function in the robot arm and navigation tasks has a Gaussian trough in the center, surrounded by four Gaussian peaks. The cost also penalizes each action with the square of its magnitude. In the comparisons to prior work, all IOC algorithms are provided with a grid of 25 evenly spaced Gaussian features and the squared action magnitude. In the nonlinear test in Section 4.6.2, the Cartesian coordinates of the arm end-effector are provided instead of the grid.

The linear and nonlinear variants of the proposed method are compared with the MaxEnt IRL and OptV algorithms (Ziebart, 2010; Dvijotham and Todorov, 2010). MaxEnt used a grid discretization for both states and actions, while OptV used discretized actions and adapted the value function features as described by Dvijotham and Todorov (2010). Since OptV cannot learn action-dependent costs, it was provided with the true weight for

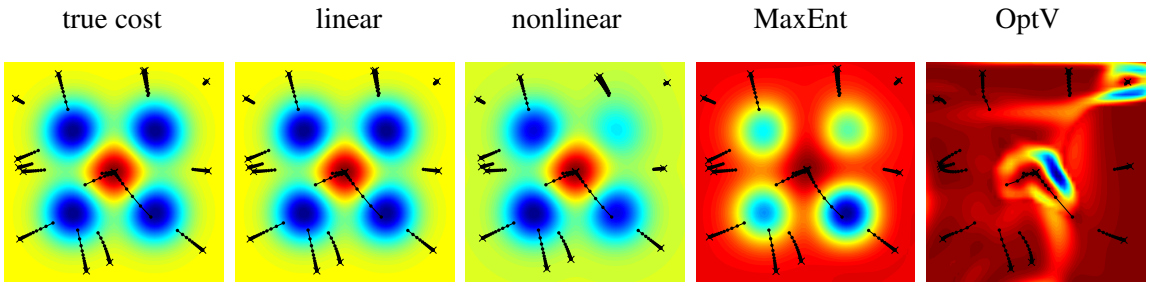


Figure 4.3: Planar navigation costs learned from 16 locally optimal examples. Black lines show optimal paths for each cost originating from example initial states. The costs learned by our algorithms better resemble the true cost than those learned by prior methods.

the action penalty term.

To evaluate a learned cost, the globally optimal paths with respect to this cost are computed from 32 random initial states that are not part of the training set. For the same initial states, an additional set of paths is computed that is optimal with respect to the true reward. In both cases, globally optimal paths are used, by first solving a discretization of the task with value iteration, and then finetuning the paths with continuous trajectory optimization. Once the evaluation paths are computed for both cost functions, we can obtain a cost loss by subtracting the true cost along the true optimal path from the true cost along the learned cost's path. This loss is low when the learned cost induces the same policy as the true one, and high when the learned cost causes the optimal path to visit regions where the true cost is high. Since the cost loss is measured entirely on globally optimal paths, it captures how well each algorithm learns the true, global cost, regardless of whether the examples are locally or globally optimal.

For the driving and humanoid locomotion tasks, where no true cost function is available, I present results based on qualitative features of the behavior and the visual appearance of the learned policies, with videos of the results presented in the linked supplementary materials.

4.6.1 Locally Optimal Examples

To test how well each method handles locally optimal examples, the navigation task was evaluated with increasing numbers of examples that were either globally or locally optimal. As discussed above, globally optimal examples were obtained with discretization,

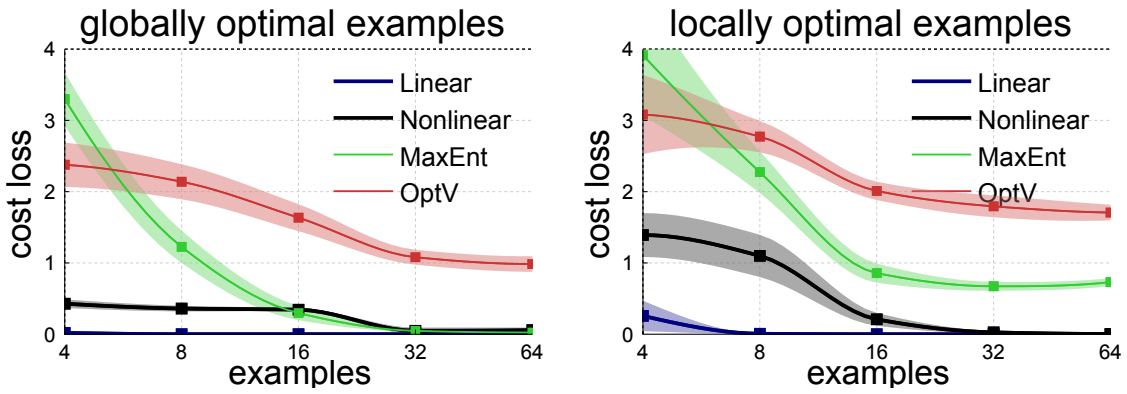


Figure 4.4: Cost loss for each algorithm with either globally or locally optimal planar navigation examples. Prior methods do not converge to the expert's policy when the examples are not globally optimal.

while locally optimal examples were computed by optimizing the trajectory from a random initialization. Each test was repeated eight times with random initial states for each example.

The results in Figure 4.4 show that both variants of the proposed algorithm converge to the correct policy. The linear variant requires fewer examples, since the features form a good linear basis for the true cost. MaxEnt assumes global optimality and does not converge to the correct policy when the examples are only locally optimal. It also suffers from discretization error. OptV has difficulty generalizing the cost to unseen parts of the state space, because the value function features do not impose meaningful structure on the reward.

4.6.2 Linear and Nonlinear Cost Functions

The robot arm task was used to evaluate each method with both the Gaussian grid features, and simple features that only provide the position of the end effector, and therefore do not form a linear basis for the true cost. The examples were globally optimal. The number of links was set to 2, resulting in a 4-dimensional state space. Only the nonlinear variant of the proposed algorithm could successfully learn the cost from the simple features, as shown in Figure 4.5. Even with the grid features, which do form a linear basis for the cost, MaxEnt suffered greater discretization error due to the complex dynamics of this task, while OptV

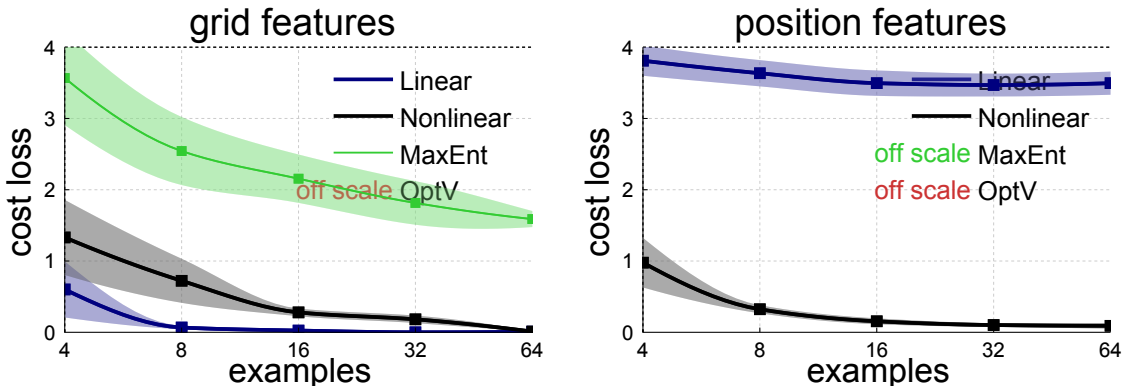


Figure 4.5: Cost loss for each algorithm with the Gaussian grid and end-effector position features on the 2-link robot arm task. Only the nonlinear variant of the proposed method could learn the cost using only the position features.

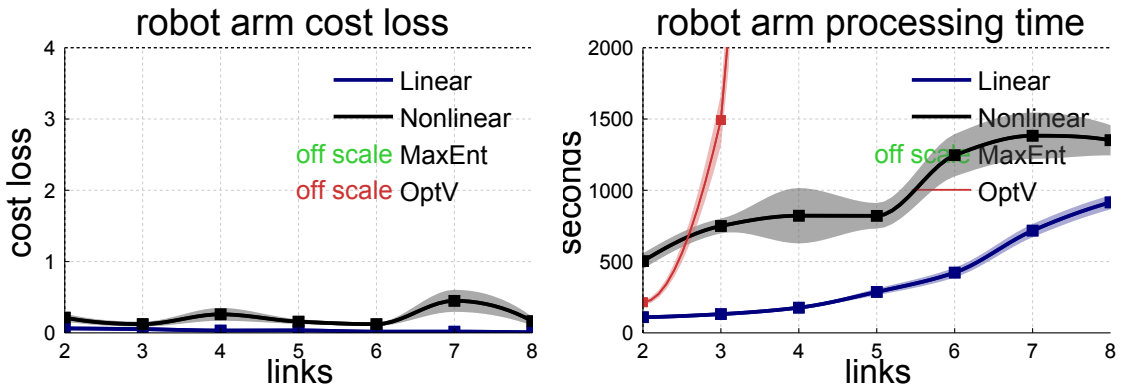


Figure 4.6: Cost loss and processing time with increasing numbers of robot arm links n, corresponding to state spaces with $2n$ dimensions. The proposed methods efficiently learn good cost functions even as the dimensionality is increased.

could not meaningfully generalize the cost function due to the increased dimensionality of the task.

4.6.3 Effects of System Dimensionality

To evaluate the effect of dimensionality, the number of robot arm links was increased from 2 to 8. As shown in Figure 4.6, the processing time of the proposed methods scaled gracefully with the dimensionality of the task, while the quality of the cost function did not deteriorate appreciably. The processing time of OptV increased exponentially due to the action space

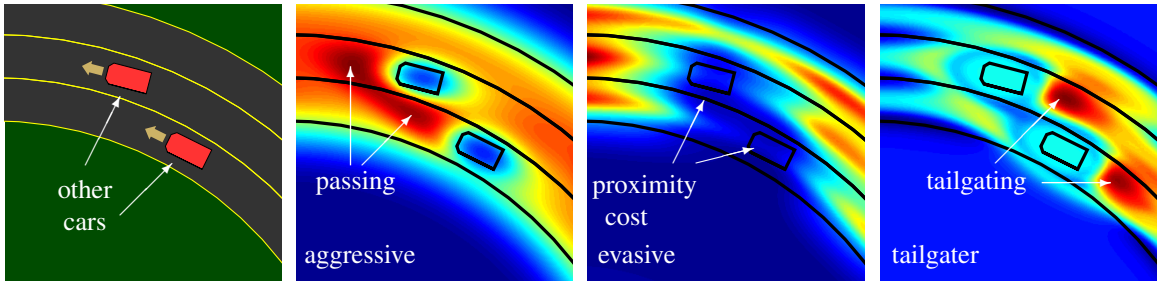


Figure 4.7: Learned driving style costs. The aggressive cost is low in front of other cars, the evasive one is high in the vicinity of each car, and the tailgater cost is low only behind the cars.

discretization. The MaxEnt discretization was intractable with more than two links, and is therefore not shown.

4.6.4 Simulated Highway Driving

The simulated driving task provides a quantitative evaluation of how well the proposed IOC algorithm can handle real human demonstrations. Although driving policies have been learned by prior IOC methods (Abbeel and Ng, 2004; Levine et al., 2011b), their discrete formulation required a discrete simulator where the agent makes simple decisions, such as choosing which lane to switch to. In constrast, the driving simulator presented in this section is a fully continuous second order dynamical system. The actions correspond directly to the gas, breaks, and steering of the simulated car, and the state space includes position, orientation, linear and angular velocities, and time, which determines the position of the other cars. The domain has two action dimensions and six state dimensions. Because of the high dimensionality, prior methods that rely on discretization cannot tractably handle this domain.

The nonlinear algorithm was used to learn from sixteen 13-second examples of an aggressive driver that cuts off other cars, an evasive driver that drives fast but keeps plenty of clearance, and a tailgater who follows closely behind the other cars. The features are speed, deviation from lane centers, and Gaussians covering the front, back, and sides of the other cars on the road. Since there is no ground truth cost for these tasks, the cost loss metric cannot be used. Instead, the resemblance of the learned policy to the demonstration is quantified by using task-relevant statistics, as in prior work (Abbeel and Ng, 2004). These

style	path	average speed	time behind	time in front
aggressive	learned	158.1 kph	3.5 sec	12.5 sec
	human	158.2 kph	3.5 sec	16.7 sec
evasive	learned	150.1 kph	7.2 sec	3.7 sec
	human	149.5 kph	4.5 sec	2.8 sec
tailgater	learned	97.5 kph	111.0 sec	0.0 sec
	human	115.3 kph	99.5 sec	7.0 sec

Table 4.1: Statistics for sample paths for the learned driving costs and the corresponding human demonstrations starting in the same initial states. The statistics of the learned paths closely resemble the holdout demonstrations.

statistics are computed by examining a number of trajectories optimized with respect to the learned cost from randomly sampled initial states. For each trajectory, the statistics include the average speed and the amount of time spent within two car-lengths behind and in front of other cars (to quantify the amount of tail-gating and passing). The statistics from the optimized trajectories are compared to those from an unobserved holdout set of user demonstrations that start from the same initial states. The results in Table 4.1 show that the statistics of the learned policies are broadly similar to the holdout demonstrations.

Plots of the learned cost functions are shown in Figure 4.7. Videos of the learned driving behaviors can be downloaded from the project website, along with source code for the IOC algorithm: `http://graphics.stanford.edu/projects/cioc`.

4.6.5 Humanoid Locomotion

In this section, the IOC algorithm is applied to learn a cost function for humanoid running in a physical simulation, which can then be used in conjunction with simple, local trajectory optimization algorithms like DDP to synthesize naturalistic running behaviors in new situations, such as on rough terrain and in the presence of perturbations. The example demonstration is obtained from motion capture of a real human run and mapped onto a humanoid skeleton with 29 degrees of freedom, corresponding to 63 state dimensions that include 4-dimensional quaternion representations of ball joints and corresponding angular velocities for every degree of freedom.

Since the motion captured demonstration only contains joint angles, the demonstration

is first tracked by using DDP with a simple cost function that minimizes the quadratic deviation from the target angles. This produces a physically consistent trajectory made up of joint angles, velocities, and torques, which can then be fed to the IOC algorithm. It should be noted that while this simple cost function is sufficient to obtain torques that realize the demonstrated run, it is not sufficient to generalize to new situations, as will be shown below.

In order to learn a cost function that captures a portable representation of the desired locomotion behavior, the IOC algorithm must be provided with a set of features that is sufficient for representing the behavior, but not so specific as to prevent generalization to new situations. For example, the quadratic deviation cost used to track the example would be a poor choice, since it would not improve generalization. The algorithm was used with a linear cost representation, and was provided with the following features:

Torque Minimization Minimization of joint torques is a common regularizer in cost functions for dynamic humanoid behaviors. It is useful to learn a seperate penalty weight on the torque at each joint, in order to allow the cost to express a preference for some joints over others. Recent work has shown that direct total torque minimization is a poor explanation for real human locomotion (Wang et al., 2012). The per-joint penalty allows the cost to capture more fine-grained joint preferences that may be caused by the dynamics of the underlying muscles or idiosyncracies of individual gaits. The joint torque feature for joint i has the following form:

$$\mathbf{f}_i(\mathbf{x}_t, \mathbf{u}_t) = -\frac{1}{2}u_{ti}^2,$$

where u_{ti} is the i^{th} component of $\mathbf{u}_t$.

Root Position and Velocity In order to make forward progress and maintain balance, the cost must be able to capture the deviation of the root link (pelvis) position and velocity from some setpoint. This is accomplished by using a single feature for the medial component of the root velocity, a single feature for the height of the root, and an additional feature for the position of the root along the transverse axis, in order to control lateral deviations. In all

cases, these features have the form

$$\mathbf{f}_j(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2}(x_{tj} - x_j^\star)^2, \tag{4.5}$$

where $x_j^\star$ is the desired setpoint. This setpoint is set to the mean value of x_{tj} in the example sequence. In the future, it would be straightforward to also learn x_{tj} as part of the IOC optimization, by employing both a linear and a quadratic feature.

Joint Angle Regularization In principle, the root position and velocity, together with torque minimization, sufficiently constrain the task to specify a forward movement with minimal effort. However, practical local trajectory optimization techniques like DDP typically fail to create a plausible locomotion behavior from such abstract objectives, as shown below. To further shape the cost and capture the unique style of the demonstrated gait, an additional set of joint angle features are used that control the degree to which each joint is allowed to deviate from a set point, which corresponds to a sort of "rest pose." The form of these features is identical to the root position and velocity features in Equation 4.5, except the index j refers to particular joint angles. The setpoints $x_j^\star$ are set to zero for all joints except the knees, corresponding to an upright standing pose. The knees are set to 0.5 radians, to encourage flexion. As with the root position and velocity, it would be straightforward to learn the target angles in the future using a set of linear and quadratic features.

Periodic Foot Motion A crucial component of bipedal locomotion is the periodic, alternating motion of the feet. This aspect of the behavior may be captured by learning Cartesian attractor positions for the feet in the sagittal plane. This is accomplished by using a set of linear and quadratic features for the medial and longitudal position of each foot. Learning the weights on the linear and quadratic features effectively acquires a target position and the weight for a quadratic deviation from that position. To create periodic motion of the feet, a different set of targets is learned for each quarter of the gait cycle, using four equal-sized intervals. This results in a total of 32 features: 4 intervals, 2 feet, 2 axes, and 2 features each (linear and quadratic). It should be noted that although these features can be interpreted as specifying a target position for the feet, they do not act as footstep constraints. Instead, they act as attractors for the feet, and the learned setpoints are often located several meters

above the ground, and thus are never reached even in the example motion, as discussed in the following section. In future work, it would be interesting to see if the phase and frequency of the gait could also be learned, to avoid the need for specifying four equal-length intervals.

Arm Swing Features To produce naturalistic arm swinging motion, an additional set of features is included that expresses the quadratic deviation of the shoulder and elbow from the reference example at each time step. Because the arm motion does not need to change drastically to adapt to new situations, such a feature does not significantly degrade the generalization of the learned cost. However, it would be worthwhile in the future to explore more abstract features for capturing periodic arm motion, perhaps in a similar spirit to the foot target features. Alternatively, a more detailed physical model, for example one that includes simple muscles, could cause naturalistic arm swinging motion to emerge naturally as a consequence of trajectory optimization.

Figure 4.8 shows plots of the original running demonstration, a running sequence generated with trajectory optimization from the learned cost function, and the result of running trajectory optimization with a naïve objective that only seeks to maintain balance and forward velocity while minimizing torques. Trajectory optimization on the new trajectory was intialized with a simple standing policy. Although the reoptimized run is not identical to the original, it exhibits many of the same overall properties, including a similar forward velocity and similar stride length. On the other hand, the naïve objective is insufficiently detailed to produce a plausible locomotion behavior, resulting in an unstable tumbling gait. The superior performance of the learned cost over this naïve baseline suggests that IOC has a shaping effect on the cost, effectively producing an objective that is more condusive to local optimization.

Table 4.2 shows the learned weights on each cost feature. From these weights, we can visualize the foot target features at each interval. These targets are visualized in Figure 4.9, by using an arrow to indicate the direction of the attractor, with length corresponding to the product of the distance to the target and the quadratic feature weight. Since the targets themselves were often several meters above the ground, they are not shown. Note that the foot targets do not act as "footstep constraints," but rather as attraction fields that guide the

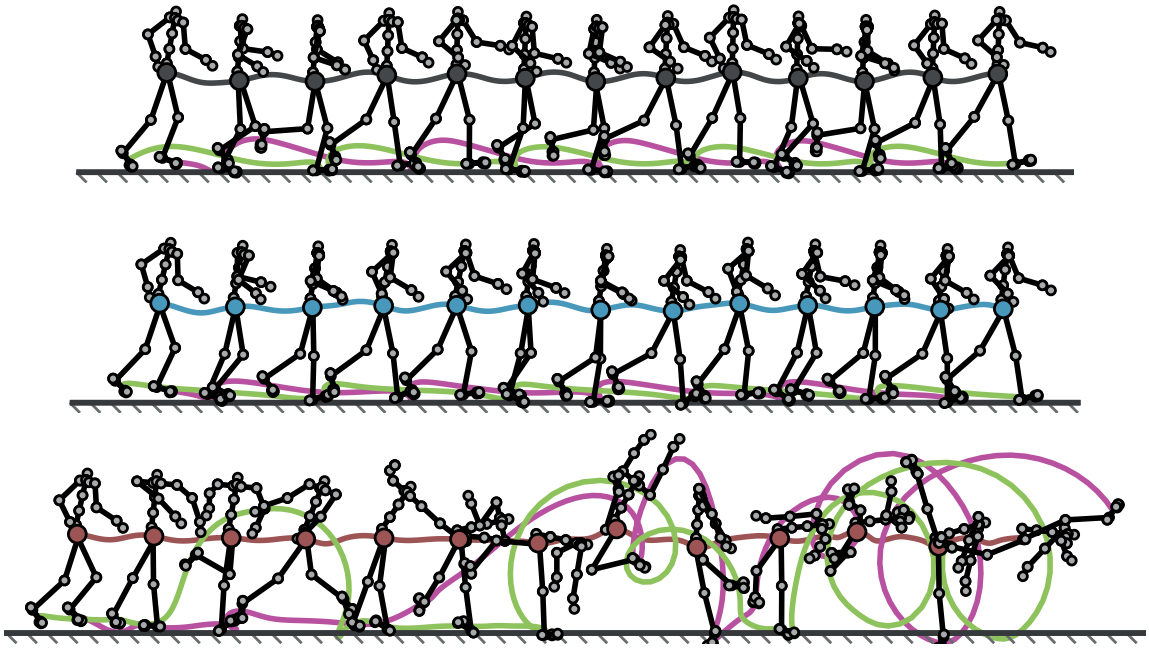


Figure 4.8: Plots of running trajectories: original human motion capture (top), optimized result using the learned cost (middle), and the result with naïve baseline cost (bottom). Colored lines indicate pelvis and foot trajectories. The learned cost produces a run that is qualitatively similar to the original motion capture.

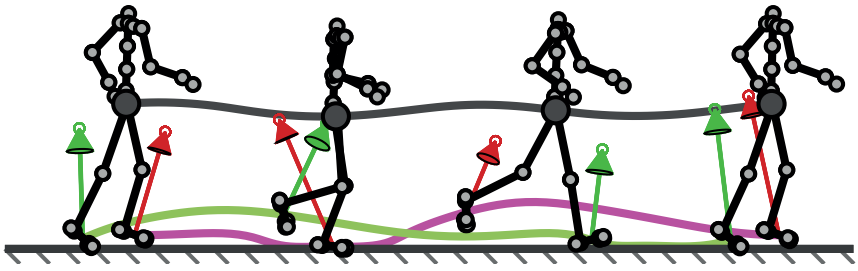


Figure 4.9: Visualization of learned foot position attractors in each of the four phases, overlaid on poses from the original example.

Feature Class	Feature	Weight
joint torque penalty	waist rotation (x, y, z)	23.7,15.6,21.1
	left shoulder rotation (x, y, z)	24.9, 24.2, 24.8
	left elbow	24.8
	right shoulder rotation (x, y, z)	24.9, 24.4, 24.9
	right elbow	24.9
	left thigh rotation (x, y, z)	25.0, 22.0, 21.5
	left knee	21.8
	left ankle rotation (y, z)	21.2, 21.2
	right thigh rotation (x, y, z)	25.0, 22.1, 21.5
	right knee	21.6
	right ankle rotation (y, z)	21.2, 21.2
root motion	forward velocity	4.93
	lateral deviation	0.0141
	height deviation	1.81
joint angle	pelvis rotation (x, y, z)	20.6, 46.0, 702
	torso rotation (x, y, z)	178, 251, 1.89
	thigh rotation (x, y, z)	24.3, 0.712, 171
	left knee	1.25
	right knee	0.463
	left ankle rotation (y, z)	3.94, 435
	right ankle rotation (y, z)	1.83, 676
foot motion	phase 1	
	left foot (x^2, x)	69.4, -87.3
	left foot (z^2, z)	-10.2, 62.7
	right foot (x^2, x)	41.6, 104
	right foot (z^2, z)	3.84, 54.9
	phase 2	
	left foot (x^2, x)	396, 127
	left foot (z^2, z)	-6.48, 7.35
	right foot (x^2, x)	77.6, 21.2
	right foot (z^2, z)	34.3, 16.9
	phase 3	
	left foot (x^2, x)	60.5, 135
	left foot (z^2, z)	29.1, 69.0
	right foot (x^2, x)	48.1, -104
	right foot (z^2, z)	13.3, 87.7
	phase 4	
	left foot (x^2, x)	123, 51.3
	left foot (z^2, z)	1.92, -18.0
	right foot (x^2, x)	562, 127
	right foot (z^2, z)	-8.71, 11.9
arm swing	shoulder and elbow deviation	2180

Table 4.2: Weights assigned to each feature by the IOC algorithm.

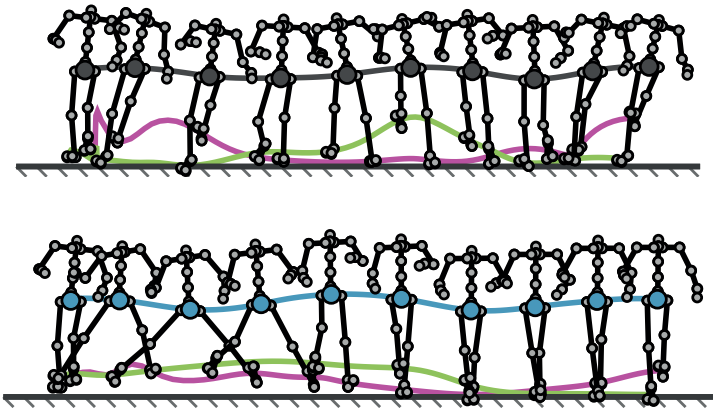


Figure 4.10: Results under lateral perturbation: quadratic tracking of the unperturbed example (top) and the learned cost function (bottom), shown from the front. A 5000 Newton push to the right is applied at the first time step. When optimizing the learned cost, the trajectory exhibits a more plausible recovery strategy, which is abscent when rigidly tracking the example motion.

feet over the course of the gait.

To test generalization, new running sequences were optimizes in the presence of a 5000 Newton, 100 ms lateral push, as well as on rough terrain. The results for the push are shown in Figure 4.10, and the rough terrain results are shown in Figure 4.11. In both tests, the results are compared to a simple baseline that minimizes the quadratic deviation from the original unperturbed, flat ground motion capture example. In the case of the lateral perturbation, note that the learned cost affords a more flexible and realistic recovery strategy, taking a large sideways step to avoid a fall. The baseline relies on razor-edge balance and remains too close to the example demonstration, producing an implausible recovery that uses very large torques to slide laterally while matching the example poses. The baseline fares better on the rough terrain, where both cost functions show reasonable generalization.

The results presented in this section indicate that the learned cost function provides sufficient detail to reproduce the example behavior, while providing a sufficiently abstract representation to allow for meaningful generalization. They also show that even very high-dimensional and dynamic behaviors such as 3D humanoid running can be optimized with simple, local trajectory optimization techniques if a sufficiently detailed cost function is

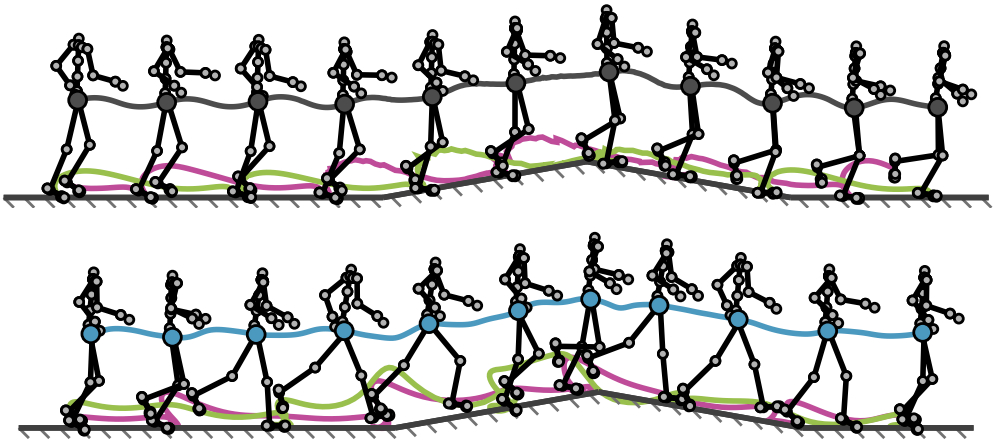


Figure 4.11: Results on rough terrain: quadratic tracking of the flat ground example (top) and the learned cost function (bottom). The learned cost function allows plausible generalization to new terrains.

learned, and that the proposed IOC algorithm is capable of learning such a cost function when provided with appropriate features.

4.7 Discussion

In this chapter, I discussed an IOC algorithm designed for continuous, high dimensional domains. The proposed method remains efficient in high dimensional state spaces by using a local, trajectory-based approximation to the cost function likelihood. This approximation also removes the global optimality requirement for the expert's demonstrations, allowing the method to learn the cost function from examples that are only locally optimal. Local optimality can be easier to demonstrate than global optimality, particularly in high dimensional domains. As shown in the evaluation, prior methods do not converge to the underlying cost function when the examples are only locally optimal, regardless of how many examples are provided. The evaluation also shows that the proposed algorithm can scale up to complex tasks like humanoid locomotion, and can handle realistic human demonstrations.

Since the algorithm relies on the derivatives of the cost features to learn the cost function, the features must be differentiable with respect to the states and actions. Unlike with forward trajectory optimization, these derivatives must be precomputed only once, since the

trajectory does not change, making the computational burden fairly low. However, features that exhibit discontinuities are still poorly suited for this method. The current formulation also only considers deterministic, fixed-horizon control problems, though Gaussian stochastic dynamics are implicitly handles as a result of the symmetry of Gaussian distributions, as discussed in the preceding chapter.

Although this method handles examples that lack global optimality, it does not make use of global optimality when it is present: the examples are always assumed to be only locally optimal. Prior methods that exploit global optimality can infer more information about the cost function from each example when the examples are indeed globally optimal.

An exciting avenue for future work is to apply this approach to a larger variety of motor skills that have previously been inaccessible for inverse optimal control methods. One challenge with such applications is to generalize and impose meaningful structure in high dimensional tasks without requiring detailed features or numerous examples. While we were able to choose an appropriate set of features for humanoid running, the particular choice of features is non-trivial. While the chosen features might also extend to other locomotion-like tasks, such as walking, drastically different tasks like jumping would require a new set of features and additional engineering effort. The nonlinear variant of the algorithm may be one way to address this challenge, but it still relies on features to generalize the cost to new situations. A more sophisticated way to construct meaningful, generalizable features would allow IOC to be easily applied to complex, high dimensional tasks.

In this chapter, trajectory optimization was used to solve the forward problem and synthesize trajectories that are optimal with respect to the learned cost. In general, performing such trajectory optimization online in new environments is impractical. In the next chapter, I will discuss how, from a known cost, we can learn closed-form controllers represented by neural networks that can execute the desired behavior in new situations without any additional optimization. While in this thesis, the problems of inverse optimal control and policy learning are considered seperately, it would be straightforward to use learned costs in combination with the algorithms in the following chapter.

Chapter 5

Guided Policy Search

Once a cost function has been learned or manually provided, we still require a mechanism for actually executing the desired motor skill in a fast-paced, real-world setting. In the previous chapter, trajectory optimization was used to construct locally optimal trajectories. While online trajectory optimization (model predictive control) is an established approach in robotics (Tassa et al., 2012), it requires expensive online replanning, requires good state estimation and does not directly handle partially observed environment, and does not make use of learning. This is in stark contrast to human or animal motor control, where habituation and learning play a central role (Rosenbaum, 1991).

An alternative to online replanning is the use of learned, parametric control policies, where a controller is learned in an offline training phase and can be used to very quickly select good learned actions online. As discussed in the introduction, the use of both hand-engineered and learned controllers has proven very successful. Parametric policies can quickly choose actions in time-sensitive domains such as motor control, and can be trained to handle partial and noisy observations, subsuming complex state estimation and modeling problems within the learned policy. However, effective methods for learning general-purpose, high-dimensional controllers are lacking. In particular, many successful learning methods rely on specialized policy classes to reduce the dimensionality of the learning problem at the cost of restricting the types of behaviors that can be learned (Ijspeert et al., 2003; Paraschos et al., 2013). In this chapter, I will consider algorithms for learning general-purpose controllers, represented by large neural networks. To tractably learn such

complex controllers without falling into poor local optima, I will combine the flexibility of trajectory optimization with the generalization capacity of parametric policies. This approach can be used learn controllers with thousands of parameters, even for complex tasks such as humanoid walking and dynamic recovery from unanticipated pushes.

I will discuss three algorithms that offer increasing amounts of integration between trajectory and policy optimization. The simplest of these methods, described in Section 5.2, simply uses a trajectory distribution to provide good samples for an importance sampling-based policy search procedure, while the more sophisticated methods co-adapt the policy and trajectory to match one another, either within the framework of variational inference in Section 5.3, or in the context of constrained optimization in Section 5.4. Although the algorithms are presented in increasing order of sophistication and generalization capacity, each approach has particular strengths and weaknesses, and a comperative analysis and discussion will be presented in Section 5.5.

5.1 Policy Search Background

In policy search, the aim is to find a policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ to control an agent in a potentially stochastic environment. The policy is a conditional distribution over the actions $\mathbf{u}_t$ at time step t, conditioned on the state $\mathbf{x}_t$. Note that in general, although the policy is conditioned on $\mathbf{x}_t$, the actual inputs to the learned policy function may include only a part of the state, for example in partially observed domains. The policy is parameterized by a vector of parameters θ. In the case of a neural network, the parameters simply correspond to the network weights.

The goal of policy search is specified by some objective. The most common objective is the minimization of the expected cost, which in a finite-horizon task has the form $J(\theta) = E_{\pi_\theta}[\sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)]$. As discussed in Chapter 3, the maximum entropy or soft optimality objective $J(\theta) = E_{\pi_\theta}[\sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)] - \mathcal{H}(\pi_\theta)$ may also be used. This idea will be revisited in Section 5.4.

Most model-free policy search algorithms use some sort of sample-based procedure to optimize the policy objective (Deisenroth et al., 2013). For example, policy gradient methods estimate the gradient of the expected cost $E[\nabla J(\theta)]$ using samples $\tau_1, \ldots, \tau_m$

drawn from the current policy π_θ, and then improve the policy by modifying the parameters according to this gradient. The gradient can be estimated using the following equation (Peters and Schaal, 2008):

$$\nabla J(\theta) = E[\ell(\tau)\nabla \log \pi_\theta(\tau)] \approx \frac{1}{m}\sum_{i=1}^{m} \ell(\tau_i)\nabla \log \pi_\theta(\tau_i),$$

where $\nabla \log \pi_\theta(\tau_i)$ decomposes to $\sum_{t=1}^{T} \nabla \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, since the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ do not depend on θ. This approach is known as the likelihood ratio method, and requires the samples to be drawn from the current policy at each gradient step. Intuitively, this algorithm makes low-cost samples more probable, and high-cost samples less probable.

Unfortunately, such a direct approach tends to work poorly for complex tasks where a successful execution of the motor skill is difficult to discover by random exploration. For example, if the goal is to construct an effective running gait, a randomly initialized neural network is unlikely to produce anything more sophisticated than falling. Samples from this falling policies will not illustrate the importance of maintain a stable gait, and increasing the probability of the best samples will only produce less costly falls. The algorithms in this chapter are referred to as guided policy search (GPS) methods, because trajectory optimization will be used to guide the policy search and avoid the need for random exploration, allowing effective running gaits to be learned even from random initialization. In the rest of the chapter, I will present three different algorithms that incorporate trajectory optimization into the policy search process.

5.2 Policy Search via Importance Sampling

A simple way to take advantage of trajectory optimization is to include good samples from an optimized trajectory distribution into the estimate of the policy's value, using a procedure similar to the one described in the previous section to increase the probability of the good samples, and decrease the probability of the bad ones. Unlike with standard reinforcement learning, such an approach would not rely on random exploration to discover effective executions of the task, but instead would include them directly from the trajectory

optimization solution. In this section, I will describe an algorithm that includes good off-policy samples into the estimator of the policy value $J(\theta)$ using importance sampling. The samples are generated from a trajectory distribution $q(\tau)$, which is constructed using trajectory optimization. If example demonstrations are available, they can be used to initialize the trajectory optimization, placing it in the basin of attraction of a good local optimum. Although the examples are only used for initialization, they can be crucial in more complex tasks, as discussed in Section 4.6.

5.2.1 Importance Sampling

Importance sampling is a technique for estimating an expectation $E_p[f(x)]$ with respect to $p(x)$ using samples drawn from a different distribution $q(x)$:

$$E_p\left[f(x)\right] = E_q\left[\frac{p(x)}{q(x)}f(x)\right] \approx \frac{1}{Z}\sum_{i=1}^{m}\frac{p(x_i)}{q(x_i)}f(x_i).$$

Importance sampling is unbiased if $Z = m$, though more often $Z = \sum_i \frac{p(x_i)}{q(x_i)}$ is used as it provides lower variance. Prior work proposed estimating $J(\theta)$ with importance sampling (Peshkin and Shelton, 2002; Tang and Abbeel, 2010). This allows using off-policy samples and results in the following estimator:

$$J(\theta) \approx \frac{1}{Z(\theta)}\sum_{i=1}^{m}\frac{\pi_\theta(\tau_i)}{q(\tau_i)}\ell(\tau_i). \tag{5.1}$$

The variance of this estimator can be reduced further by observing that past costs do not depend on future actions (Sutton et al., 1999; Baxter et al., 2001):

$$J(\theta) \approx \sum_{t=1}^{T}\frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\pi_\theta(\tau_{i,1:t})}{q(\tau_{i,1:t})}\ell\left(\mathbf{x}_t^i, \mathbf{u}_t^i\right), \tag{5.2}$$

where $\pi_\theta(\tau_{i,1:t})$ denotes the probabiliy of the first t steps of τ_i, and $Z_t(\theta)$ normalizes the weights. To include samples from multiple distributions, a fused distribution of the form $q(\tau) = \frac{1}{n}\sum_j q_j(\tau)$ may be used, where each q_j is a different sampling distribution.

Previous methods have employed this estimator to include off-policy samples from previous policies, so as to enable sample reuse between iterations (Peshkin and Shelton, 2002; Tang and Abbeel, 2010). In guided policy search (GPS), importance sampling is used both to include samples from previous policies, and to include guiding samples from a guiding distribution constructed with trajectory optimization. The construction of the guiding distribution is discussed in the following subsection.

Prior methods optimized Equation 5.1 or 5.2 directly to improve $J(\theta)$. Unfortunately, with complex policies and long rollouts, this often produces poor results, as the estimator only considers the relative probability of each sample and does not require any of these probabilities to be high. The optimum can assign a low probability to all samples, with the best sample slightly more likely than the rest, thus receiving the only nonzero weight. Tang and Abbeel (2010) attempted to mitigate this issue by constraining the optimization by the variance of the weights. However, this is ineffective when the distributions are highly peaked, as is the case with long rollouts, because a very small fraction of samples have nonzero weights, and their variance tends to zero as the sample count increases. As shown in the evaluation, this problem can be very common in large, high-dimensional tasks.

To address this issue, Equation 5.2 can be augmented with a novel regularizing term. A variety of regularizers are possible, but a simple and effective option is to use the logarithm of the normalizing constant:

$$\Phi(\theta) = \sum_{t=1}^{T} \left[\frac{1}{Z_t(\theta)} \sum_{i=1}^{m} \frac{\pi_\theta(\tau_{i,1:t})}{q(\tau_{i,1:t})} \ell(\mathbf{x}_t^i, \mathbf{u}_t^i) - w_r \log Z_t(\theta) \right]. \tag{5.3}$$

This objective is minimized using an analytic gradient derived in Appendix B. It is easy to check that this estimator is consistent, since $Z_t(\theta) \to 1$ in the limit of infinite samples. The regularizer acts as a soft maximum over the logarithms of the weights, ensuring that at least some samples have a high probability under π_θ. Furthermore, by adaptively adjusting w_r, the algorithm can control how far the policy is allowed to deviate from the samples, which can be used to limit the optimization to regions that are better represented by the samples if it repeatedly fails to make progress.

5.2.2 Guiding Samples

In this subsection, I will motivate and describe the particular choice of sampling distribution $q(\tau)$ used in the guided policy search algorithm. As mentioned in the previous subsection, prior methods employed importance sampling to reuse samples from previous policies (Peshkin and Shelton, 2002; Kober and Peters, 2009; Tang and Abbeel, 2010). However, when learning policies with hundreds of parameters, local optima make it very difficult to find a good solution, and to effectively optimize high-dimensional policies for complex tasks like locomotion, we must use off-policy guiding samples drawn from a guiding trajectory distribution that illustrates low-cost executions of the task.

An effective guiding distribution covers low-cost regions while avoiding large $q(\tau)$ densities, which result in low importance weights. This suggests that a good guiding distribution is an I-projection of $\rho(\tau) \propto \exp(\ell(\tau))$, which is a distribution q that minimizes the KL-divergence $D_{\mathrm{KL}}(q\|\rho) = E_q[\ell(\tau)] - \mathcal{H}(q)$. Minimizing the KL-divergence produces a distribution with low average cost due to the first term, and high entropy due to the second. As discussed in Section 3.4, we can construct an approximate Gaussian I-projection of ρ by using a variant of differential dynamic programming (DDP). This approximation corresponds to the Laplace approximation, and consists of the following conditional distribution over actions at each time step:

$$\pi_{\mathcal{G}}(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, Q^{-1}_{\mathbf{u},\mathbf{u}t}),$$

where $\mathbf{K}_t$, $\mathbf{k}_t$, and $Q_{\mathbf{u},\mathbf{u}t}$ are computed using DDP. It should be noted that $\pi_{\mathcal{G}}(\tau)$ is only truly Gaussian under linear dynamics. When the dynamics are nonlinear, $\pi_{\mathcal{G}}(\tau)$ approximates a Gaussian around the nominal trajectory. Fortunately, the feedback term usually keeps the samples close to this trajectory, making them suitable guiding samples for the policy search. The nominal trajectory around which this approximation is constructed can itself be optimized with multiple iterations of DDP, and in general may be initialized from example.

While this sampling distribution captures low-cost regions, it does not consider the current policy π_θ. We can adapt it to π_θ by sampling from an I-projection of $\rho_\theta(\tau) \propto \exp(-\ell(\tau))\pi_\theta(\tau)$, which is the optimal distribution for estimating $E_{\pi_\theta}[\exp(-\ell(\tau))]$.[1] As

[1]While we could also consider the optimal sampler for $E_{\pi_\theta}[\ell(\tau)]$, such a sampler would also need to

we will see in the next section, this distribution has a special meaning in the context of variational inference. However, in the context of importance sampling, its choice is largely informal.

An approximate I-projection of ρ_θ can be constructed simply by running the DDP algorithm with the augmented cost function $\tilde{\ell}(\mathbf{x}_t, \mathbf{u}_t) = \ell(\mathbf{x}_t, \mathbf{u}_t) - \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. The resulting distribution is then an approximate I-projection of $\rho_\theta(\tau) \propto \exp(-\tilde{\ell}(\tau)) = \exp(-\ell(\tau))\pi_\theta(\tau)$. In practice, many domains do not require adaptive samples. Adaptation becomes necessary when the initial samples cannot be reproduced by any policy, such as when they act differently in similar states, or when the representational power of the policy is limited either by size constraints, or by partial observability. In that case, adapted samples will avoid different actions in similar states, making them more suited for guiding the policy. A more detailed discussion of adaptation is provided in the sections on variational and constrained GPS, as well as the experimental evaluation.

The guiding samples are incorporated into the policy search by building one or more initial DDP solutions and supplying the resulting samples to the importance sampled policy search algorithm. These solutions can be initialized with human demonstrations or with an offline planning algorithm. When learning from demonstrations, we can perform just one step of DDP starting from the example demonstration, thus constructing a Gaussian distribution around the example. If adaptive guiding distributions are used, they are constructed at each iteration of the policy search starting from the previous DDP solution.

Although the policy search component is model-free, DDP requires a model of the system dynamics. Numerous recent methods have proposed to learn the model (Abbeel et al., 2006; Deisenroth and Rasmussen, 2011; Ross and Bagnell, 2012), and if we use initial examples, only local models are required.

One might also wonder why the DDP policy $\pi_\mathcal{G}$ is not itself a suitable controller. The issue is that $\pi_\mathcal{G}$ is time-varying and only valid around a single trajectory, while the final policy can be learned from many DDP solutions in many situations. Guided policy search can be viewed as transforming a collection of trajectories into a controller. This controller

also cover regions with very low reward, which are often very large and would not be represented well by a Gaussian I-projection.

Algorithm 1 Importance Sampled Guided Policy Search

1: Generate DDP solutions $\pi_{\mathcal{G}_1}, \ldots, \pi_{\mathcal{G}_n}$
2: Sample $\tau_1, \ldots, \tau_m$ from $q(\tau) = \frac{1}{n} \sum_i \pi_{\mathcal{G}_i}(\tau)$
3: Initialize $\theta^\star \leftarrow \arg\max_\theta \sum_i \log \pi_{\theta^\star}(\tau_i)$
4: Build initial sample set $\mathcal{S}$ from $\pi_{\mathcal{G}_1}, \ldots, \pi_{\mathcal{G}_n}, \pi_{\theta^\star}$
5: **for** iteration $k = 1$ to K **do**
6: Choose current sample set $\mathcal{S}_k \subset \mathcal{S}$
7: Optimize $\theta_k \leftarrow \arg\max_\theta \mathcal{L}_{\mathcal{S}_k}(\theta)$
8: Append samples from π_{θ_k} to $\mathcal{S}_k$ and $\mathcal{S}$
9: Optionally generate adaptive guiding samples
10: Estimate the values of π_{θ_k} and $\pi_{\theta^\star}$ using $\mathcal{S}_k$
11: **if** π_{θ_k} is better than $\pi_{\theta^\star}$ **then**
12: Set $\theta^\star \leftarrow \theta_k$
13: Decrease w_r
14: **else**
15: Increase w_r
16: Optionally, resample from $\pi_{\theta^\star}$
17: **end if**
18: **end for**
19: Return the best policy $\pi_{\theta^\star}$

can adhere to any parameterization, reflecting constraints on computation or available sensors in partially observed domains. In the experimental evaluation, I will show that such a policy generalizes to situations where the time-varying DDP policy fails.

5.2.3 Importance Sampled Guided Policy Search

Algorithm 1 summarizes the importance-sampled GPS algorithm. The overall objective is to optimize the expected return $J(\theta) = E_{\pi_\theta}[\sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)]$ with respect to the policy parameters θ by means of trajectory optimization and importance sampling. On line 1, the algorithm builds one or more DDP solutions, which can be initialized from demonstrations. Initial guiding samples are sampled from these solutions and used on line 3 to pretrain the initial policy $\pi_{\theta^\star}$. Since the samples are drawn from stochastic feedback policies, $\pi_{\theta^\star}$ can already learn useful feedback rules during this pretraining stage. The sample set $\mathcal{S}$ is constructed on line 4 from the guiding samples and samples from $\pi_{\theta^\star}$, and the policy search then alternates between optimizing $\Phi(\theta)$ and gathering new samples from the current policy

π_{θ_k}. If the sample set $\mathcal{S}$ becomes too big, a subset $\mathcal{S}_k$ is chosen on line 6. In practice, we can simply choose the samples with high importance weights under the current best policy $\pi_{\theta^\star}$, as well as the guiding samples.

The objective $\Phi(\theta)$ is optimized on line 7 with LBFGS. This objective can itself be susceptible to local optima: when the weight on a sample is very low, it is effectively ignored by the optimization. If the guiding samples have low weights, the optimization cannot benefit from them. The issue of local optima is also analyzed in more detail in the following subsection. We can partially mitigate this issue by repeating the optimization twice, once starting from the best current parameters $\theta^\star$, and once by initializing the parameters with an optimization that maximizes the log weight on the highest-reward sample. While this approach is not guaranteed to avoid bad local optima, it produces a substantial improvement in practice. Prior work suggested restarting the optimization from each previous policy (Tang and Abbeel, 2010), but this is very slow with complex policies, and still fails to explore the guiding samples, for which there are no known policy parameters.

Once the new policy π_{θ_k} is optimized, the algorithm adds samples from π_{θ_k} to $\mathcal{S}$ on line 8. If using adaptive guiding samples, the adaptation is done on line 9 and new guiding samples are also added. Equation 5.2 is then used to estimate the returns of both the new policy and the current best policy $\pi_{\theta^\star}$ on line 10. Since $\mathcal{S}_k$ now contains samples from both policies, this estimation is expected to be reasonably accurate. If the new policy is better, it replaces the best policy. Otherwise, the regularization weight w_r is increased. Higher regularization causes the next optimization to stay closer to the samples, making the estimated return more accurate. Once the policy search starts making progress, the weight is decreased. In practice, w_r is clamped between 10^{-2} and 10^{-6} and adjusted by a factor of 10. The policy can also fail to improve if the samples from $\pi_{\theta^\star}$ had been unusually good by chance. To address this, additional samples from $\pi_{\theta^\star}$ may occasionally be drawn on line 16 to prevent the policy search from getting stuck due to an overestimate of the best policy's value. In the tested implementation, this resampling was done if the policy failed to improve for four consecutive iterations.

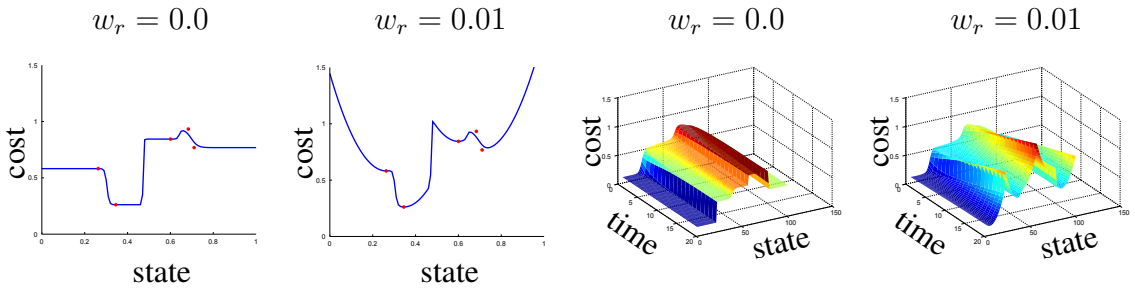


Figure 5.1: Toy example illustrating the importance-sampled GPS objective landscape $\Phi(\theta)$. The left side shows a one-dimensional state space with a time horizon of 1, with samples denoted by dots and the estimated value $\Phi(\theta)$ denoted by the blue line. The right side extends this plot over a longer time horizon, illustrating how the local optima and plateaus become more pronounced as the trajectory dimensionality increases.

5.2.4 Limitations of Importance Sampling

Although the importance-sampled guided policy search algorithm can be used to train effective policies even for complex humanoid locomotion tasks, as shown in Section 5.5, it suffers from a number of limitations that will be addressed by the other algorithms discussed in this chapter.

The underlying cause for many of these limitations is the susceptibility of the importance-sampled objective $\Phi(\theta)$ to local optima. When a particular sample receives a low importance weight, it does not figure in the objective and is ignored, even if it has a very low cost. The simple heuristic described in the previous section uses a second initialization for the optimization that is constructed by directly maximizing the weight on the best sample. While this heuristic can alleviate local optima in some cases, for example when the best guiding sample has a low weight, it is only a heuristic, and is ineffective when, for example, the best sample cannot be reproduced by any policy.

We can develop a better intuition for the nature of these local optima by using a simple toy example, shown in Figure 5.1. In this example, the state space has only one dimension, so we can plot it on the horizontal axis. The policy is a fixed-width Gaussian with a single parameter: its position on the horizontal axis. We can therefore graph the estimated value of the policy as a function of the position of this Gaussian. The two graphs on the left show this plot for two regularization values: 0.0 (no regularization) and 0.01. When the regularizer is used, the estimated value clearly favors positions near sample points, as desired, but

also tends to place each sample at the bottom of its own local optimum. The density of these local optima depends on the number of samples, the width of the policy, and the regularization weight, but its apparent that the local optima problem is quite severe even in this simple one-dimensional example. Furthermore, as can be seen from the unregularized graph with a weight of 0.0, disabling the regularizer does not solve the problem: instead of local optima, we end up with large flat plateaus, which also make optimization extremely difficult.

The left side of Figure 5.1 shows the objective for a time horizon of 1. If we instead see what happens with longer time horizons on the right side of the figure, we notice that, as the dimensionality of the trajectory increases, the probabilities $\pi_\theta(\tau)$ become more peaked due to the higher dimensionality, resulting in deeper local optima and sharper plateaus. This suggests that the optimization problem increases in difficulty for longer time horizons, which is an additional and serious challenge.

One of the symptoms of these local optima is that the adaptation procedure discussed in Section 5.2.2 is not always effective. In tasks where no policy from the policy class can reproduce the example demonstration – for example due to partial observability, limited representation size, or a low-quality, inconsistent example demonstration, – adaptation is necessary to modify the guiding distribution to produce useful guiding samples. However, this modified guiding distribution is likely to be higher in cost than the initial one, as it trades off optimality in favor of being representable by the policy. When this happens, the best-sample initialization heuristic will be ineffective for pulling the policy out of poor local optima, as the best samples will come from the initial, irreproducible distribution, rather than the adapted one. While we could of course modify the heuristic to instead initialize to the adapted samples, the problem of local optima cannot in general be solved effectively so long as the objective landscape has so many pits and plateaus. In the next two sections, I will describe two other guided policy search algorithms that use a more principled approach to adapt the trajectory demonstration to the policy in such a way that a simple supervised learning objective can be employed instead, which does not suffer from so many local optima.

5.3 Policy Search via Variational Inference

To address the problem of local optima and avoid the need for importance sampling, we can make the following observations: although direct optimization of the policy cost is difficult, optimizing a trajectory distribution is relatively easy, because it allows us to take advantage of the dynamics model and decomposes the parameter values across time steps, allowing the use of simple dynamic programming algorithms. It is therefore advantageous to offload as much of the optimization work as possible onto the trajectory optimization procedure, making the policy optimization relatively simple. In this section, I will discuss a method where all exploration and temporal structure is handled with trajectory optimization, while the policy is trained to match this distribution in a maximum likelihood sense, using standard supervised learning techniques. By carefully arranging the division of labor between policy and trajectory optimization, we can devise a principled variational algorithm with local convergence guarantees and without the need for importance sampling. It turns that this can be done by using the adaptation procedure described in the previous section, where the trajectory is optimized for both minimum cost and maximum probability under the policy, but using a simple supervised procedure for training the policy. The resulting method is equivalent to a variational optimization of a maximum likelihood policy objective. This objective differs somewhat from the more standard expected cost objective, and the differences are discussed further at the end of this section.

5.3.1 Variational Policy Search

In the previous section, the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ was optimized to minimize the expected cost $E[\ell(\tau)]$, where the expectation is taken with respect to the system dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ and the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. An alternative to this standard formulation is to convert the task into an inference problem, by using a variant of the graphical model discussed in Section 3.3. This model relates the state $\mathbf{x}_{t+1}$ to the previous state and action via the dynamics, relates stations to actions via the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, and introduces binary variables $\mathcal{O}_t$ at each time step to serve as indicators for "optimality." The distribution over these variables is defined as $p(\mathcal{O}_t = 1|\mathbf{x}_t, \mathbf{u}_t) = \exp(-\ell(\mathbf{x}_t, \mathbf{u}_t))$. By settings $\mathcal{O}_t = 1$ at all time steps and learning the maximum likelihood values for θ, we can perform maximum likelihood policy

optimization, as suggested in previous work (Toussaint et al., 2008). The corresponding optimization problem has the objective

$$\begin{aligned} p(\mathcal{O}|\theta) &= \int p(\mathcal{O}|\tau)p(\tau|\theta)d\tau \\ &= \int \exp\left(-\sum_{t=1}^{T}\ell(\mathbf{x}_t,\mathbf{u}_t)\right)p(\mathbf{x}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)d\tau, \end{aligned} \tag{5.4}$$

where we use $p(\mathcal{O})$ as shorthand for $p(\mathcal{O}_{1,\ldots,T}=1)$. Note that this objective is not equivalent either to minimizing the expected cost nor to minimizing the soft optimality objective in Section 3.3. Instead, this objective minimizes the expected exponential cost, which has a slightly different meaning, as will be discussed later in this section.

Following prior work (Neumann, 2011), we can decompose $\log p(\mathcal{O}|\theta)$ by using a variational distribution $q(\tau)$:

$$\log p(\mathcal{O}|\theta) = \mathcal{L}(q,\theta) + D_{\mathrm{KL}}(q(\tau)\|p(\tau|q,\theta)),$$

where the variational lower bound $\mathcal{L}$ is given by

$$\mathcal{L}(q,\theta) = \int q(\tau)\log\frac{p(\mathcal{O}|\tau)p(\tau|\theta)}{q(\tau)}d\tau,$$

and the second term is the Kullback-Leibler (KL) divergence

$$D_{\mathrm{KL}}(q(\tau)\|p(\tau|\mathcal{O},\theta)) = -\int q(\tau)\log\frac{p(\tau|\mathcal{O},\theta)}{q(\tau)}d\tau = -\int q(\tau)\log\frac{p(\mathcal{O}|\tau)p(\tau|\theta)}{q(\tau)p(\mathcal{O}|\theta)}d\tau. \tag{5.5}$$

We can then optimize the maximum likelihood objective in Equation 5.4 by iteratively minimizing the KL divergence with respect to $q(\tau)$ and maximizing the bound $\mathcal{L}(q,\theta)$ with respect to θ. This is the standard formulation for expectation maximization (EM) (Koller and Friedman, 2009), and has been applied to policy optimization in previous work (Kober and Peters, 2009; Vlassis et al., 2009; Furmston and Barber, 2010; Neumann, 2011). The E-step in EM minimizes the KL divergence with respect to $q(\tau)$, while the M-step maximizes the variational lower bound with respect to θ. Because the KL-divergence is

always positive, the bound $\mathcal{L}$ is indeed a lower bound on the log likelihood, and the lower the KL-divergence, the tighter the bound becomes.

Prior policy optimization methods typically represent $q(\tau)$ by sampling trajectories from the current policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ and reweighting them, for example by the exponential of their cost. While this can improve policies that already visit regions of low cost, it relies on random policy-driven exploration to discover those low cost regions. The variational GPS algorithm instead to directly optimizes $q(\tau)$ to minimize both its expected cost and its divergence from the current policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. For a Gaussian distribution $q(\tau)$, the KL divergence in Equation 5.5 can be minimized using exactly the same adaptation procedure as discussed in the previous section.

5.3.2 Variational E-Step

The E-step minimizes the KL-divergence in Equation 5.5 for a particular class of trajectory distributions $q(\tau)$. By using a Gaussian distribution, the DDP algorithm can be adapted to perform this optimization. Rewriting the KL-divergence and omitting any terms that do not depend on τ, we get

$$D_{\mathrm{KL}}(q(\tau)\|p(\tau|\mathcal{O},\theta)) = \int q(\tau)\left[-\log p(\mathcal{O}|\tau) - \log p(\tau|\theta) + q(\tau)\right] d\tau$$
$$= \int q(\tau)\left[\sum_{t=1}^{T} \ell(\mathbf{x}_t,\mathbf{u}_t) - \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t) - \log p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)\right] d\tau - \mathcal{H}(q).$$

If we fix $q(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t) \approx p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)$, for example by using a locally linear Gaussian approximation to the dynamics, minimizing the KL divergence becomes equivalent to the soft optimality objective with the augmented cost function $\tilde{\ell}(\mathbf{x}_t,\mathbf{u}_t) = \ell(\mathbf{x}_t,\mathbf{u}_t) - \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, just like in the adapted variant of the importance-sampled GPS algorithm, and the solution corresponds to the same time-varying linear Gaussian policy $\pi_G(\mathbf{u}_t|\mathbf{x}_t)$. This dynamics constraint is reasonable when the dynamics are either deterministic or have low variance. When the dynamics have higher variance, we can optimize $q(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)$ as well, but this tends to produce overly optimistic policies, as it allows $q(\tau)$ to "choose" the dynamics that produce the lowest cost and lie close to the true dynamics.

Once we compute $\pi_{\mathcal{G}}(\mathbf{u}_t|\mathbf{x}_t)$ using DDP, it is straightforward to obtain the marginal distributions $q(\mathbf{x}_t)$, which will be useful in the next section for minimizing the variational bound $\mathcal{L}(q, \theta)$. Using μ_t and Σ_t to denote the mean and covariance of the marginal at time t and assuming that the initial state distribution at $t = 1$ is given, the marginals can be computed recursively as

$$\mu_{t+1} = \begin{bmatrix} f_{\mathbf{x}t} & f_{\mathbf{u}t} \end{bmatrix} \begin{bmatrix} \mu_t \\ \mathbf{k}_t + \mathbf{K}_t \mu_t \end{bmatrix}$$

$$\Sigma_{t+1} = \begin{bmatrix} f_{\mathbf{x}t} & f_{\mathbf{u}t} \end{bmatrix} \begin{bmatrix} \Sigma_t & \Sigma_t \mathbf{K}_t^{\mathrm{T}} \\ \mathbf{K}_t \Sigma_t & Q_{\mathbf{u},\mathbf{u}t}^{-1} + \mathbf{K}_t \Sigma_t \mathbf{K}_t^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} f_{\mathbf{x}t} & f_{\mathbf{u}t} \end{bmatrix}^{\mathrm{T}} + \mathbf{F}_t,$$

where $f_{\mathbf{x}t}$ and $f_{\mathbf{u}t}$ again denote the derivatives of the dynamics mean, $\mathbf{F}_t$ is the covariance of the Gaussian dynamics distribution, and the feedback matrices $\mathbf{K}_t$ and $\mathbf{k}_t$, together with the Q-function matrix $Q_{\mathbf{u},\mathbf{u}t}$, are obtained from DDP. As noted before, when the dynamics are nonlinear or the modified cost $\tilde{\ell}(\mathbf{x}_t, \mathbf{u}_t)$ is nonquadratic, this solution only approximates the minimum of the KL divergence. In practice, the approximation is quite good when the dynamics and the cost $\ell(\mathbf{x}_t, \mathbf{u}_t)$ are smooth. Unfortunately, the policy term $\log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ in the modified cost $\tilde{\ell}(\mathbf{x}_t, \mathbf{u}_t)$ can be quite jagged early on in the optimization, particularly for nonlinear policies. To mitigate this issue, the derivatives of the policy can be computed not only along the current nominal trajectory, but also at samples drawn from the current marginals $q(\mathbf{x}_t)$, and average them together. This averages out local perturbations in $\log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ and improves the approximation.

The variational E-step serves to minimize the KL-divergence in Equation 5.5. This minimization makes the lower bound $\mathcal{L}$ a better approximation of the true log-likelihood $\log p(\mathcal{O}|\theta)$. In the M-step discussed in the next subsection, this bound is maximizes with respect to the policy parameters θ as a proxy for the maximization of the log-likelihood.

5.3.3 Variational M-Step

The variational GPS algorithm alternates between minimizing the KL divergence in Equation 5.5 with respect to $q(\tau)$ as described in the previous section, and maximizing the bound $\mathcal{L}(q, \theta)$ with respect to the policy parameters θ. Minimizing the KL divergence reduces the

difference between $\mathcal{L}(q, \theta)$ and $\log p(\mathcal{O}|\theta)$, so that the maximization of $\mathcal{L}(q, \theta)$ becomes a progressively better approximation for the maximization of $\log p(\mathcal{O}|\theta)$. The bound $\mathcal{L}(q, \theta)$ can be maximized by a variety of standard optimization methods, such as stochastic gradient descent (SGD) or LBFGS. The gradient is given by

$$\nabla \mathcal{L}(q, \theta) = \int q(\tau) \sum_{t=1}^{T} \nabla \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t) d\tau \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla \log \pi_\theta(\mathbf{u}_t^i|\mathbf{x}_t^i), \tag{5.6}$$

where the samples $(\mathbf{x}_t^i, \mathbf{u}_t^i)$ are drawn from the marginals $q(\mathbf{x}_t, \mathbf{u}_t)$. When using SGD, new samples can be drawn at every iteration, since sampling from $q(\mathbf{x}_t, \mathbf{u}_t)$ only requires the precomputed marginals from the preceding section. Because the marginals are computed using linearized dynamics, we can be assured that the samples will not deviate drastically from the optimized trajectory, regardless of the true dynamics. The resulting SGD optimization is analogous to a supervised learning task with an infinite training set. When using LBFGS, a new sample set can be generated every n LBFGS iterations. Values of n from 20 to 50 tend to produce a good tradeoff between convergence and speed.

The policies employed in the experiments in this chapter are Gaussian, with the mean given by a neural network and a fixed covariance. In this case, we can optimize the policy more quickly and with many fewer samples by only sampling states and evaluating the integral over actions analytically. Letting $\mu_{\mathbf{x}_t}^\pi, \Sigma_{\mathbf{x}_t}^\pi$ and $\mu_{\mathbf{x}_t}^q, \Sigma_{\mathbf{x}_t}^q$ denote the means and covariances of $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ and $q(\mathbf{u}_t|\mathbf{x}_t)$, we can write $\mathcal{L}(q, \theta)$ as

$$\begin{aligned}
\mathcal{L}(q, \theta) &\approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \int q(\mathbf{u}_t|\mathbf{x}_t^i) \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t^i) d\mathbf{u}_t + \text{const} \\
&= \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} -\frac{1}{2} \left(\mu_{\mathbf{x}_t^i}^\pi - \mu_{\mathbf{x}_t^i}^q\right)^{\mathrm{T}} \Sigma_{\mathbf{x}_t}^{\theta-1} \left(\mu_{\mathbf{x}_t^i}^\pi - \mu_{\mathbf{x}_t^i}^q\right) - \frac{1}{2} \log \left|\Sigma_{\mathbf{x}_t^i}^\pi\right| - \frac{1}{2} \mathrm{tr}\left(\Sigma_{\mathbf{x}_t^i}^{\pi-1} \Sigma_{\mathbf{x}_t^i}^q\right) + \text{const}.
\end{aligned}$$

This equation can then be differentiated analytically with respect to the policy parameters θ and optimizing using the same SGD or LBFGS procedure as before.

5.3.4 Variational Guided Policy Search

The complete variational GPS algorithm is summarized in Algorithm 2. As discussed

Algorithm 2 Variational Guided Policy Search

1: Initialize $q(\tau)$ using DDP with cost $\tilde{\ell}(\mathbf{x}_t, \mathbf{u}_t) = \alpha_0 \ell(\mathbf{x}_t, \mathbf{u}_t)$
2: **for** iteration $k = 1$ to K **do**
3: Compute marginals $(\mu_1, \Sigma_t), \ldots, (\mu_T, \Sigma_T)$ for $q(\tau)$
4: Optimize $\mathcal{L}(q, \theta)$ with respect to θ using standard nonlinear optimization methods
5: Set α_k based on annealing schedule, for example $\alpha_k = \exp\left(\frac{K-k}{K} \log \alpha_0 + \frac{k}{K} \log \alpha_K\right)$
6: Optimize $q(\tau)$ using DDP with cost $\tilde{\ell}(\mathbf{x}_t, \mathbf{u}_t) = \alpha_k \ell(\mathbf{x}_t, \mathbf{u}_t) - \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t)$
7: **end for**
8: Return optimized policy $\pi_\theta(\mathbf{u}_t | \mathbf{x}_t)$

previous, the algorithm alternates between the E-step, which minimizes the KL-divergence in Equation 5.5, and the M-step, which optimizes the policy. In general, a single policy can be optimized from multiple initial states and under different conditions, in which case multiple trajectories are optimized in the E-step, while the M-step samples from all trajectory distributions to train a single policy.

An additional detail that must be considered to obtain good results is that, unlike the average cost objective, the maximum likelihood objective is sensitive to the magnitude of the cost. Specifically, the logarithm of Equation 5.4 corresponds to a soft minimum over all likely trajectories under the current policy, with the softness of the minimum inversely proportional to the cost magnitude. As the magnitude increases, this objective scores policies based primarily on their best-case cost, rather than the average case. This is sort of "optimistic" optimization can be desirable early on, when the goal is to find a policy that succeeds even occasionally, but can become a serious problem at the end of the optimization, when it interferes with finding a policy that has good average case performance. As discussed in the next subsection, this problem is a fundamental limitation of this algorithm, but we can mitigate the problem to some degree by gradually annealing the cost by multiplying it by α_k at the k^{th} iteration, starting with a high magnitude to favor aggressive exploration, and ending with a low magnitude to avoid an overly risky final policy. A reasonable schedule is to start with α_k set to 1 and reduce it exponentially to 0.1 by the 50^{th} iteration of the EM procedure.

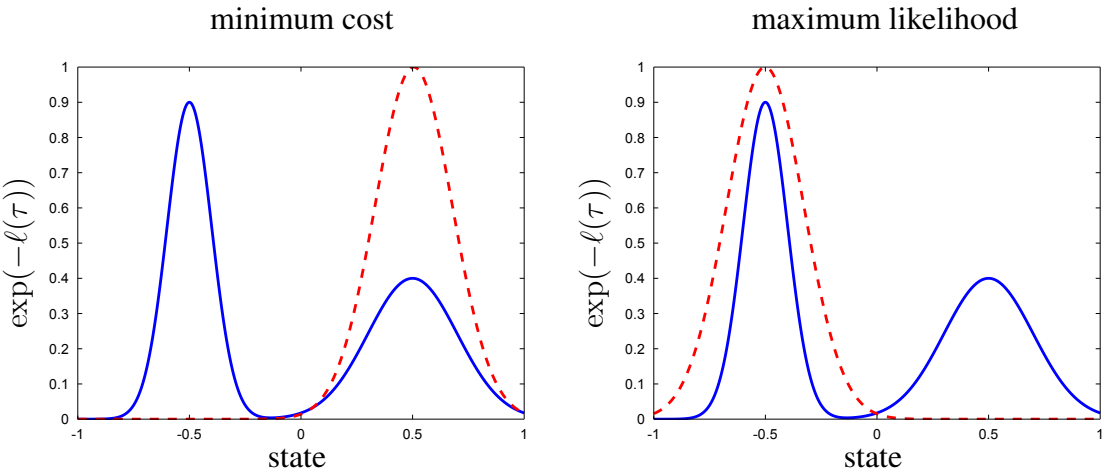


Figure 5.2: Toy example illustrating the limitations of a risk-seeking maximum likelihood objective. The blue line indicates the exponential of the negative cost plotted on a one-dimensional state space, with the optimal policy under the expected cost shown on the left, and the maximum likelihood policy shown on the right. Note that the cost near the bottom of the plot approaches negative infinity, as its exponential approaches zero, making the maximum likelihood policy very high in cost.

5.3.5 Limitations of Variational Policy Search

Variational GPS mitigates some of the local optima challenges associated with the previously discussed importance-sampling algorithm, since the probability of the sample no longer figures in the objective. The method is still vulnerable to local optima due to nonlinear dynamics, but their effect is not as severe in practice. This is demonstrated in Section 4.6 by the comparison between variational GPS and adapted GPS. Since adapted GPS includes a trajectory adaptation step that is identical to the variational E-step, any difference between the performance of the two methods is due to the difference in the policy optimization objective, which in the case of importance-sampling is much more multimodal.

However, the variational approach also introduces new limitations that stem from the form of the maximum likelihood policy objective. The objective in Equation 5.4 does not optimize the expected cost of the policy, but rather the expected exponential cost. As discussed in the preceding section, this causes the policy to be risk-seeking, since only the best-case performance of the policy is optimized. This problem can again be visualized by means of a toy example, shown in Figure 5.2, where the state is again one-dimensional,

and the policy is a fixed-width Gaussian that is positioned on the horizontal axis. In the figure, the vertical axis shows the exponential of the negative cost, which means that the cost (shown in blue) near the bottom of the graph approaches negative infinity. The width of the policy Gaussian (shown in red) is chosen to be too wide to fit onto the tall peak. Under the expected cost objective, the optimal placement of the policy therefore falls on the short peak, since placing it on the tall peak would incur a very large cost from the regions where the cost approaches negative infinity. The exponential cost maximum likelihood objective places the policy on the tall peak, despite incurring a very high average cost as a result.

Note that this problem does not necessarily occur every time. In fully observed problems, the optimal policy is deterministic regardless of the objective that is used (with the exception of probabilistic objectives like soft optimality). The optimal deterministic policy is the same for both exponential and standard cost, so variational GPS is not in general risk seeking when the true optimal policy can be represented by the current policy class. However, this is not always guaranteed and, as shown in the evaluation at the end of this chapter, although variational GPS improves considerably on importance sampling, it can produce overly risky policies in practice.

This example suggests a general problem with maximum likelihood policy optimization. When the policy is optimized to be an M-projection onto the trajectory distribution by minimizing $D_{\mathrm{KL}}(\pi_\theta(\tau)\|q(\tau))$, as is the case in the variational GPS M-step, the policy attempts to cover the entire space of high-probability trajectories, even if this means including some low-probability trajectories in the mix. However, this could result in arbitrarily bad trajectories having a high probability. A more reasonable approach is to optimize the opposite KL-divergence: $D_{\mathrm{KL}}(q(\tau)\|\pi_\theta(\tau))$. A method that optimizes such an objective is discussed in the following section.

5.4 Policy Search via Constrained Optimization

While the variational GPS algorithm can avoid the challenges associated with importance sampling, it introduces additional problems through its use of a risk-seeking exponential cost objective. In this section, I will show how a constrained trajectory optimization approach can be used to optimize a more standard soft optimality objective, which minimizes

expected cost and maximizes entropy, while still retaining the overall structure of variational GPS, which alternates between supervised optimization of the policy and trajectory optimization. The key idea in this algorithm, termed constrained GPS, is to reformulate the soft optimality policy objective $D_{\text{KL}}(\pi_\theta(\tau)\|\rho(\tau))$ from Equation 3.4 as the following constrained optimization problem:

$$\begin{aligned} \min_{\theta, q(\tau)} \; & D_{\text{KL}}(q(\tau)\|\rho(\tau)) \\ s.t. \; & q(\mathbf{x}_1) = p(\mathbf{x}_1), \\ & q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t), \\ & D_{\text{KL}}(q(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|q(\mathbf{x}_t, \mathbf{u}_t)) = 0. \end{aligned} \tag{5.7}$$

As before, the distribution $\rho(\tau) \propto \exp(-\ell(\tau))$ is proportional to the exponential of the negative cost. As discussed in Chapter 4, this distribution can also be defined as $\rho(\tau) \propto p(\mathbf{x}_1) \prod_{t=1}^{T} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) \exp(-\ell(\mathbf{x}_t, \mathbf{u}_t))$, though the simpler $\rho(\tau) \propto \exp(-\ell(\tau))$ holds when the trajectory is parameterized entirely by its actions, with the states an uncontrolled consequence of the dynamics. The first two constraints ensure that the distribution $q(\tau)$ is consistent with the domain's initial state distribution and dynamics, and are enforced implicitly by the trajectory optimization algorithm. The last constraint ensures that the conditional action distributions $q(\mathbf{u}_t|\mathbf{x}_t)$ match the policy distribution $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. Since the action distributions fully determine the state distribution, $q(\tau)$ and the $\pi_\theta(\tau)$ become identical when the constraints are satisfied, making the constrained optimization equivalent to optimizing the policy with respect to $D_{\text{KL}}(\pi_\theta(\tau)\|\rho(\tau))$. This objective differs from the expected cost, but offers more reasonable handling of trajectory distributions, as discussed in Section 3.3. The two objectives also become equal as the magnitude of the cost is increased to infinity. In practice, a good deterministic policy can be obtained by taking the mean of the stochastic policy optimized under even a moderate cost magnitude.

The constrained GPS algorithm approximately optimizes Equation 5.7 with dual gradient descent (Boyd and Vandenberghe, 2004) and local linearization, leading to an iterative algorithm that alternates between optimizing one or more Gaussian distributions $q_i(\tau)$ with dynamic programming, and optimizing the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ to match $q(\mathbf{u}_t|\mathbf{x}_t)$. A separate

Algorithm 3 Constrained Guided Policy Search

1: Initialize the trajectories $\{q_1(\tau), \ldots, q_M(\tau)\}$
2: **for** iteration $k = 1$ to K **do**
3: Optimize each $q_i(\tau)$ with respect to $\mathcal{L}(\theta, q_i(\tau), \lambda_i)$
4: Optimize θ with respect to $\sum_{i=1}^{M} \mathcal{L}(\theta, q_i(\tau), \lambda_i)$
5: Update dual variables λ using Equation 5.8
6: **end for**
7: **return** optimized policy parameters θ

Gaussian $q_i(\tau)$ is used for each initial condition (for example when learning to control a bipedal walker that must respond to different lateral pushes), just as in the variational GPS algorithm. Because the trajectories are only coupled by the policy parameters θ, I will drop the subscript i in the derivation and consider just a single $q(\tau)$, as in the previous section.

First, consider the Lagrangian of Equation 5.7, where the initial state and dynamics constraints, which are enforced implicitly by trajectory optimization, are omitted:

$$\begin{aligned}\mathcal{L}(\theta, q, \lambda) =& D_{\mathrm{KL}}(q(\tau) \| \rho(\tau)) + \\ & \sum_{t=1}^{T} \lambda_t D_{\mathrm{KL}}(q(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| q(\mathbf{x}_t, \mathbf{u}_t)).\end{aligned}$$

Dual gradient descent maintains a vector of dual variables λ and alternates between optimizing the Lagrangian $\mathcal{L}$ with respect to $q(\tau)$ and θ, and updating the dual variables λ with subgradient descent, using a step size η:

$$\lambda_t \leftarrow \lambda_t + \eta D_{\mathrm{KL}}(q(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| q(\mathbf{x}_t, \mathbf{u}_t)). \tag{5.8}$$

The KL-divergence is estimated from samples, and the inner optimization over $q(\tau)$ and θ is performed in alternating fashion, first over each $q(\tau)$, and then over θ. Although neither the objective nor the constraints are in general convex, this approach in general tends to yield a reasonable local optimum. The full method is summarized in Algorithm 3. Each trajectory is initialized on line 1, either with unconstrained trajectory optimization, or from example demonstrations. The policy is then optimized for K iterations of dual gradient descent. In each iteration, each trajectory distribution $q_i(\tau)$ is optimized on line 3, using a few iterations

of the trajectory optimization algorithm described in the following subsection. This step can be parallelized across all trajectories. The policy is then optimized to match all of the distributions $q_i(\tau)$ on line 4, using a simple supervised learning procedure described in Section 5.4.2. This optimization is similar, but not identical, to the variational GPS M-step. Finally, the dual variables are updated according to Equation 5.8.

5.4.1 Constrained Trajectory Optimization

The trajectory optimization phase optimizes each $q_i(\tau)$ with respect to the Lagrangian $\mathcal{L}(\theta, q_i(\tau), \lambda_i)$. Since the optimization is identical for each trajectory, though it may have a different initial state and dynamics, I again drop the subscript i in this section. The optimization is carried out with dynamic programming, similarly to DDP. However, unlike in the case of importance sampled or variational GPS, the presence of the policy KL-divergence constraints necessitates a novel algorithm. One iteration of this procedure is summarized in Algorithm 4, and multiple iterations are used at each trajectory optimization step.

Each $q(\tau)$ has a mean $\hat{\tau} = (\hat{\mathbf{x}}_{1..T}, \hat{\mathbf{u}}_{1..T})$, a conditional action distribution $q(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{u}_t + \mathbf{K}\mathbf{x}_t, \mathbf{A}_t)$ at each time step (where the covariance is no longer equal to $Q^{-1}_{\mathbf{u},\mathbf{u}t}$ in general), and dynamics $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t, \mathbf{F}_t)$, which again corresponds to locally linearized Gaussian dynamics with mean $f_t(\mathbf{x}_t, \mathbf{u}_t)$ and covariance $\mathbf{F}_t$. We can again assume without loss of generality that all $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ are initially zero, and denote a perturbation from a nominal trajectory, which is updated at every iteration. Given this definition of $q(\tau)$, the objective can be written as

$$\mathcal{L}(q) = \sum_{t=1}^{T} E_{q(\mathbf{x}_t,\mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t)] - \frac{1}{2}\log|\mathbf{A}_t| + \lambda_t E_{q(\mathbf{x}_t)}[D_{\mathrm{KL}}(\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|q(\mathbf{u}_t|\mathbf{x}_t))].$$

As in the case of DDP with soft optimality, the expectations can be evaluated using the Laplace approximation, which models the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ as a (locally) linear Gaussian with mean $\mu^\pi_t(\mathbf{x}_t)$ and covariance Σ^π_t, and the cost with its first and second derivatives $\ell_{\mathbf{x}\mathbf{u}t}$ and $\ell_{\mathbf{x}\mathbf{u},\mathbf{x}\mathbf{u}t}$ (recall that subscripts denote derivatives, in this case twice with respect to both

Algorithm 4 Trajectory Optimization Iteration

Compute $f_{\mathbf{xu}t}$, $\mu^\pi_{\mathbf{x}t}$, $\ell_{\mathbf{xu}t}$, $\ell_{\mathbf{xu},\mathbf{xu}t}$ around $\hat{\tau}$
for $t = T$ to 1 **do**
  Compute $\mathbf{K}_t$ and $\mathbf{k}_t$ using Equations 5.9 and 5.10
  Compute $\mathcal{L}_{\mathbf{x}t}$ and $\mathcal{L}_{\mathbf{x},\mathbf{x}t}$ using Equation 5.11
end for
Initialize $\alpha \leftarrow 1$
repeat
  Obtain new trajectory $\hat{\tau}'$ using $\mathbf{u}_t = \alpha\mathbf{k}_t + \mathbf{K}_t\mathbf{x}_t$
  Decrease step size α
until $\mathcal{L}(\hat{\tau}', \mathbf{K}, \mathbf{A}) > \mathcal{L}(\hat{\tau}, \mathbf{K}, \mathbf{A})$
Compute $f_{\mathbf{xu}t}$, $\mu^\pi_{\mathbf{x}t}$, $\ell_{\mathbf{xu}t}$, $\ell_{\mathbf{xu},\mathbf{xu}t}$ around $\hat{\tau}'$
repeat
  Compute $\mathbf{S}_t$ using current $\mathbf{A}_t$ and $\mathbf{K}_t$
  for $t = T$ to 1 **do**
    repeat
      Compute $\mathbf{K}_t$ using Equation 5.13
      Compute $\mathbf{A}_t$ by solving CARE in Equation 5.14
    until $\mathbf{A}_t$ and $\mathbf{K}_t$ converge (about 5 iterations)
    Compute $\mathcal{L}_{\mathbf{S}t}$ using Equation 5.12
  end for
until all $\mathbf{S}_t$ and $\mathbf{A}_t$ converge (about 2 iterations)
return new mean $\hat{\tau}'$ and covariance terms $\mathbf{A}_t$, $\mathbf{K}_t$

the state and action):

$$\begin{aligned}\mathcal{L}(q) \approx \sum_{t=1}^{T} \frac{1}{2}\begin{bmatrix}\hat{\mathbf{x}}_t \\ \hat{\mathbf{u}}_t\end{bmatrix}^{\mathrm{T}} \ell_{\mathbf{xu},\mathbf{xu}t}\begin{bmatrix}\hat{\mathbf{x}}_t \\ \hat{\mathbf{u}}_t\end{bmatrix} + \begin{bmatrix}\hat{\mathbf{x}}_t \\ \hat{\mathbf{u}}_t\end{bmatrix}^{\mathrm{T}} \ell_{\mathbf{xu}t} + \frac{1}{2}\mathrm{tr}\left(\Sigma_t \ell_{\mathbf{xu},\mathbf{xu}t}\right) - \frac{1}{2}\log|\mathbf{A}_t| + \\ \frac{\lambda_t}{2}\log|\mathbf{A}_t| + \frac{\lambda_t}{2}(\hat{\mathbf{u}}_t - \mu^\pi_t(\hat{\mathbf{x}}_t))^{\mathrm{T}}\mathbf{A}_t^{-1}(\hat{\mathbf{u}}_t - \mu^\pi_t(\hat{\mathbf{x}}_t)) + \frac{\lambda_t}{2}\mathrm{tr}\left(\mathbf{A}_t^{-1}\Sigma^\pi_t\right) + \\ \frac{\lambda_t}{2}\mathrm{tr}\left(\mathbf{S}_t\left(\mathbf{K}_t - \mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t)\right)^{\mathrm{T}}\mathbf{A}_t^{-1}\left(\mathbf{K}_t - \mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t)\right)\right),\end{aligned}$$

where constants are omitted, $\mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t)$ is the gradient of the policy mean at $\hat{\mathbf{x}}_t$, and Σ_t denotes the joint marginal covariance over states and actions in $q(\mathbf{x}_t, \mathbf{u}_t)$. $\mathbf{S}_t$ will be used to refer to the covariance of $q(\mathbf{x}_t)$ and $\mathbf{A}_t$ to refer to the *conditional* covariance of $q(\mathbf{u}_t|\mathbf{x}_t)$.

Each iteration of trajectory optimization first forms this approximate Lagrangian by

computing the derivatives of the dynamics and cost function on line 1. A dynamic programming algorithm then computes the gradients and Hessians with respect to the mean state and action at each time step (keeping $\mathbf{A}_t$ fixed), as summarized on lines 2-5, which makes it possible to take a Newton-like step by multiplying the gradient by the inverse Hessian, analogously to DDP. The action then becomes a linear function of the corresponding state, resulting in linear feedback terms $\mathbf{K}_t$. After taking this Newton-like step, a line search is performed to ensure improvement on lines 7-10, and then $\mathbf{A}_t$ is updated at each time step on lines 12-21. To derive the gradients and Hessians, it will be convenient to define the following quantities, which incorporate information about future time steps analogously to the Q-function in DDP:

$$
\begin{aligned}
Q_{\mathbf{xu}t} &= \ell_{\mathbf{xu}t} + f_{\mathbf{xu}}^{\mathrm{T}} \mathcal{L}_{\mathbf{x}t+1} \\
Q_{\mathbf{xu},\mathbf{xu}t} &= \ell_{\mathbf{xu},\mathbf{xu}t} + f_{\mathbf{xu}}^{\mathrm{T}} \mathcal{L}_{\mathbf{x},\mathbf{x}t+1} f_{\mathbf{xu}},
\end{aligned}
$$

where the double subscripts $\mathbf{xu}$ again denote derivatives with respect to $(\mathbf{x}_t, \mathbf{u}_t)^{\mathrm{T}}$, and $\mathcal{L}_{\mathbf{x}t+1}$ and $\mathcal{L}_{\mathbf{x},\mathbf{x}t+1}$ are the gradient and Hessian of the objective with respect to $\hat{\mathbf{x}}_{t+1}$. As with the iLQG variant of DDP, this algorithm assumes locally linear dynamics and drops the higher order dynamics derivatives. Proceeding recursively backwards through time, the first and second derivatives with respect to $\hat{\mathbf{u}}_t$ are then given by

$$
\begin{aligned}
\mathcal{L}_{\mathbf{u}t} &= Q_{\mathbf{u},\mathbf{u}t}\hat{\mathbf{u}}_t + Q_{\mathbf{u},\mathbf{x}t}\hat{\mathbf{x}}_t + Q_{\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1}(\hat{\mathbf{u}}_t - \mu_t^{\pi}(\hat{\mathbf{x}}_t)) \\
\mathcal{L}_{\mathbf{u},\mathbf{u}t} &= Q_{\mathbf{u},\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1},
\end{aligned}
$$

where the application of the chain rule to include the effect of $\hat{\mathbf{u}}_t$ on subsequent time steps is subsumed inside $Q_{\mathbf{u}t}$, $Q_{\mathbf{u},\mathbf{u}t}$, and $Q_{\mathbf{u},\mathbf{x}t}$. Noting that $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ are zero, we can solve for the optimal change to the action $\mathbf{k}_t$:

$$
\mathbf{k}_t = -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1}\right)^{-1}\left(Q_{\mathbf{u}t} - \lambda_t \mathbf{A}_t^{-1}\mu_t^{\pi}(\hat{\mathbf{x}}_t)\right). \tag{5.9}
$$

The feedback $\mathbf{K}_t$ is the derivative of $\hat{\mathbf{u}}_t$ with respect to $\hat{\mathbf{x}}_t$:

$$
\mathbf{K}_t = -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1}\right)^{-1}\left(Q_{\mathbf{u},\mathbf{x}t} - \lambda_t \mathbf{A}_t^{-1}\mu_{\mathbf{x}t}^{\pi}(\hat{\mathbf{x}}_t)\right). \tag{5.10}
$$

To complete the dynamic programming step, we can now differentiate the objective with respect to $\hat{\mathbf{x}}_t$, treating $\hat{\mathbf{u}}_t = \mathbf{k}_t + \mathbf{K}_t\hat{\mathbf{x}}_t$ as a function of $\hat{\mathbf{x}}_t$:

$$\begin{aligned}
\mathcal{L}_{\mathbf{x}t} &= Q_{\mathbf{x},\mathbf{x}t}\hat{\mathbf{x}}_t + Q_{\mathbf{x},\mathbf{u}t}(\mathbf{k}_t + \mathbf{K}_t\hat{\mathbf{x}}_t) + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{x}t}\hat{\mathbf{x}}_t + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{u}t}(\mathbf{K}_t\hat{\mathbf{x}}_t + \mathbf{k}_t) + Q_{\mathbf{x}t} + \\
&\quad \lambda_t(\mathbf{K}_t - \mu^{\pi}_{\mathbf{x}t}(\hat{\mathbf{x}}_t))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{K}_t\hat{\mathbf{x}}_t + \mathbf{k}_t - \mu^{\pi}_t(\hat{\mathbf{x}}_t)) \\
&= Q_{\mathbf{x},\mathbf{u}t}\mathbf{k}_t + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{u}t}\mathbf{k}_t + Q_{\mathbf{x}t} + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u}t} + \\
&\quad \lambda_t(\mathbf{K}_t - \mu^{\pi}_{\mathbf{x}t}(\hat{\mathbf{x}}_t))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{k}_t - \mu^{\pi}_t(\hat{\mathbf{x}}_t)) \\
\mathcal{L}_{\mathbf{x},\mathbf{x}t} &= Q_{\mathbf{x},\mathbf{x}t} + Q_{\mathbf{x},\mathbf{u}t}\mathbf{K}_t + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{x}t} + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{u}t}\mathbf{K}_t + \\
&\quad \lambda_t(\mathbf{K}_t - \mu^{\pi}_{\mathbf{x}t}(\hat{\mathbf{x}}_t))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{K}_t - \mu^{\pi}_{\mathbf{x}t}(\hat{\mathbf{x}}_t)),
\end{aligned} \tag{5.11}$$

where the simplification again happens because $\hat{\mathbf{x}}_t$ is zero.

Once $\mathbf{k}_t$ and $\mathbf{K}_t$ are computed at each time step, a simulator rollout can be performed using the deterministic policy $\mathbf{u}_t = \mathbf{k}_t + \mathbf{K}_t\mathbf{x}_t$ to obtain a new mean nominal trajectory, just as in standard DDP. Since the dynamics may deviate from the previous linearization far from the previous trajectory, a line search is performed on $\mathbf{k}_t$ to ensure that the objective improves as a function of the new mean, as summarized on lines 7-10.

Once the new mean is found, both the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ and the dynamics are relinearized around the new nominal trajectory on line 13, and a second dynamic programming pass is performed to update the covariance $\mathbf{A}_t$ and feedback terms $\mathbf{K}_t$. As before, it is convenient to introduce a variable that incorporates gradient information from future time steps:

$$Q_{\mathbf{xu},\mathbf{xu}t} = \ell_{\mathbf{xu},\mathbf{xu}t} + 2f_{\mathbf{xu}t}^{\mathrm{T}}\mathcal{L}_{\mathbf{S}t+1}f_{\mathbf{xu}t}.$$

This equation emerges from the fact that $\mathbf{S}_{t+1} = f_{\mathbf{xu}t}^{\mathrm{T}}\Sigma_t f_{\mathbf{xu}t}$ (due to the form of $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$), so that

$$\frac{1}{2}\mathrm{tr}\left(\Sigma_t\ell_{\mathbf{xu},\mathbf{xu}t}\right) + \frac{\partial\mathbf{S}_{t+1}}{\partial\Sigma_t}\cdot\mathcal{L}_{\mathbf{S}t+1} = \frac{1}{2}\mathrm{tr}\left(\Sigma_t Q_{\mathbf{xu},\mathbf{xu}t}\right).$$

This allows $Q_{\mathbf{xu},\mathbf{xu}t}$ to simply be substituted for $\ell_{\mathbf{xu},\mathbf{xu}t}$ in the objective to include all the effect of the covariance on future time steps. To then derive the gradient of the objective, first with respect to $\mathbf{A}_t$ and $\mathbf{K}_t$ for optimization, and then with respect to $\mathbf{S}_t$ to complete

the dynamic programming step, first note that

$$\Sigma_t = \begin{bmatrix} \mathbf{S}_t & \mathbf{S}_t \mathbf{K}_t^{\mathrm{T}} \\ \mathbf{K}_t \mathbf{S}_t & \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^{\mathrm{T}} + \mathbf{A}_t \end{bmatrix}.$$

This allows the term $\mathrm{tr}\left(\Sigma_t Q_{\mathbf{xu},\mathbf{xu}t}\right)$ to be expanded to get

$$\begin{aligned} &\mathrm{tr}\left(\Sigma_t Q_{\mathbf{xu},\mathbf{xu}t}\right) = \mathrm{tr}\left(\mathbf{S}_t Q_{\mathbf{x},\mathbf{x}t}\right) + \mathrm{tr}\left(\mathbf{S}_t \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{x}t}\right) + \\ &\mathrm{tr}\left(\mathbf{S}_t Q_{\mathbf{x},\mathbf{u}t} \mathbf{K}_t\right) + \mathrm{tr}\left(\mathbf{A}_t Q_{\mathbf{u},\mathbf{u}t}\right) + \mathrm{tr}\left(\mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}\right). \end{aligned}$$

Using this identity, we can obtain the derivatives of the objective with respect to $\mathbf{K}_t$, $\mathbf{A}_t$, and $\mathbf{S}_t$:

$$\begin{aligned} \mathcal{L}_{\mathbf{A}t} &= \frac{1}{2} Q_{\mathbf{u},\mathbf{u}t} + \frac{\lambda_t - 1}{2} \mathbf{A}_t^{-1} - \frac{\lambda_t}{2} \mathbf{A}_t^{-1} \mathbf{M} \mathbf{A}_t^{-1} \\ \mathcal{L}_{\mathbf{K}t} &= Q_{\mathbf{u},\mathbf{u}t} \mathbf{K}_t \mathbf{S}_t + Q_{\mathbf{u},\mathbf{x}t} \mathbf{S}_t + \lambda_t \mathbf{A}_t^{-1} \mathbf{K}_t \mathbf{S}_t - \lambda_t \mathbf{A}_t^{-1} \mu_{\mathbf{x}t}^{\pi}(\hat{\mathbf{x}}_t) \mathbf{S}_t \\ \mathcal{L}_{\mathbf{S}t} &= \frac{1}{2} \left[Q_{\mathbf{x},\mathbf{x}t} + \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{x}t} + Q_{\mathbf{x},\mathbf{u}t} \mathbf{K}_t + \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t} \mathbf{K}_t \right. \\ &\quad \left. + \lambda_t (\mathbf{K}_t - \mu_{\mathbf{x}t}^{\pi}(\hat{\mathbf{x}}_t))^{\mathrm{T}} \mathbf{A}_t^{-1} (\mathbf{K}_t - \mu_{\mathbf{x}t}^{\pi}(\hat{\mathbf{x}}_t)) \right], \end{aligned} \tag{5.12}$$

where $\mathbf{M} = \Sigma_t^{\pi} + (\hat{\mathbf{u}}_t - \mu_t^{\pi}(\hat{\mathbf{x}}_t))(\hat{\mathbf{u}}_t - \mu_t^{\pi}(\hat{\mathbf{x}}_t))^{\mathrm{T}} + (\mathbf{K}_t - \mu_{\mathbf{x}t}^{\pi}(\hat{\mathbf{x}}_t)) \mathbf{S}_t (\mathbf{K}_t - \mu_{\mathbf{x}t}^{\pi}(\hat{\mathbf{x}}_t))^{\mathrm{T}}$. The equation for $\mathcal{L}_{\mathbf{S}t}$ is simply half of $\mathcal{L}_{\mathbf{x},\mathbf{x}t}$, which indicates that $Q_{\mathbf{xu},\mathbf{xu}t}$ is the same as during the first backward pass. Solving for $\mathbf{K}_t$, we also obtain the same equation as before:

$$\mathbf{K}_t = -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1}\right)^{-1} \left(Q_{\mathbf{u},\mathbf{x}t} - \lambda_t \mathbf{A}_t^{-1} \mu_{\mathbf{x}t}^{\pi}(\hat{\mathbf{x}}_t)\right). \tag{5.13}$$

To solve for $\mathbf{A}_t$, we set the derivative to zero and multiply both sides on both the left and right by $\sqrt{2}\mathbf{A}_t$ to get

$$\mathbf{A}_t Q_{\mathbf{u},\mathbf{u}t} \mathbf{A}_t + (\lambda_t - 1) \mathbf{A}_t - \lambda_t \mathbf{M} = 0. \tag{5.14}$$

The above equation is a continuous-time algebraic Riccati equation (CARE), and can be solved by any standard CARE solver.[2] Solving the CARE is a well studied problem, with

[2] Although such equations often come up in the context of optimal control, our use of algebraic Riccati equations is unrelated to the manner in which they are usually employed.

running times comparable to eigenvalue decomposition (Arnold and Laub, 1984). The implementation used in the experiments in this chapter makes use of the CARE solver built into the MATLAB package.

Since $\mathbf{K}_t$ depends on $\mathbf{A}_t$, which itself depends on both $\mathbf{K}_t$ and $\mathbf{S}_t$, it is necessary to use the old values of each quantity, and repeat the solver for several iterations. On lines 15-18, $\mathbf{K}_t$ and $\mathbf{A}_t$ are repeatedly computed at each time step, which usually results in convergence within a few iterations. On lines 12-21, the entire backward pass is repeated several times to update $\mathbf{S}_t$ based on the new $\mathbf{A}_t$, which converges even faster. In practice, just two backward passes are usually sufficient.

This derivation allows $q(\tau)$ to be optimized under a Laplace approximation. Although the Laplace approximation provides a reasonable objective, the linearized policy may not reflect the real structure of $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ in the entire region where $q(\mathbf{x}_t)$ is high. Since the policy is trained by sampling states from $q(\mathbf{x}_t)$, it optimizes a different objective. This can lead to divergence when the policy is highly nonlinear. To solve this problem, we can estimate the policy terms with M random samples $\mathbf{x}_{ti}$ drawn from $q(\mathbf{x}_t)$, rather than by linearizing around the mean, just as was done in the variational GPS E-step. The resulting optimization algorithm has a similar structure, with $\mu_t^\pi(\hat{\mathbf{x}}_t)$ and $\mu_{\mathbf{x}t}^\pi(\hat{\mathbf{x}}_t)$ in the derivatives of the mean replaced by their averages over the samples. The gradients of the covariance terms become more complex, but in practice, simply substituting the sample averages of μ_t^π and $\mu_{\mathbf{x}t}^\pi$ into the above algorithm works almost as well, and is somewhat faster. A complete derivation of the true gradients is provided in Appendix C for completeness.

5.4.2 Policy Optimization

Once $q(\tau)$ is fixed, the policy optimization becomes a weighted supervised learning problem. Training points $\mathbf{x}_{ti}$ are sampled from $q(\mathbf{x}_t)$ at each time step, and the policy is trained to be an I-projection onto the corresponding conditional Gaussian. When the policy is conditionally Gaussian, with the mean $\mu^\pi(\mathbf{x}_t)$ and covariance $\Sigma^\pi(\mathbf{x}_t)$ being any function of $\mathbf{x}_t$,

the policy objective is given by

$$\begin{aligned}\mathcal{L}(\theta) &= \sum_{t=1}^{T} \lambda_t \sum_{i=1}^{N} D_{\mathrm{KL}}(\pi_\theta(\mathbf{u}_t|\mathbf{x}_{ti}) \| q(\mathbf{u}_t|\mathbf{x}_{ti})) \\ &= \sum_{t=1}^{T} \lambda_t \sum_{i=1}^{N} \frac{1}{2} \left\{ \mathrm{tr}(\Sigma_t^\pi(\mathbf{x}_{ti}) \mathbf{A}_t^{-1}) - \log |\Sigma^\pi(\mathbf{x}_{ti})| + \right. \\ &\quad \left. (\mathbf{K}_t \mathbf{x}_{ti} + \mathbf{k}_t - \mu^\pi(\mathbf{x}_{ti}))^{\mathrm{T}} \mathbf{A}_t^{-1} (\mathbf{K}_t \mathbf{x}_{ti} + \mathbf{k}_t - \mu^\pi(\mathbf{x}_{ti})) \right\}.\end{aligned}$$

This is a least-squares optimization on the policy mean, with targets $\mathbf{K}_t \mathbf{x}_{ti} + \mathbf{k}$ and weight matrix $\mathbf{A}_t^{-1}$, and can be performed by standard algorithms such as stochastic gradient descent (SGD) or, as in our implementation, LBFGS.

Note that as the constraints are satisfied and $q(\tau)$ approaches $\pi_\theta(\tau)$, $q(\mathbf{u}_t|\mathbf{x}_t)$ becomes proportional to the exponential of the Q-function of $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ under the soft optimality objective. Minimizing $D_{\mathrm{KL}}(\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| q(\mathbf{u}_t|\mathbf{x}_t))$ therefore resembles policy iteration with an "optimistic" approximate Q-function. This is a distinct advantage over the opposite KL-divergence (where $q(\tau)$ is an I-projection of $\pi_\theta(\tau)$) used in the variational GPS algorithm, which causes the policy to be risk-seeking by optimizing the expected exponential negative cost. In the next section, we will see that this objective allows constrained GPS to outperform variational GPS on more challenging tasks.

5.5 Experimental Evaluation

In this section, I will present experimental evaluations of the three guided policy search algorithms. All of the experiments focus on locomotion-like tasks, which are particularly challenging due to their high dimensionality and underactuation. I will show that the proposed methods outperform previous policy search techniques, compare and contrast the three algorithms, and show results on very challenging humanoid locomotion and push recovery tasks.

The dynamical systems considered in this section include planar swimming, hopping, and walking, as well as 3D humanoid running. Each task was simulated with the MuJoCo simulator (Todorov et al., 2012), using systems of rigid links with noisy motors at the joints.

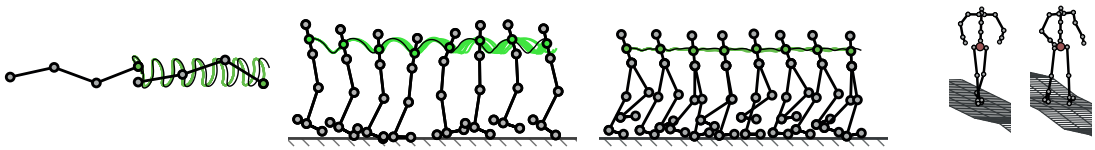


Figure 5.3: Left to right, illustrations of the swimmer, monopod hopper, bipedal walker, and 3D humanoid model. The lines indicate center of mass trajectories for the initial example (black) and a policy learned with importance-sampled guided policy search (green).

The cost function for each task consisted of a sum of three terms:

$$\ell(\mathbf{x}_t, \mathbf{u}_t) = w_{\mathbf{u}} \|\mathbf{u}_t\|^2 + w_v (v_x - v_x^\star)^2 + w_h (p_y - p_y^\star)^2,$$

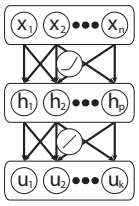
where v_x and $v_x^\star$ are the current and desired horizontal velocities, p_y and $p_y^\star$ are the current and desired heights of the root link, and $w_{\mathbf{u}}$, w_v, and w_h determine the weight on each objective term. The initial trajectories for the swimmer and hopper were generated using DDP, the initial trajectory for the bipedal planar walker was initialized from a demonstration obtained using a previous hand-crafted locomotion system (Yin et al., 2007), and the example demonstration for humanoid running was obtained from motion capture of a human subject. For perturbed tasks, such as traversal of sloped and even terrain or push recovery, the initial perturbation responses were generated with trajectory optimization, initialized from an unperturbed execution of the task. Illustrations of each of the dynamical systems are provided in Figure 5.3, and the particular details of each system are discussed below.

Swimmer: The swimmer is a 3-link snake, with 10 state dimensions for the position and angle of the head, the joint angles, and the corresponding velocities, as well as 2 action dimensions for the torques. The surrounding fluid applies a drag on each link, allowing the snake to propel itself. The simulation step is 0.05s, the cost weights are $w_{\mathbf{u}} = 0.0001$, $w_v = 1$, and $w_h = 0$, and the desired velocity is $v_x^\star = 1$m/s.

Hopper: The hopper has 4 links: torso, upper leg, lower leg, and foot. The state has 12 dimensions, and the actions have 3. The cost weights are $w_{\mathbf{u}} = 0.001$, $w_v = 1$, and $w_h = 10$, and the desired velocity and height are $v_x^\star = 1.5$m/s and $p_y^\star = 1.5$m. A lower time step of 0.02s was used to handle contacts.

Walker: The walker has 7 links, corresponding to two legs and a torso, 18 state dimensions and 6 torques. The cost weights are $w_{\mathbf{u}} = 0.0001$, $w_v = 1$, and $w_h = 10$, and the desired velocity and height are $v_x^\star = 2.1$m/s and $p_y^\star = 1.1$m. The time step is 0.01s.

3D Humanoid: The humanoid consists of 13 links, with a free-floating 6 DoF base, 4 ball joints, 3 joints with 2 DoF, and 5 hinge joints, for a total of 29 degrees of freedom. Ball joints are represented by quaternions, while their velocities are represented by 3D vectors, so the entire model has 63 dimensions. The cost weights are are $w_{\mathbf{u}} = 0.00001$, $w_v = 1$, and $w_h = 10$, and the desired velocity and height are $v_x^\star = 2.5$m/s and $p_y^\star = 0.9$m. The time step is 0.01s.



For all tasks, the learned control policies consisted of neural networks with one hidden layer, with a soft rectifier $a = \log(1 + \exp(z))$ at the first layer and linear connections to the output layer. Gaussian noise with a diagonal covariance was added to the output to create a stochastic policy. A schematic of the controller architecture is shown on the right. When evaluating the cost of a policy, the noise was removed, yielding a deterministic controller. While this class of policies is very expressive, it poses a considerable challenge for policy search methods, due to its nonlinearity and high dimensionality.

The input to the neural network controller consisted of the current state of the system, made up of joint angles, joint velocities, root velocities, and, except for the swimmer, the height of the root above the ground. When specified, the input was also augmented by a set of forward kinematics features, which provided the Cartesian position of each joint center with respect to the root, as well as indicator features for contacts. By including or excluding these features, the difficulty of the task could be altered. However, even for tasks that involved traversal of uneven terrain, the slope of the terrain was not included as an observation, requiring the controller to accomplish each task "blind."

5.5.1 Experiments with Importance Sampling

The first set of experiments was intended to evaluate the performance of the importance-sampled variant of guided policy search in comparison to a variety of previous approaches.

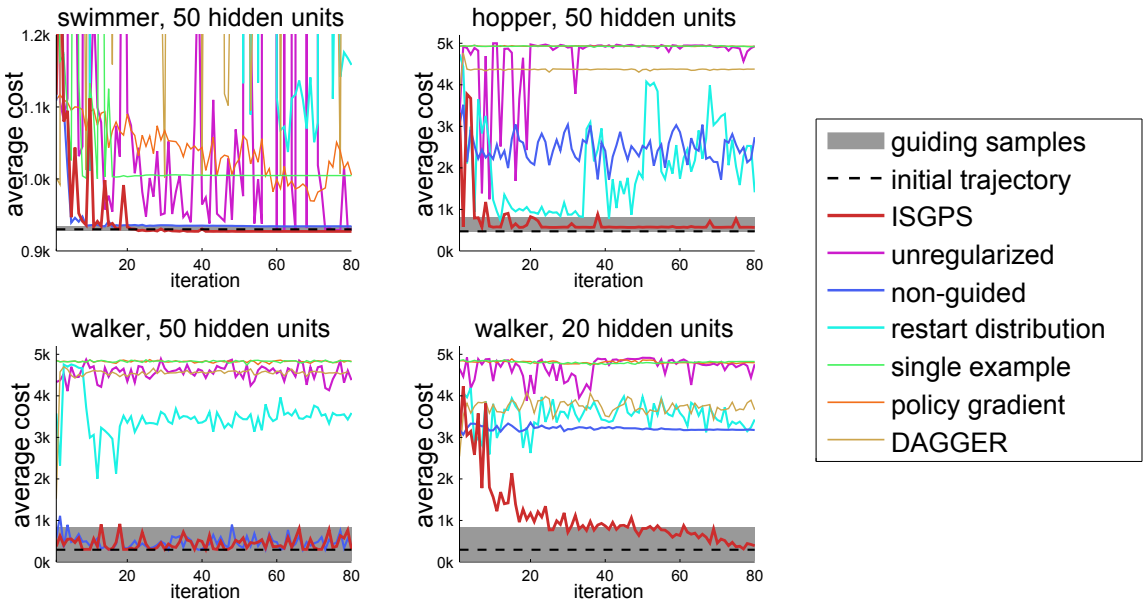


Figure 5.4: Comparison of importance-sampled guided policy search (ISGPS) with ablated variants and prior methods. ISGPS successfully learned each gait, while methods that do not use guiding samples or regularization failed to make progress. All methods used 10 rollouts per iteration. Guided variants (ISGPS, unregularized, restart, DAGGER) also used 40 guiding samples.

These experiments involved learning simple locomotion gaits for the planar swimmer, hopper, and walker, with a horizon of 500 time steps. The policy input in these experiments consisted only of the system state, and did not include the forward kinematics or contact features. Although learning a simple cyclic gait is sufficient to succeed on each task, simulator complexity and motor noise (up to 10% of the average torque on each joint) necessitate an intelligent feedback control that can react to small perturbations. Combined with challenging features like high dimensionality, underactuation, and the nonlinearity of the controller, these tasks are in fact extremely challenging for standard reinforcement learning and imitiation learning methods.

The results are presented in Figure 5.4. The evaluated methods include several ablated variants of importance-sampled guided policy search. The first variant, labeled "unregularized," omits the regularization term in the IS objective, to ascertain whether standard importance sampling could be used. The second, referred to as "non-guided," does not use the guiding samples during the policy search, but does include them in the supervised

initialization. The third variant also does not use the guiding samples in the policy objective, but uses the guiding distributions as "restart distributions" to specify new initial state distributions that cover states we expect to visit under a good policy, analogously to prior work (Kakade and Langford, 2002; Bagnell et al., 2003). This comparison was meant to check that the guiding samples actually aided policy learning, rather than simply focusing the policy search on "good" states.

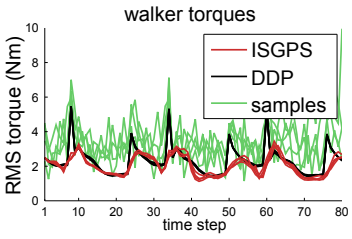
The first prior method, referred to as "single example," initializes the policy using the single initial example, as in prior work on learning from demonstration, rather than by using guiding samples, and then improves the policy with importance sampling but no guiding samples, again as in prior work (Peshkin and Shelton, 2002). The second method uses standard policy gradients, with a PGT or GPOMDP-type estimator (Sutton et al., 1999; Baxter et al., 2001). The third method was DAGGER, an imitation learning algorithm that aims to find a policy that matches the "expert" DDP actions (Ross et al., 2011).

The results in Figure 5.4 are shown in terms of the average cost of 10 samples at each iteration, along with the value of the initial example and a shaded interval indicating two standard deviations of the guiding sample costs. Importance-sampled GPS learned each gait, matching the reward of the initial example. The comparison to the unregularized variant shows that the regularization term is crucial for obtaining a good policy. The non-guided variant, which only used the guiding samples for pretraining, sometimes found a good solution because even a partially successful initial policy may succeed on one of its initial samples, which then serves the same function as the guiding samples by indicating high-reward regions. However, a successful non-guided outcome hinged entirely on the initial policy. This is illustrated by the fourth graph in Figure 5.4, which shows a walking policy with 20 hidden units that is less successful initially. The full algorithm was still able to improve using the guiding samples, while the non-guided variant did not make progress. The restart distribution variant performed poorly. Wider initial state distributions greatly increased the variance of the estimator, and because the guiding distributions were only used to sample states, their ability to also point out good actions was not leveraged.

Standard policy gradient and "single example" learning from demonstration methods failed to learn any of the gaits. This suggests that guiding distribution is crucial for learning such complex behaviors, and initialization from a single example is insufficient. This

insight is also used in the variational and constrained GPS variants, which sample from the guiding distribution during policy optimization, rather than just using the single optimized trajectory. DAGGER also performed poorly, since it assumed that the expert could provide optimal actions in all states, while the DDP policy was actually only valid close to the example. DAGGER therefore wasted a lot of effort on states where the DDP policy was not valid, such as after the hopper or walker fell. In the following sections, I will also present comparisons to an improved variant of DAGGER that weights the samples by their probability under the guiding distribution. This variant somewhat mitigates this problem, but is still unable to learn more complex tasks.

These results show that even simple locomotion on flat ground with a general-purpose neural network control poses a serious challenge to more standard reinforcement learning algorithms, even other importance-sampled methods. In fact, to the best of my knowledge, no prior method has successfully demonstrated bipedal locomotion under general learned neural-network control, without relying on hand-engineered feedbacks or carefully crafted features.



Interestingly, the learned neural network policies often used less torque than the initial DDP solution. The plot on the right shows the torque magnitudes for the walking policy, the deterministic DDP policy around the example, and the guiding samples. The smoother, more compliant behavior of the learned policy can be explained in part by the fact that the neural network can produce more subtle corrections than simple linear feedback.

The initial trajectories for the simple locomotion tasks could all be reproduced by neural networks with sufficient training, making adaptive guiding distributions unnecessary. To test the adaptive variant of importance-sampled GPS, a different walking example was constructed that switched to a "prancing" gait after 2.5s, making the initial guiding samples difficult to recreate with a stationary policy, since the gait switch depended only on time, not on state. As discussed in Section 5.2.2, adapting the guiding distribution to the policy can produce more suitable samples in such cases.

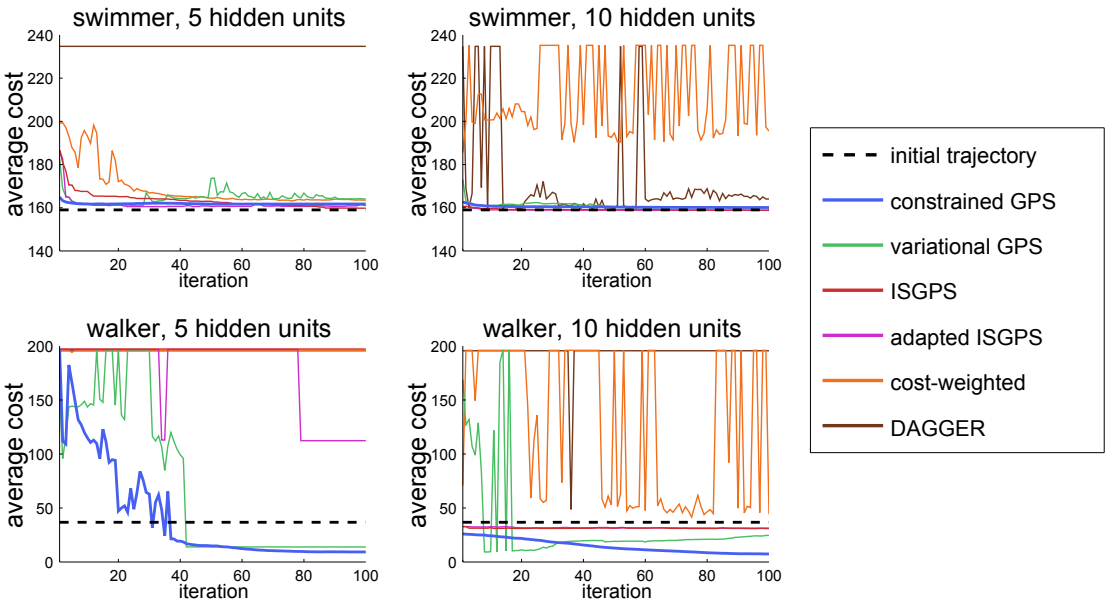
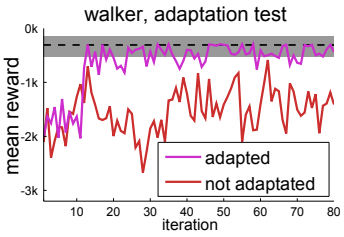


Figure 5.5: Comparisons on locomotion tasks with varying numbers of hidden units. Policies that fail and become unstable are off the scale, and are clamped to the maximum cost. Frequent vertical oscillations indicate a policy that oscillates between stable and unstable solutions.



The plot on the right shows the results with and without adaptive samples. With adaptation, importance-sampled GPS quickly learned a successful gait, while without it, it made little progress.[3] However, as we will see in the following sections, adaptative importance-sampled GPS is still limited by the local optima of the IS objective, and on more complex tasks, these local optima put it at a disadvantage against the inherently adaptive variational and constrained GPS algorithms.

5.5.2 Comparisons Between Algorithms

The second set of experiments used the same simple locomotion domains, but was aimed at identifying the relative strengths and weaknesses of the various guided policy search methods. These experiments involved 500-step planar swimming and walking tasks, without kinematic or contact features, and with varying numbers of hidden units. The results are presented in Figure 5.5. The evaluated methods include all three GPS algorithms, including

[3] Note that the adapted variant used 20 samples per iteration: 10 on-policy samples and 10 adaptive guiding samples.

both adapted and non-adapted variants of ISGPS, as well as two prior methods.

The first prior method, labeled "cost-weighted," is a reinforcement learning algorithm based on PoWER, a recently proposed variational method for policy search (Kober and Peters, 2009). Although PoWER requires a linear policy parameterization and a specific exploration strategy, we can construct an analogous non-linear algorithm by replacing the analytic M-step with nonlinear optimization, as in the M-step of variational GPS. This method provides a good point of comparison with variational GPS, since it employs the same variational decomposition of the policy objective and the same M-step, but wheras variational GPS uses trajectory optimization for the E-step, PoWER constructs the variational distribution $q(\tau)$ by taking samples from the current policy and reweighting them by the exponential of their cost.

The second prior method is a variant of DAGGER that weights each sample by its probability under the initial trajectory distribution, mitigating a problem with standard DAGGER that causes the algorithm to attempt to match the DDP solution in states that are far from the initial trajectory (and therefore do not have a near-optimal DDP action). Another variant of DAGGER that also adapts the DDP solution analogously to variational GPS was also tested, but found to produce inferior results in all cases.

ISGPS was able to solve both swimming tasks, but failed to learn a walking gait with 5 hidden units. A policy with 5 units is too restrictive to reproduce the initial gait, and due to its lack of adaptation, ISGPS could not find parameter settings that gave a high importance weight to the guiding samples, and therefore could not learn a good policy. The adapted variant of ISGPS can in principle rectify this problem, and achieved a lower average cost, but became trapped in local optima and could not learn a successful gait. On the other hand, the variational and constrained GPS algorithms could learn effective controllers for both tasks. Recall that the main limitation of variational GPS, caused by its use of the expected exponential cost objective, only becomes apparent when the optimal deterministic policy cannot be represented by the current policy class. On these simpler tasks, this was not an issue, and the final variational GPS policy was always nearly deterministic. In the next section, I will discuss more challenging tasks that illustrate the difference between variational and constrained GPS in more detail.

In the comparison to the PoWER-like "cost-weighted" algorithm, the prior method

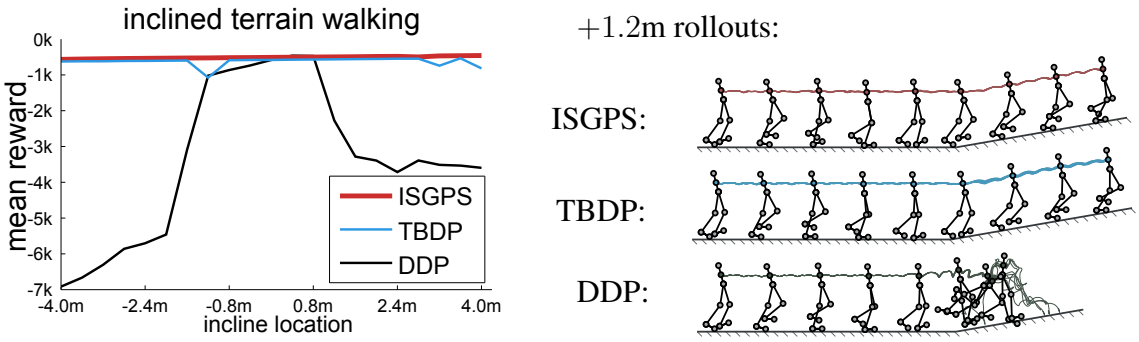


Figure 5.6: Comparison of ISGPS, TBDP, and DDP at varying incline locations (left) and plots of their rollouts (right). ISGPS and TBDP generalized to all incline locations.

was provided with 1000 samples at each iteration, to make up for the fact that it does not otherwise take advantage of the known dynamics model. However, in spite of such a large number of available samples, this approach could only learn an effective gait for the swimmer with five hidden units, since the search space for the walker is far too large to handle with random, trial-and-error based exploration. Lastly, the weighted version of DAGGER succeeded in learning controllers for both swimming tasks, but could not learn an effective policy for the walker. In general, the performance of this approach was highly sensitive to the particular weighting scheme used. In this experiment, the weights were proportional to the probability of the samples under the trajectory distribution built around the example, raised to some power. This power had to be adjusted carefully to obtain good results.

5.5.3 Uneven Terrain Locomotion

While the tasks in the previous section provide a point of comparison against prior methods, they do not illustrate the importance of generalization, which is a key advantage of parametric policies. In this section, I will present experiments aimed at assessing the degree to which the learned policies can generalize a bipedal walking gait learned on uneven terrain to other terrains.

In the first terrain experiment, importance-sampled guided policy was used to train a controller with 100 hidden units that walks 4m on flat ground, and then climbs a 10° degree incline. The policy was provided with forward kinematics and contact features, but

could not perceive the slope of the ground, requiring it to learn a gait that was robust to inclines. To test generalization, the start of the incline was then moved forward and backward, and both the learned policy and the initial DDP solution were compared. In addition, a simplified trajectory-based dynamic programming (TBDP) approach was included in the comparison. This method aggregates local DDP policies and uses the policy of the nearest neighbor to the current state (Atkeson and Stephens, 2008). Prior TBDP methods add new trajectories until the TBDP policy achieves good results. Since the initial DDP policy already succeeds on the training environment, only this initial policy was used. Since no attempt was made to construct a global value function, this approach can be regarded as the simplest possible TBDP-style approach. Both ISGPS and TBDP therefore used the same DDP solution: TBDP used it to build a nonparametric policy, and ISGPS used it to generate guiding samples. As shown in Figure 5.6, both ISGPS and TBDP generalized to all slope positions, while the DDP policy, which expected the incline to start at a specific time, could only climb it in a small interval around its original location. This illustrates the importance of stationary policies for achieving good generalization in novel environments, but does not demonstrate the value of training parametric control, since TBDP attained comparable results.

To test whether parametric controllers achieve superior generalization, the next experiment consisted of traversing terrain consisting of 1m and 2m segments with varying slope (up to 10°), again with kinematic and contact features and 100 hidden units. The policy was trained on one training terrain, and tested on four test terrains. The results in Figure 5.7 show that ISGPS generalized to the new terrains, while the nearest-neighbor TBDP policy did not, with all rollouts eventually failing on each test terrain. Unlike TBDP, the trained neural network learned generalizable rules for balancing on sloped terrain. While these rules might not generalize to much steeper inclines without additional training, they indicate a degree of generalization significantly greater than nearest-neighbor lookup. Example rollouts from each policy can be viewed on the project website: `http://graphics.stanford.edu/projects/gpspaper/index.htm`.

The third terrain experiment was intended to compare the various guided policy search methods and evaluate the importance of the kinematic and contact features. In this test, the guided policy search methods were trained on either one or three training terrains, with

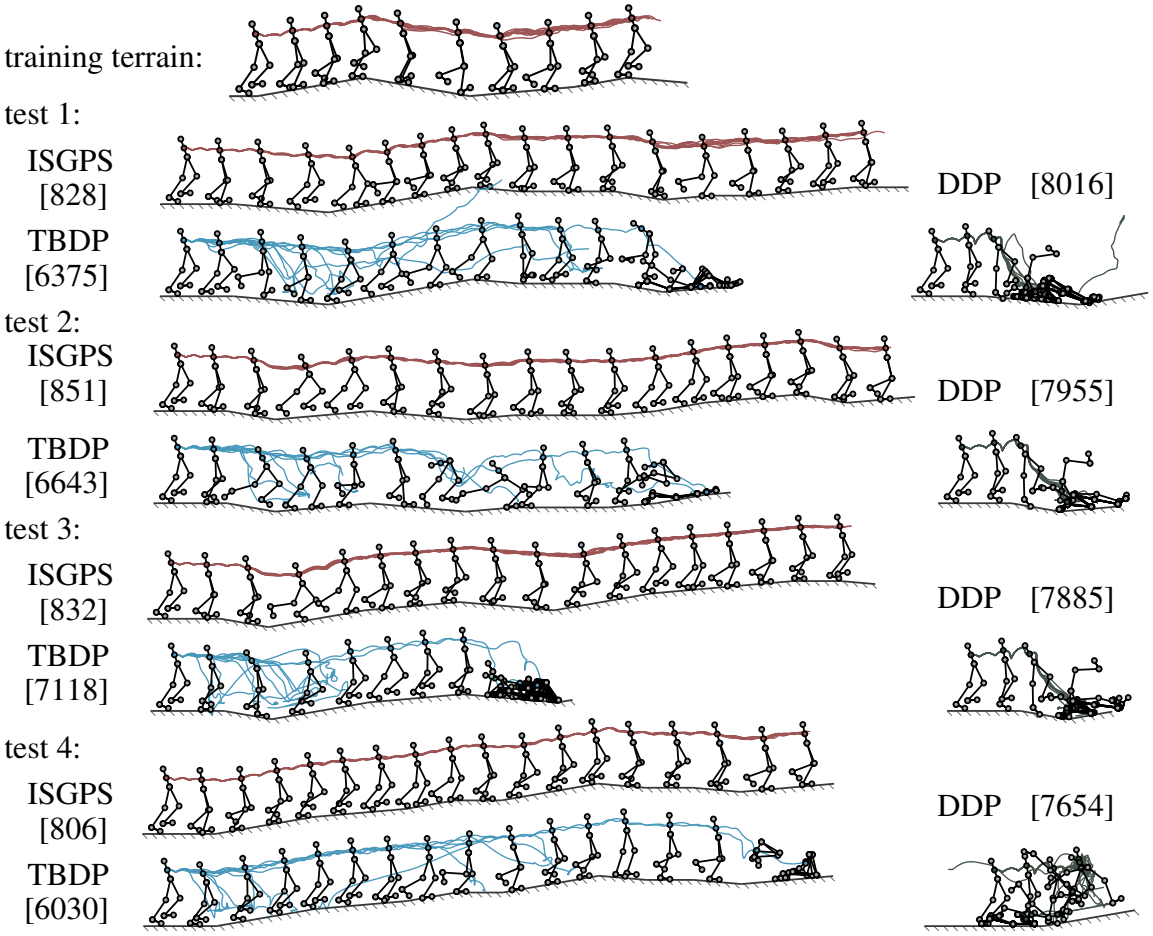


Figure 5.7: Rollouts of ISGPS, TBDP, and DDP on test terrains, with average costs in brackets. All DDP rollouts and most TBDP rollouts fall within a few steps, and all TBDP rollouts fall before reaching the end, while ISGPS generalizes successfully. Colored lines indicate root joint trajectories.

only the state information (joint angles and velocities) provided as input. The omission of the contact and kinematics features significantly increases the difficulty of the problem, to the point that ISGPS can no longer recover a successful policy. While other previous methods, such as DAGGER and PoWER, were also tested, they could not learn reasonable policies and are not included in the results graph.

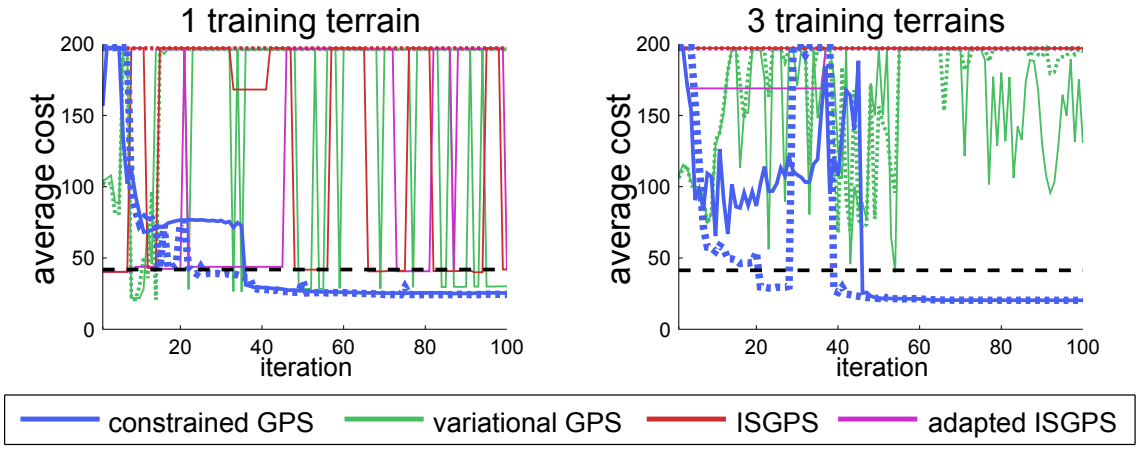


Figure 5.8: Uneven terrain. Solid lines show performance on the training terrains, dotted lines show generalization to unseen terrains. Constrained GPS was able to generalize from both training conditions.

The results of this experiment are shown in Figure 5.8. The solid lines indicate the average cost on the training terrains, while the dotted lines show the average cost when generalizing to four test terrains. The constrained GPS method could learn successful, generalizable policies with either one or three training terrains. Both variational and importance-sampled GPS were actually able to learn a successful policy for traversing the single training terrain by the last iteration, although they experienced considerable oscillation, suggesting that the solution was not entirely stable. More importantly, the resulting learned policies could not generalize effectively to the test terrains, likely due to their lack of stability.

Like the locomotion tasks in the previous section, the main challenge in this task was to learn an effective gait. However, one advantage of general-purpose policies like neural networks is that a single policy can in principle learn multiple strategies, and deploy them as needed in response to the state. This issue is explored in the next section.

5.5.4 Push Recovery

To explore the ability of each GPS algorithm to learn policies that choose different strategies based on the state, the next experiment involved learning walking policies that recover

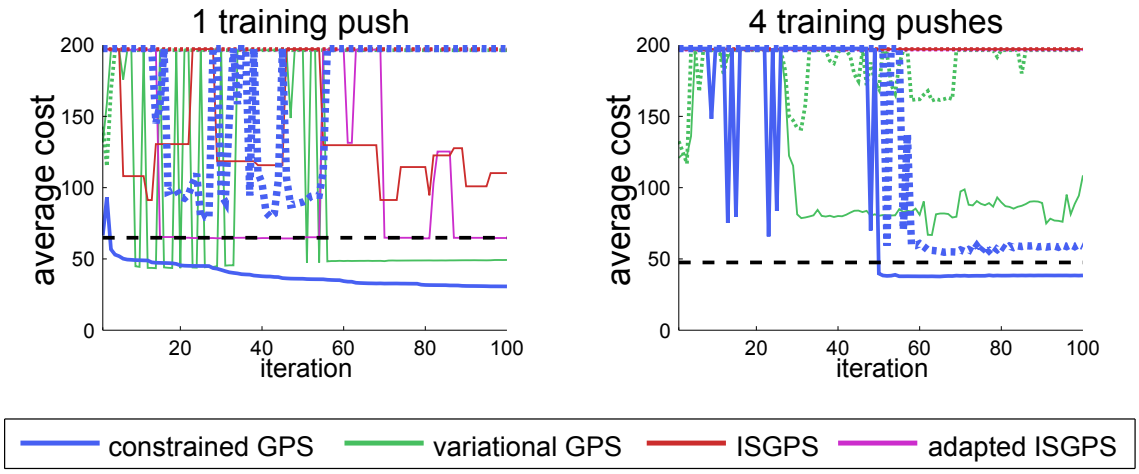


Figure 5.9: Push response results. Solid lines show training push results, dotted lines show generalization. Only constrained GPS was able to learn a successful, generalizable policy from four training pushes.

from strong lateral pushes, in the range of 250 to 500 Newtons, delivered for 100ms. Responding to pushes of such magnitude requires not just disturbance rejection, but a well-timed recovery strategy, such as a protective step or even standing up after a fall. Because the push can occur at different points in the gait, multiple recovery strategies must be learned simultaneously. A strategy that succeeds in one pose may fail in another.

The policies were provided with 50 hidden units, and again used only the joint angles and velocities for input. The experiments involved either one or four training pushes, with four different pushes in the generalization test. Each push could be either forward or backward, could have a force of 250 or 500 Newtons and, most importantly, could occur at any initial state sampled from an example gait cycle. The results are shown in Figure 5.9, with dotted lines indicating performance on the test pushes. Only the constrained GPS algorithm could learn a policy from four training pushes that generalized well to the entire test set. Variational and importance-sampled GPS could recover a policy that succeeded on the single training push, but no method could successfully generalize from this training set, underscoring the importance of learning multiple recovery stragies. With four training pushes, variational GPS learned a policy that succeeded on three of the four training pushes, and one of the four test pushes.

Figure 5.10 shows the recoveries learned by constrained and variational GPS on the training and test pushes. Note that even the weaker 250 Newton pushes are quite violent,

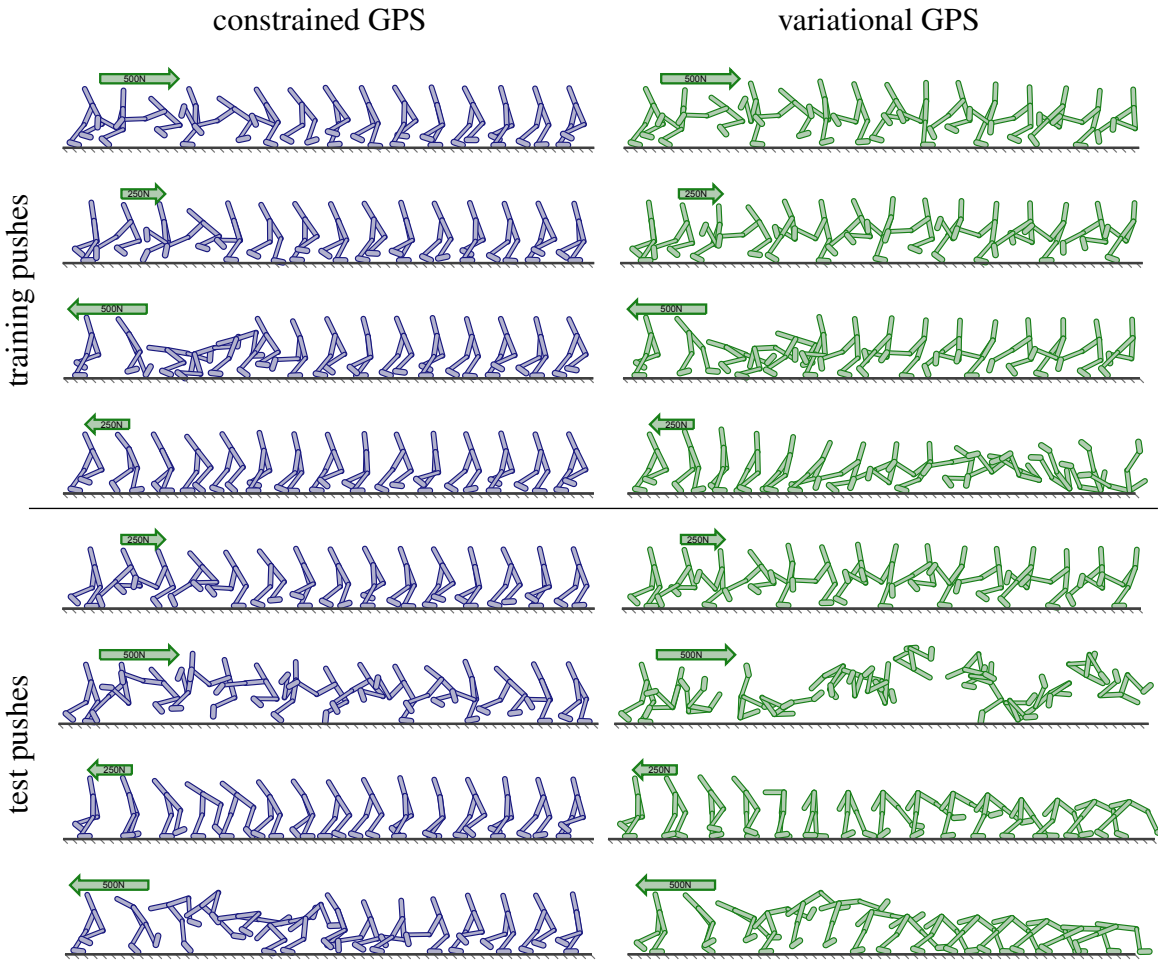


Figure 5.10: Push responses with policies learned by constrained and variational GPS on four training pushes. The horizontal spacing between the figures is expanded for clarity.

and sufficient to lift the 24kg walker off the ground completely. Nonetheless, the constrained GPS policy was able to recover succesfully from all test pushes. A video of the constrained and variational GPS policies on each training and test terrain, as well as an additional sequence of pushes delivered in quick succession, is available on the project website: `http://graphics.stanford.edu/projects/gpspaper/index.htm`. This video illustrates some of the strategies learned by constrained GPS, such as kicking against the ground to recover from a backward fall. The ability to learn such generalizable strategies is one of the key benefits of training expressive parametric controllers.

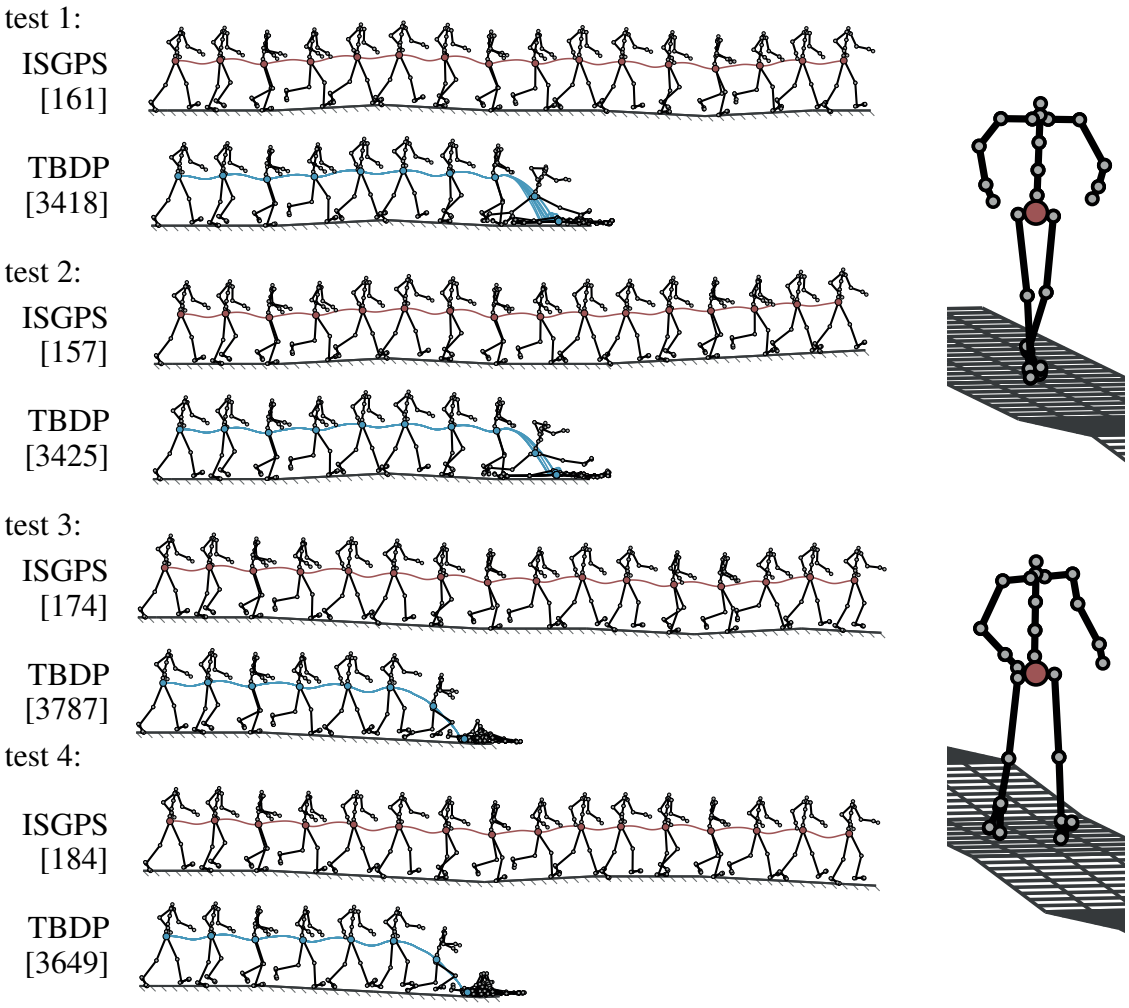


Figure 5.11: Humanoid running on test terrains, with average cost in brackets (left), along with illustrations of the 3D humanoid model (right). ISGPS learned policies that generalized to the test terrains, while the TBDP policy failed to maintain balance.

5.5.5 Humanoid Running

In the final experiment, the importance-sampled GPS algorithm was used to learn a 3D humanoid running gait. This task is highly dynamic, requires good timing and balance, and has 63 state dimensions, making it well suited for exploring the capacity of guided policy search to scale to high-dimensional problems. Previous locomotion methods often rely on hand-crafted components to introduce prior knowledge or ensure stability (Tedrake et al., 2004; Whitman and Atkeson, 2009), while GPS used only general purpose neural networks.

As in Section 5.5.3, the policy was trained on terrain of varying slope, and tested on four random terrains. Due to the complexity of this task, three different training terrains were used, and the policy was provided with 200 hidden units and all of the forward kinematics and contact features. The motor noise was reduced from 10% to 1%. The guiding distributions were initialized from motion capture of a human run, and DDP was used to find the torques that realized this run on each training terrain. Since the example was recorded on flat ground, the terrain contained more mild 3° slopes.

Rollouts from the learned policy on the test terrains are shown in Figure 5.11, with comparisons to TBDP. Importance-sampled GPS generalized to all test terrains, while TBDP could not traverse any of them. This indicates that the task required nontrivial generalization (despite the mild slopes), and that GPS was able to learn generalizable rules to maintain speed and balance. The supplementary video on the project website shows that the learned gait also retained the smooth, lifelike appearance of the human demonstration: `http://graphics.stanford.edu/projects/gpspaper/index.htm`.

5.6 Discussion

In this chapter, I presented three algorithms for training neural network policies by means of trajectory optimization. All three methods use example demonstrations to initialize the trajectory, trajectory optimization to explore the space of possible task executions, and a gradient-based optimization procedure to train the policy. The experimental evaluation demonstrates that the proposed algorithms achieve a dramatic improvement over previous work on a series of challenging locomotion tasks, including full 3D humanoid running.

Although the constrained GPS algorithm achieves the best results in direct comparisons, each of the proposed algorithms has its own strengths and weaknesses. Some of the properties of the three algorithms are summarized in Table 5.1. The variational GPS algorithm, which adapts the trajectories and the policy together within a variational framework, provides the simplest effective approach for trajectory-based policy optimization, since both the E-step and M-step are performed by standard, well known algorithms. The policy training M-step consists of optimizing the policy to match $q(\tau)$, which can be done with

algorithm	overall objective	principal strength	principal weakness
ISGPS	$\min_\theta E_{\pi_\theta}[\ell(\tau)]$	can use non-differentiable features & non-Markovian policy	local optima
Variational GPS	$\max_\theta E_{\pi_\theta}[\rho(\tau)]$	uses only simple methods (DDP & SGD)	unconvential risk-seeking objective
Constrained GPS	$\min_\theta E_{\pi_\theta}[\ell(\tau)] - \mathcal{H}(\pi_\theta)$	best results on all experiments	complex, non-standard trajectory optimizer

Table 5.1: Comparison of the various guided policy search algorithms. The policy is denoted $\pi_\theta(\tau)$, the trajectory distribution is denoted $q(\tau)$, and $\rho(\tau) \propto \exp(-\ell(\tau))$ is the exponential negative cost distribution.

standard supervised learning procedures such as stochastic gradient descent, and the trajectory optimizing E-step is performed with the standard DDP algorithm, using the method in Section 3.4 to construct a distribution over trajectories from the DDP solution. Because of this, implementation of variational GPS is relatively straightforward.

On the other hand, variational GPS optimizes a risk-seeking exponential cost objective, in contrast to importance-sampled GPS, which optimizes the expected cost, and constrained GPS, which optimizes the expected cost minus the entropy, corresponding to the soft optimality objective. This makes importance-sampled and constrained GPS preferable in more complex domains, where the optimal policy might not be representable by any policy from the current policy class.

Importance-sampled GPS has a few significant disadvantages compared to the variational and constrained methods, but the model-free structure of the policy optimization step (when not using adaptive samples) also offers unique benefits. The importance-sampled objective makes importance-sampled GPS more vulnerable to local optima, as low importance weights cause even good samples to be ignored if they have a low probability under the current policy. This also makes adaptation difficult, because although the trajectory can be adapted to better resemble the current policy, there is no guarantee that the subsequent importance-sampled policy optimization will take advantage of the adapted samples.

On the other hand, if adaptation is not used, importance-sampled GPS never needs to

differentiate the policy with respect to the states and actions, unlike variational and constrained GPS, which must differentiate the policy as part of the DDP backward pass. If the policy does not need to be differentiated with respect to the state, it can take advantage of complex features such as contact indicators that are non-trivial to differentiate. For this reason, the humanoid running experiment used the importance-sampled algorithm, since this more complex task requires the more elaborate forward kinematics and contact features. Although the differentiation can still be performed (for example by finite differences) if we prefer to use variational or constrained GPS, it can impose an additional computational burden on these algorithms.

An additional benefit of the importance-sampled algorithm is that it can readily handle non-Markovian policies, such as recurrent neural networks, that maintain their own hidden state between time steps. Such policies may be essential in partially observed domains, where the current observations do not provide complete information about the state of the system. Variational and constrained GPS cannot immediately handle policies with hidden state, because the policy is included in the cost function of trajectory optimization, which must remain Markovian. In the future, a modified variant of these algorithms could be developed that includes the policy state in the overall state of the system, allowing it to be optimized with the same DDP algorithm.

In summary, the three algorithms discussed in this chapter each have their own strengths and weaknesses, and all three are able to solve challenging tasks that are inaccessible for previous reinforcement learning and imitation learning methods. For all three methods, the key ingredient in their success is the integration of trajectory optimization with policy search, with trajectory optimization solving the difficult task of discovering successful executions of the task, while the policy search is reduced to the easier task of matching the optimized trajectory.

Chapter 6

Conclusion

In this thesis, I presented algorithms for learning motor skills by using local trajectory methods. All of the algorithms discussed in this thesis build on the stochastic optimal control theory presented in Chapter 3, which describes how trajectory distributions can be optimized by using local differential dynamic programming methods. In Chapter 4, I showed how this approach can be used to quickly and efficiently learn the objective of a motor skill from example demonstrations, recovering an unknown cost function. This cost function encompasses a concise and portable representation of the skill, and can then be used in conjuction with a policy learning technique to recover a control policy. Algorithms for solving the policy search problem are discussed in Chapter 5 and, like the proposed inverse optimal control (IOC) method, they make use of local trajectory methods. Policy learning is performed by alternating between optimizing a trajectory distribution to minimize both its cost and its deviation from the current policy, and optimizing the policy to achieve low cost, typically by matching the current trajectory distributions. Three algorithms are presented that center around this framework, using either importance sampling, variational inference, or constrained optimization to learn the policy.

The experimental evaluations in Chapters 4 and 5 show that the proposed algorithms can operate on large, complex problems such as 3D humanoid locomotion. Both the IOC and policy search methods are able to solve problems that are inaccessible for prior methods, and to the best of my knowledge, the demonstrated policies are the first learned neural network controllers that can succeed on underactuated bipedal locomotion tasks without

hand-engineered, task-specific features or controller architectures.

In this chapter, I will discuss some high level limitations of the proposed methods and future work that could address them, as well as broader challenges in the areas of inverse optimal control and policy search. I will then discuss the variety of future applications that can be enabled by current and future developments in motor skill learning.

6.1 Challenges and Limitations

Local analysis of trajectories is a common theme in all of the algorithms I described, and the locality of such methods is also the cause of an underlying limitation of such techniques. Since the trajectory distributions used in all of the algorithms are constructed by means of a local linearization, they only remain valid so long as the width of this distribution is small. While this approximation is reasonable in many circumstances, it also introduces a fundamental tension between constructing a distribution with the largest possible entropy, and limiting the distribution to only those regions that are approximated well by linear dynamics. In particular, when the dynamics are only expanded to first order, the distribution typically ends up being too wide, as there is no term to account for dynamics nonlinearities. This results in the IOC algorithm maintaining an overly broad trajectory distribution, which makes the example appear to have a worse likelihood than it really does, and causes the policy search algorithms to learn actions far from the nominal trajectory that may be highly suboptimal. In practice, these issues can be mitigated by increasing the cost magnitude, which directly increases the sharpness of the trajectory distribution, but this introduces an additional free parameter that must be selected.

Furthermore, differentiation of the dynamics can be problematic in tasks with discontinuous dynamics, as can be the case in the presence of contacts. In the presented experiments, the MuJoCo simulator was used with smoothed contact dynamics to enable continuous trajectory optimization (Todorov et al., 2012). While this approach greatly mitigates the problem of contact discontinuities, it requires a tradeoff between simulation realism and differentiability.

For these reasons, a promising direction for future work is to move away from local dynamics expansions based on Taylor series, and instead model the real dynamics within

the trajectory distribution by means of samples. Sample-based trajectory optimization techniques have been proposed in the past, and can serve as the inspiration for future guided policy search and inverse optimal control methods that do not require local expansions (Kalakrishnan et al., 2011b).

The proposed algorithms all assume that a model of the system dynamics is known, and do not currently address the problem of learning it. Previous work has addressed this problem with a variety of regression and active learning techniques (Deisenroth and Rasmussen, 2011; Ross and Bagnell, 2012; Deisenroth et al., 2013), and such methods can in principle be combined with the proposed techniques. However, it may also be possible to use model-free stochastic trajectory optimization techniques that would remove the need for a model altogether. Such methods would still benefit from the trajectory-policy decomposition of guided policy search, but would optimize the trajectory distribution by using local random perturbations, for example as suggested by Kalakrishnan et al. (2011b).

6.2 Future Directions and Applications

Although I presented a number of experiments to evaluate the proposed methods, the full range of applications for these techniques has not yet been thoroughly explored. One promising direction for future work is to explore combined application of inverse optimal control and policy learning. In this thesis, these methods were considered in isolation, but they provide complementary functionality that allows the recovery of complete motor skill policies using only example demonstrations. In combining inverse optimal control and policy search, the IOC algorithm can be modified to learn cost functions that are particularly easy to minimize, analogously to reward shaping. Furthermore, inverse optimal control and a model-free policy search procedure (or a model-based procedure that uses learned dynamics) could be interleaved to allow model-free inverse optimal control, as suggested in recent work on relative entropy inverse reinforcement learning (Boularias et al., 2011).

Another possible direction for future work in motor skill learning is to incorporate developments in the field of deep learning, which deals with training large models from simple inputs to capture high-level structure in the data, often by means of neural networks

(Bengio et al., 2013). Developments in deep learning can inform the details of the controller training procedure, such as effective regularization techniques to prevent overfitting (Vincent et al., 2008; Le et al., 2011; Hinton et al., 2012). More broadly, it can serve as an inspiration for training hierarchical control policies, where appropriate network architectures and regularization are used to learn a meaningful representation of raw input data that is most suitable for control. One direction for developing such ideas is to learn controllers that operate on visual input, acquiring compact control laws similar in spirit to the gaze heuristic discussed in Chapter 1.

Yet another future direction is to learn a single control policy for a broad range of tasks, with the aim of acquiring a representation that allows information sharing across tasks, discovering common themes that enable greater generalization. Since guided policy search algorithms learn a single policy that unifies a variety of trajectories in different environments and with different initial conditions, a sufficiently expressive policy could in principle learn complex sensorimotor associations from a large number of trajectories, all optimized in parallel in their respective environments. The push recovery evaluation in Section 5.5.4 touched on this subject by learning a single policy that encompassed multiple push response strategies. In principle, a single policy could be trained to encompass push recovery, uneven terrain locomotion, standing up after a fall, etc., and by learning all of these skills with a single policy, it could produce an even more robust locomotion controller. This could offer superior generalization and discovery of higher-level motor skills.

Appendix A

Gradients for Inverse Optimal Control

As discussed in Chapter 4, the IOC log-likelihood is given by

$$\log p(\tau|\ell) = \sum_{t=1}^{T} \log p(\mathbf{u}_t|\mathbf{x}_t, \ell) = \sum_{t=1}^{T} -\frac{1}{2} Q_{\mathbf{u}t}^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} + \frac{1}{2} \log |Q_{\mathbf{u},\mathbf{u}t}| - \frac{1}{2} \log 2\pi.$$

To differentiate this objective with respect to the cost parameters $\mathbf{w}$, we can augment the standard DDP recursion with a parallel recursion that maintains the gradients of the value function matrix $\nabla_{\mathbf{w}} V_{\mathbf{x},\mathbf{x}t+1}$ and vector $\nabla_{\mathbf{w}} V_{\mathbf{x}t+1}$. Given these quantities, the gradient of the likelihood for time step t with respect to cost parameter w_k is given by

$$\begin{aligned}\nabla_{w_k} \log p(\mathbf{u}_t|\mathbf{x}_t, \ell) = & -Q_{\mathbf{u}t}^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}^{-1} [\nabla_{w_k} Q_{\mathbf{u}t}] + \frac{1}{2} Q_{\mathbf{u}t}^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}^{-1} [\nabla_{w_k} Q_{\mathbf{u},\mathbf{u}t}] Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} + \\ & \frac{1}{2} \mathrm{tr}\left(Q_{\mathbf{u},\mathbf{u}t}^{-1} [\nabla_{w_k} Q_{\mathbf{u},\mathbf{u}t}]\right),\end{aligned}$$

where the gradients of the Q function matrices are given by

$$\begin{aligned}\nabla_{w_k} Q_{\mathbf{x}\mathbf{u}t} &= \nabla_{w_k} \ell_{\mathbf{x}\mathbf{u}t} + f_{\mathbf{x}\mathbf{u}t}^{\mathrm{T}} [\nabla_{w_k} V_{\mathbf{x}t+1}] \\ \nabla_{w_k} Q_{\mathbf{x}\mathbf{u},\mathbf{x}\mathbf{u}t} &= \nabla_{w_k} \ell_{\mathbf{x}\mathbf{u},\mathbf{x}\mathbf{u}t} + f_{\mathbf{x}\mathbf{u}t}^{\mathrm{T}} [\nabla_{w_k} V_{\mathbf{x},\mathbf{x}t+1}] f_{\mathbf{x}\mathbf{u}t}.\end{aligned}$$

The gradients of the cost matrices $\nabla_{w_k} \ell_{\mathbf{x}\mathbf{u}t}$ and $\nabla_{w_k} \ell_{\mathbf{x}\mathbf{u},\mathbf{x}\mathbf{u}t}$ depend on the specific parameterization of the cost function, as discussed in Sections 4.4 and 4.5. The dynamic programming step can be completed by computed the new value function matrix gradients $\nabla_{\mathbf{w}} V_{\mathbf{x},\mathbf{x}t}$

and $\nabla_{\mathbf{w}} V_{\mathbf{x}t}$ as following:

$$\begin{aligned}
\nabla_{w_k} V_{\mathbf{x}t} &= \nabla_{w_k} Q_{\mathbf{x}t} - [\nabla_{w_k} Q_{\mathbf{x},\mathbf{u}t}] Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} - Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} [\nabla_{w_k} Q_{\mathbf{u}t}] + \\
&\quad Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} [\nabla_{w_k} Q_{\mathbf{u},\mathbf{u}t}] Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u}t} \\
\nabla_{w_k} V_{\mathbf{x},\mathbf{x}t} &= \nabla_{w_k} Q_{\mathbf{x},\mathbf{x}t} - [\nabla_{w_k} Q_{\mathbf{x},\mathbf{u}t}] Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t} - Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} [\nabla_{w_k} Q_{\mathbf{u},\mathbf{x}t}] + \\
&\quad Q_{\mathbf{x},\mathbf{u}t} Q_{\mathbf{u},\mathbf{u}t}^{-1} [\nabla_{w_k} Q_{\mathbf{u},\mathbf{u}t}] Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},\mathbf{x}t}.
\end{aligned}$$

The final gradient with respect to each cost parameter w_k is then given by

$$\nabla_{w_k} \log p(\tau|\ell) = \sum_{t=1}^{T} \nabla_{w_k} \log p(\mathbf{u}_t|\mathbf{x}_t, \ell),$$

which can readily be computed once the value function and its gradients are available at each time step.

Appendix B

Gradient Derivations for ISGPS

In order to optimize the importance-sampled GPS objective $\Phi(\theta)$, we must compute its gradient $\nabla\Phi(\theta)$. We first write the gradient in terms of the gradients of $Z_t(\theta)$ and π_θ:

$$\nabla\Phi(\theta) = \sum_{t=1}^{T}\left[\frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\nabla\pi_\theta(\tau_{i,1:t})}{q(\tau_{i,1:t})}\ell(\mathbf{x}_t^i,\mathbf{u}_t^i) - \frac{\nabla Z_t(\theta)}{Z_t(\theta)^2}\sum_{i=1}^{m}\frac{\pi_\theta(\tau_{i,1:t})}{q(\tau_{i,1:t})}\ell(\mathbf{x}_t^i,\mathbf{u}_t^i) - w_r\frac{\nabla Z_t(\theta)}{Z_t(\theta)}\right].$$

From the definition of $Z_t(\theta)$, we have that

$$\frac{\nabla Z_t(\theta)}{Z_t(\theta)} = \frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\nabla\pi_\theta(\tau_{i,1:t})}{q(\tau_{i,1:t})}.$$

Letting $\tilde{J}_t(\theta) = \frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\pi_\theta(\tau_{i,1:t})}{q(\tau_{i,1:t})}r(\mathbf{x}_t^i,\mathbf{u}_t^i)$, we can rewrite the gradient as

$$\begin{aligned}\nabla\Phi(\theta) &= \sum_{t=1}^{T}\frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\nabla\pi_\theta(\tau_{i,1:t})}{q(\tau_{i,1:t})}\Big[\ell(\mathbf{x}_t^i,\mathbf{u}_t^i) - \tilde{J}_t(\theta) - w_r\Big] \\ &= \sum_{t=1}^{T}\frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\pi_\theta(\tau_{i,1:t})}{q(\tau_{i,1:t})}\nabla\log\pi_\theta(\tau_{i,1:t})\xi_t^i,\end{aligned}$$

using the identity $\nabla\pi_\theta(\tau) = \pi_\theta(\tau)\nabla\log\pi_\theta(\tau)$. When the policy is represented by a large neural network, it is convenient to write the gradient as a sum where the output at each

state appears only once, to produce a set of errors that can be fed into a standard back-propagation algorithm. For a neural network policy with uniform output noise σ and mean $\mu(\mathbf{x}_t)$, we have

$$\nabla \log \pi_\theta(\tau_{i,1:t}) = \sum_t \nabla \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t) = \sum_t \nabla\mu(\mathbf{x}_t)\frac{\mathbf{u}_t - \mu(\mathbf{x}_t)}{\sigma^2},$$

and the gradient of the objective is given by

$$\nabla\Phi(\theta) = \sum_{t=1}^{T}\sum_{i=1}^{m} \nabla\mu(\mathbf{x}_t^i)\frac{\mathbf{u}_t^i - \mu(\mathbf{x}_t^i)}{\sigma^2}\sum_{t'=t}^{T}\frac{1}{Z_{t'}(\theta)}\frac{\pi_\theta(\tau_{i,1:t'})}{q(\tau_{i,1:t'})}\xi_{t'}^i.$$

The gradient can now be computed efficiently by feeding the terms after $\nabla\mu(\mathbf{x}_t^i)$ into the standard back-propagation algorithm.

Appendix C

Gradient Derivations for CGPS

As discussed in Section 5.4.1, the Laplace approximation may not accurately capture the structure of $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ in the entire region where $q(\mathbf{x}_t)$ is large, and since the policy is trained by sampling states from $q(\mathbf{x}_t)$, policy optimization optimizes a different objective. This can lead to nonconvergence when the policy is highly nonlinear. To reconcile this problem, we can approximate the policy terms in the objective with M random samples $\mathbf{x}_{ti}$, drawn from $q(\mathbf{x}_t)$, rather than by using a linearization of the policy:

$$\begin{aligned}\mathcal{L}(q) \approx \sum_{t=1}^{T} \ell(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \frac{1}{2}\mathbf{tr}\left(\Sigma_t \ell_{\mathbf{xu},\mathbf{xu}t}\right) - \frac{1}{2}\log|\mathbf{A}_t| + \\ \frac{\lambda_t}{2M}\sum_{i=1}^{M}(\mathbf{u}_{ti} - \mu_t^\pi(\mathbf{x}_{ti}))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{u}_{ti} - \mu_t^\pi(\mathbf{x}_{ti})) + \\ \frac{\lambda_t}{2}\mathbf{tr}\left(\mathbf{A}_t^{-1}\Sigma_t^\pi\right) + \frac{\lambda_t}{2}\log|\mathbf{A}_t|,\end{aligned}$$

where the actions are given by $\mathbf{u}_{ti} = \mathbf{K}_t\mathbf{x}_{ti} + \hat{\mathbf{u}}_t$. Note that the samples $\mathbf{x}_{ti}$ depend on $\hat{\mathbf{x}}_t$, according to $\mathbf{x}_{ti} = \hat{\mathbf{x}}_t + \mathbf{L}_t^{\mathrm{T}}\mathbf{s}_{ti}$, where $\mathbf{s}_{ti}$ is a sample from a zero-mean spherical Gaussian, and $\mathbf{L}_t$ is the upper triangular Cholesky decomposition of $\mathbf{S}_t$.[1] As before, we differentiate

[1] Keeping the same samples $\mathbf{s}_{ti}$ across iterations reduces variance and can greatly improve convergence.

with respect to $\hat{\mathbf{u}}_t$, substituting $Q_{\mathbf{u},\mathbf{u}t}$ and $Q_{\mathbf{u}t}$ as needed:

$$\mathcal{L}_{\mathbf{u}t} = Q_{\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1} \hat{\mu}_t^{\pi}$$
$$\mathcal{L}_{\mathbf{u},\mathbf{u}t} = Q_{\mathbf{u},\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1},$$

where $\hat{\mu}_t^{\pi} = \frac{1}{M} \sum_{i=1}^{M} (\mathbf{u}_{ti} - \mu_t^{\pi}(\mathbf{x}_{ti}))$ is the average difference between the linear feedback and the policy. This yields the following correction and feedback terms:

$$\mathbf{k}_t = -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1}\right)^{-1} \left(Q_{\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1} \hat{\mu}_t^{\pi}\right)$$
$$\mathbf{K}_t = -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1}\right)^{-1} \left(Q_{\mathbf{u},\mathbf{x}t} - \lambda_t \mathbf{A}_t^{-1} \hat{\mu}_{\mathbf{x}t}^{\pi}\right),$$

where $\hat{\mu}_{\mathbf{x}t}^{\pi} = \frac{1}{M} \sum_{i=1}^{M} \mu_{\mathbf{x}t}^{\pi}(\mathbf{x}_{ti})$ is the average policy gradient. So far, the change to the mean is identical to simply averaging the policy values and policy gradients over all the samples. In fact, a reasonable approximation can be obtained by doing just that, and substituting the sample averages directly into the equations in Section 5.4.1. This is the approximation used in the experimental evaluation, as it is slightly faster and does not appear to significantly degrade the results. However, the true gradients with respect to $\mathbf{A}_t$ are different. Below, the objective is differentiated with respect to $\mathbf{A}_t$ and $\hat{\mathbf{K}}_t$ as before, where $\hat{\mathbf{K}}_t$ is used to distinguish the covariance term from the feedback in the first dynamic programming pass, which is no longer identical. The derivatives with respect to $\mathbf{A}_t$ and $\hat{\mathbf{K}}_t$ are

$$\mathcal{L}_{\mathbf{A}t} = \frac{1}{2} Q_{\mathbf{u},\mathbf{u}t} + \frac{\lambda_t - 1}{2} \mathbf{A}_t^{-1} - \frac{\lambda_t}{2} \mathbf{A}_t^{-1} \mathbf{M} \mathbf{A}_t^{-1}$$
$$\mathcal{L}_{\hat{\mathbf{K}}t} = Q_{\mathbf{u},\mathbf{u}t} \hat{\mathbf{K}}_t \mathbf{S}_t + Q_{\mathbf{u},\mathbf{x}t} \mathbf{S}_t + \lambda_t \mathbf{A}_t^{-1} (\hat{\mathbf{K}}_t \hat{\mathbf{S}}_t - \mathbf{C}_t),$$

where $\mathbf{M} = \Sigma_t^{\pi} + \sum_{i=1}^{M} (\mathbf{u}_{ti} - \mu_t^{\pi}(\mathbf{x}_{ti}))(\mathbf{u}_{ti} - \mu_t^{\pi}(\mathbf{x}_{ti}))$, $\hat{\mathbf{S}}_t = \frac{1}{M} \sum_{i=1}^{M} \mathbf{x}_{ti} \mathbf{x}_{ti}^{\mathrm{T}}$, and $\mathbf{C}_t = \frac{1}{M} \sum_{i=1}^{M} \mu_t^{\pi}(\mathbf{x}_{ti}) \mathbf{x}_{ti}^{\mathrm{T}}$, where we simplified using the assumption that $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ are zero. We can again solve for $\mathbf{A}_t$ by solving the CARE in Equation 5.14. To solve for $\hat{\mathbf{K}}_t$, we rearrange the terms to get

$$\hat{\mathbf{K}}_t \hat{\mathbf{S}}_t \mathbf{S}_t^{-1} + \frac{1}{\lambda_t} \mathbf{A}_t Q_{\mathbf{u},\mathbf{u}t} \hat{\mathbf{K}}_t = \mathbf{C}_t \mathbf{S}_t^{-1} - \frac{1}{\lambda_t} \mathbf{A}_t Q_{\mathbf{u},\mathbf{x}t}.$$

This equation is linear in $\hat{\mathbf{K}}_t$, but requires solving a sparse linear system with dimensionality equal to the number of entries in $\hat{\mathbf{K}}_t$, which increases the computational cost.

Differentiating with respect to $\mathbf{S}_t$, we get:

$$\begin{aligned}\mathcal{L}_{\mathbf{S}t} = \frac{1}{2}\big[&Q_{\mathbf{x},\mathbf{x}t} + \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{x}t} + Q_{\mathbf{x},\mathbf{u}t}\mathbf{K}_t + \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}\mathbf{K}_t \\ &+\text{choldiff}(\mathbf{D}_t) + \text{choldiff}(\mathbf{D}_t)^{\mathrm{T}}\big]\end{aligned}$$

where $\mathbf{D}_t = \frac{\lambda_t}{M}\sum_{i=1}^{M}(\hat{\mathbf{K}}_t - \mu^{\pi}_{\mathbf{x}t}(\mathbf{x}_{ti}))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{u}_{ti} - \mu^{\pi}_t(\mathbf{x}_{ti}))\mathbf{s}_{ti}^{\mathrm{T}}$ and choldiff$(\dots)$ indicates the differentiation of the Cholesky decomposition, for example using the method described by Giles (2008). While this will provide the correct gradient, choldiff$(\mathbf{D}_t) + $ choldiff$(\mathbf{D}_t)^{\mathrm{T}}$ is not guaranteed to be positive definite. In this case, it is useful to regularize the gradient by interpolating it with the one obtained from the Laplace approximation.

Bibliography

Abbeel, P., Coates, A., Quigley, M., and Ng, A. (2006). An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems (NIPS)*.

Abbeel, P. and Ng, A. (2004). Apprenticeship learning via inverse reinforcement learning. In *ICML*.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483.

Arikan, O. and Forsyth, D. A. (2002). Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3):483–490.

Arnold, W.F., I. and Laub, A. (1984). Generalized eigenproblem algorithms and software for algebraic riccati equations. *Proceedings of the IEEE*, 72(12):1746–1754.

Atkeson, C. and Morimoto, J. (2002). Nonparametric representation of policies and value functions: A trajectory-based approach. In *Advances in Neural Information Processing Systems (NIPS)*.

Atkeson, C. and Stephens, B. (2008). Random sampling of states in dynamic programming. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 38(4):924–929.

Bagnell, A., Kakade, S., Ng, A., and Schneider, J. (2003). Policy search by dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*.

Baxter, J., Bartlett, P., and Weaver, L. (2001). Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381.

Bellman, R. E. and Dreyfus, S. E. (1962). *Applied Dynamic Programming*. Princeton University Press, Princeton, NJ.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Bertsekas, D. P. (2001). *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA.

Birgin, E. G. and Martínez, J. M. (2009). Practical augmented lagrangian methods. In *Encyclopedia of Optimization*, pages 3013–3023.

Boularias, A., Kober, J., and Peters, J. (2011). Relative entropy inverse reinforcement learning. In *AISTATS*, pages 182–189.

Boyan, J. A. and Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems (NIPS)*.

Boyd, S., El Ghaoui, L., Feron, E., and Balakrishnan, V. (1994). *Linear Matrix Inequalities in System and Control Theory*. SIAM, Philadelphia, PA.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Byl, K. and Tedrake, R. (2008). Approximate optimal control of the compass gait on rough terrain. In *International Conference on Robotics and Automation*.

Coates, A., Abbeel, P., and Ng, A. (2008). Learning for control from multiple demonstrations. In *International Conference on Machine Learning (ICML)*.

Coros, S., Beaudoin, P., and van de Panne, M. (2010). Generalized biped walking control. *ACM Transactions on Graphics*, 29(4).

Daniel, C., Neumann, G., and Peters, J. (2012). Hierarchical relative entropy policy search. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Deisenroth, M., Neumann, G., and Peters, J. (2013). A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142.

Deisenroth, M. and Rasmussen, C. (2011). PILCO: a model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*.

Diehl, M., H., F., and N., H. (2009). Efficient numerical methods for nonlinear MPC and moving horizon estimation. *Nonlinear Model Predictive Control*.

Dvijotham, K. and Todorov, E. (2010). Inverse optimal control with linearly-solvable MDPs. In *International Conference on Machine Learning (ICML)*.

Furmston, T. and Barber, D. (2010). Variational methods for reinforcement learning. *Journal of Machine Learning Research*, 9:241–248.

Gayme, D. and Topcu, U. (2012). Optimal power flow with large-scale storage integration. *IEEE Transactions on Power Systems*, 28(2):709–717.

Geyer, H. (2005). *Simple models of legged locomotion based on compliant limb behavior*. PhD thesis, Friedrich-Schiller-Universität Jena.

Geyer, H. and Herr, H. (2010). A muscle-reflex model that encodes principles of legged mechanics produces human walking dynamics and muscle activities. *Neural Systems and Rehabilitation Engineering*, 18(3):263–273.

Giles, M. (2008). An extended collection of matrix derivative results for forward and reverse mode algorithmic differentiatio. Technical Report 08/01, Oxford University.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *IEEE Conference on Evolutionary Computation*.

Herbort, O., Butz, M. V., and Pedersen, G. (2009). The SURE_REACH model for motor learning and control of a redundant arm: from modeling human behavior to applications in robotics. *From Motor to Interaction Learning in Robots*, pages 83 – 104.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. abs/1207.0580.

Ijspeert, A., Nakanishi, J., and Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. In *International Conference on Robotics and Automation*.

Ijspeert, A., Nakanishi, J., and Schaal, S. (2003). Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems (NIPS)*.

Jacobson, D. and Mayne, D. (1970). *Differential Dynamic Programming*. Elsevier.

Jain, A., Wojcik, B., Joachims, T., and Saxena, A. (2013). Learning trajectory preferences for manipulators via iterative improvement. In *Advances in Neural Information Processing Systems (NIPS)*.

Johansen, R. S. (2009). Automated semi-procedural animation for character locomotion. Master's thesis, Aarhus University.

John, C. T., Andreson, F. C., Higginson, J. S., and Delp, S. L. (2012). Stabilisation of walking by intrinsic muscle properties revealed in a three-dimensional muscle-driven simulation. *Computer Methods in Biomechanics and Biomedical Engineering*, 16(4):451–462.

Kaelbling, L., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Kalakrishnan, M., Buchli, J., Pastor, P., Mistry, M., and Schaal, S. (2011a). Learning, planning, and control for quadruped locomotion over challenging terrain. *International Journal of Robotics Research*, 30(236).

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., and Schaal, S. (2011b). STOMP: stochastic trajectory optimization for motion planning. In *International Conference on Robotics and Automation*.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *ASME Transactions Journal of Basic Engineering*, 82(1):35–45.

Kappen, H. J., Gómez, V., and Opper, M. (2012). Optimal control as a graphical model inference problem. *Machine Learning*, 87(2):159–182.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *International Journal of Robotic Research*, 32(11):1238–1274.

Kober, J. and Peters, J. (2009). Learning motor primitives for robotics. In *International Conference on Robotics and Automation (ICRA)*.

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *International Conference on Robotics and Automation*.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Kolter, J. Z., Coates, A., Ng, A. Y., Gu, Y., and DuHadway, C. (2008). Space-indexed dynamic programming: learning to follow trajectories. In *International Conference on Machine Learning (ICML)*.

Kolter, J. Z., Jackowski, Z., and Tedrake, R. (2012). Design, analysis and learning control of a fully actuated micro wind turbine. In *Proceedings of the 2012 American Control Conference (ACC)*.

Konidaris, G., Kuindersma, S., Barto, A. G., and Grupen, R. A. (2010). Constructing skill trees for reinforcement learning agents from demonstration trajectories. In *Advances in Neural Information Processing Systems (NIPS)*.

Konidaris, G. D. and Osentoski, S. (2008). Value function approximation in reinforcement learning using the fourier basis. Technical Report UM-CS-2008-19, Department of Computer Science, University of Massachusetts Amherst.

Kovar, L., Gleicher, M., and Pighin, F. H. (2002). Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482.

Lampe, T. and Riedmiller, M. (2013). Acquiring visual servoing reaching and grasping skills using neural reinforcement learning. In *IEEE International Joint Conference on Neural Networks (IJCNN)*.

Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., and Ng, A. Y. (2011). On optimization methods for deep learning. In *International Conference on Machine Learning (ICML)*.

Lee, J., Chai, J., Reitsma, P. S. A., Hodgins, J. K., and Pollard, N. S. (2002). Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics*, 21(3):491–500.

Lee, J. and Lee, K. (2006). Precomputing avatar behavior from human motion data. *Graphical Models*, 68(2):158–174.

Lee, Y., Kim, S., and Lee, J. (2010a). Data-driven biped control. *ACM Transactions on Graphics*, 29(4):129:1–129:8.

Lee, Y., Lee, S. J., and Popović, Z. (2009). Compact character controllers. *ACM Transactions on Graphics*, 28(5):169:1–169:8.

Lee, Y., Wampler, K., Bernstein, G., Popović, J., and Popović, Z. (2010b). Motion fields for interactive character locomotion. *ACM Transactions on Graphics*, 29(6):138:1–138:8.

Levine, S. (2013). Exploring deep and recurrent architectures for optimal control. In *NIPS 2013 Workshop on Deep Learning*.

Levine, S. and Koltun, V. (2012). Continuous inverse optimal control with locally optimal examples. In *International Conference on Machine Learning (ICML)*.

Levine, S. and Koltun, V. (2013a). Guided policy search. In *International Conference on Machine Learning (ICML)*.

Levine, S. and Koltun, V. (2013b). Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems (NIPS)*.

Levine, S., Krähenbühl, P., Thrun, S., and Koltun, V. (2010a). Gesture controllers. volume 29, pages 124:1–124:11.

Levine, S., Lee, Y., Koltun, V., and Popović, Z. (2011a). Space-time planning with parameterized locomotion controllers. *ACM Transactions on Graphics*, 30(3):23:1–23:11.

Levine, S. and Popovic, J. (2012). Physically plausible simulation for character animation. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*.

Levine, S., Popović, Z., and Koltun, V. (2010b). Feature construction for inverse reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Levine, S., Popović, Z., and Koltun, V. (2011b). Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems (NIPS)*.

Levine, S., Theobalt, C., and Koltun, V. (2009). Real-time prosody-driven synthesis of body language. *ACM Transactions on Graphics*, 28(5):172:1–172:10.

Levine, S., Wang, J. M., Haraux, A., Popovic, Z., and Koltun, V. (2012). Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics*, 31(4):28.

Lewis, F. L., Jagannathan, S., and Yesildirek, A. (1999). *Neural Network Control of Robot Manipulators and Nonlinear Systems*. Taylor and Francis.

Li, W. and Todorov, E. (2004). Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229.

Liu, L., Yin, K., van de Panne, M., Shao, T., and Xu, W. (2010). Sampling-based contact-rich motion control. *ACM Transactions on Graphics*, 29(3).

Lo, W. and Zwicker, M. (2008). Real-time planning for parameterized human motion. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 29–38.

Maei, H. R., Szepesvari, C., S., B., Precup, D., Silver, D., and Sutton, R. (2010). Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems (NIPS)*.

Mannor, S., Rubinstein, R. Y., and Gat, Y. (2003). The cross-entropy method for fast policy search. In *International Conference on Machine Learning (ICML)*.

McLeod, P. and Dienes, Z. (1996). Do fielders know where to go to catch the ball or only how to get there? *Journal of Experimental Psychology*, 22(3):531–543.

Meyer, D. E., Abrams, R. A., Kornblum, S., Wright, C. E., and Smith, J. E. K. (1988). Optimality in human motor performance: Ideal control of rapid aimed movements. *Psychological Review*, 95(3):340–370.

Miller, W. T., Sutton, R., and Werbos, P. J. (1990). *Neural Networks for Control*. MIT Press, Cambridge, MA, USA.

Morimoto, J., Cheng, G., Atkeson, C., and Zeglin, G. (2004). A simple reinforcement learning algorithm for biped walking. In *International Conference on Robotics and Automation*.

Muico, U., Lee, Y., Popović, J., and Popović, Z. (2009). Contact-aware nonlinear control of dynamic characters. *ACM Transactions on Graphics*, 28(3):81:1–81:9.

Nelson, W. L. (1983). Physical principles for economies of skilled movement. *Biological Cybernetics*, 46(2):135–147.

Neumann, G. (2011). Variational inference for policy search in changing situations. In *International Conference on Machine Learning (ICML)*.

Ng, A. Y. and Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence*, pages 406–415.

Ng, A. Y. and Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of ICML*.

Oculus VR (2014). Oculus Rift. http://www.oculusvr.com.

Pandy, M. G., Zajac, F. E., Sim, E., and Levine, W. S. (1990). An optimal control model for maximum-height human jumping. *Journal of Biomechanics*, 23(12):1185–1198.

Paraschos, A., Daniel, C., Peters, J., and Neumann, G. (2013). Probabilistic movement primitives. In *Advances in Neural Information Processing Systems (NIPS)*.

Park, T. and Levine, S. (2013). Inverse optimal control for humanoid locomotion. In *Robotics Science and Systems Workshop on Inverse Optimal Control and Robotic Learning from Demonstration*.

Pascanu, R. and Bengio, Y. (2012). On the difficulty of training recurrent neural networks. Technical Report arXiv:1211.5063, Universite de Montreal.

Peshkin, L. and Shelton, C. (2002). Learning from scarce experience. In *International Conference on Machine Learning (ICML)*.

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

Pontryagin, L. S. (1964). *The mathematical theory of optimal processes*. Pergamon Press.

Raibert, M. H. and Hodgins, J. K. (1991). Animation of dynamic legged locomotion. In *ACM SIGGRAPH*.

Ramachandran, D. and Amir, E. (2007). Bayesian inverse reinforcement learning. In *International Joint Conference on Artifical Intelligence*.

Ratliff, N., Bagnell, J. A., and Zinkevich, M. A. (2006). Maximum margin planning. In *Proceedings of ICML*.

Ratliff, N., Silver, D., and Bagnell, J. A. (2009). Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53.

Reinbolt, J. A., Seth, A., and Delp, S. L. (2011). Simulation of human movement: applications using opensim. *Procedia IUTAM*, 2:186–198.

Righetti, L., Kalakrishnan, M., Pastor, P., Binney, J., Kelly, J., Voorhies, R., Sukhatme, G., and Schaal, S. (2014). An autonomous manipulation system based on force control and optimization. *Autonomous Robots*, 36(1-2):11–30.

Rosenbaum, D. A. (1991). *Human Motor Control*. Academic Press, San Diego, CA, USA.

Ross, S. and Bagnell, A. (2012). Agnostic system identification for model-based reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Ross, S., Gordon, G., and Bagnell, A. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research*, 15:627–635.

Ross, S., Melik-Barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A., and Hebert, M. (2013). Learning monocular reactive UAV control in cluttered natural environments. In *International Conference on Robotics and Automation (ICRA)*.

Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optmization*. Springer-Verlag, New York, NY, USA.

Safonova, A. and Hodgins, J. K. (2007). Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics*, 26(3):106.

Salmon, D. (1995). Neurobiology of skill and habit learning. *Current Opinion in Neurobiology*, 5(2):184–190.

Schaal, S. (2003). Dynamic movement primitives - a framework for motor control in humans and humanoid robots. In *The International Symposium on Adaptive Motion of Animals and Machines*.

Schulman, J., Ho, J., Lee, A., Awwal, I., Bradlow, H., and Abbeel, P. (2013). Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*.

Sciavicco, L. and Siciliano, B. (2007). *Modeling and control of robot manipulators*. MacGraw-Hill, Heidelberg, Germany.

Singh, S. and Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

Sok, K., Kim, M., and Lee, J. (2007). Simulating biped behaviors from human motion data. *ACM Transactions on Graphics*, 26(3):107.

Stulp, F. and Sigaud, O. (2012). Path integral policy improvement with covariance matrix adaptation. In *International Conference on Machine Learning (ICML)*.

Stulp, F., Theodorou, E., and Schaal, S. (2012). Reinforcement learning with sequences of motion primitives for robust manipulation.

Sutton, R. (1988). Learning to predict by the method of temporal difference. *Machine Learning*, 3:9–44.

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS)*.

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA, USA.

Tang, J. and Abbeel, P. (2010). On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems (NIPS)*.

Tassa, Y., Erez, T., and Smart, W. D. (2007). Receding horizon differential dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*.

Tassa, Y., Erez, T., and Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Tedrake, R., Zhang, T., and Seung, H. (2004). Stochastic policy gradient reinforcement learning on a simple 3d biped. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Tesauro, G. J. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38:58–68.

Theodorou, E., Buchli, J., and Schaal, S. (2010). Reinforcement learning of motor skills in high dimensions: a path integral approach. In *International Conference on Robotics and Automation (ICRA)*.

Thoroughman, K. A. and Shadmehr, R. (2000). Learning of action through adaptive combination of motor primitives. *Nature*, 407:742–747.

Tierney, L. and Kadane, J. B. (1986). Accurate approximations for posterior moments and marginal densities. *Journal of the American Statistical Association*, 81(393):82–86.

Todorov, E. (2006a). Linearly-solvable markov decision problems. In *Advances in Neural Information Processing Systems (NIPS)*.

Todorov, E. (2006b). Optimal control theory. *Bayesian brain: probabilistic approaches to neural coding*, pages 269–298.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Todorov, E. and Li, W. (2005). A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference*.

Toussaint, M. (2009). Robot trajectory optimization using approximate inference. In *International Conference on Machine Learning (ICML)*.

Toussaint, M., Charlin, L., and Poupart, P. (2008). Hierarchical POMDP controller optimization by likelihood maximization. In *Uncertainty in Artificial Intelligence (UAI)*.

Treuille, A., Lee, Y., and Popović, Z. (2007). Near-optimal character animation with continuous control. *ACM Transactions on Graphics*, 26(3):7:1–7:8.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning (ICML)*.

Vlassis, N., Toussaint, M., Kontes, G., and Piperidis, S. (2009). Learning model-free robot control by a Monte Carlo EM algorithm. *Autonomous Robots*, 27(2):123–130.

Wampler, K. and Popović, Z. (2009). Optimal gait and form for animal locomotion. *ACM Transactions on Graphics*, 28(3).

Wamplers, K., Andersen, E., Herbst, E., Lee, Y., and Popović, Z. (2010). Character animation in two-player adversarial games. *ACM Transactions on Graphics*, 29(5).

Wamplers, K., Popović, J., and Popović, Z. (2013). Animal locomotion controllers from scratch. *Computer Graphics Forum*.

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2009). Optimizing walking controllers. *ACM Transactions on Graphics*, 28(5):168:1–168:8.

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2010). Optimizing walking controllers for uncertain inputs and environments. *ACM Transactions on Graphics*, 29(3).

Wang, J. M., Hamner, S. R., Delp, S. L., and Koltun, V. (2012). Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Transactions on Graphics*, 31(4):25.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Webb, J. D., Blemker, S. S., and Delp, S. L. (2012). 3d finite element models of shoulder muscles for computing lines of actions and moment arms. *Computer Methods in Biomechanics and Biomedical Engineering*.

Whitman, E. and Atkeson, C. (2009). Control of a walking biped using a combination of simple policies. In *9th IEEE-RAS International Conference on Humanoid Robots*.

Williams, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Woodworth, R. S. (1899). The accuracy of voluntary movement. *Psychological Review Monographs*, 3(3):i–114.

Yin, K., Loken, K., and van de Panne, M. (2007). SIMBICON: simple biped locomotion control. *ACM Transactions Graphics*, 26(3).

Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19(2):4–10.

Ziebart, B. (2010). *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, Carnegie Mellon University.