January 2011

Optimal Control for Autonomous Motor Behavior

Tom Erez
Washington University in St. Louis

WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering and Applied Science

Department of Computer Science and Engineering

Thesis Examination Committee:
William D. Smart, Chair
Jeremy Buhler
Daniel Moran
Robert Pless
Emanuel Todorov
Kilian Weinberger

OPTIMAL CONTROL FOR AUTONOMOUS MOTOR BEHAVIOR

by

Tom Erez

A dissertation presented to the
Graduate School of Arts and Sciences
of Washington University in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2011
Saint Louis, Missouri

ABSTRACT OF THE DISSERTATION

Optimal Control for Autonomous Motor Behavior

by

Tom Erez

Doctor of Philosophy in Computer Science

Washington University in St. Louis, 2011

Research Advisor: Professor William D. Smart

This dissertation presents algorithms that allow robots to generate optimal behavior from first principles. Instead of hard-coding every desired behavior, we encode the task as a cost function, and use numerical optimization to find action sequences that can accomplish the task. Using the theoretical framework of optimal control, we develop methods for generating autonomous motor behavior in high-dimensional domains of legged locomotion.

We identify three foundational problems that limit the application of existing optimal control algorithms, and present guiding principles that address these issues. First, some traditional algorithms use global optimization, where every possible state is considered. This approach cannot be applied in continuous domains, where every additional mechanical degree of freedom exponentially increases the volume of state space. In order to sidestep this curse of dimensionality, we focus on trajectory optimization, which finds locally-optimal solutions while scaling only polynoimally with state dimensionality. Second, many algorithms of optimal control and reinforcement learning cannot be directly applied to continuous domains with contacts, due to the non-smooth dynamics. We present techniques of contact smoothing that enable the use of standard continuous optimization tools. Finally, domains of legged locomotion give rise to extremely non-convex optimization, which is riddled with local minima. This issue is addressed by using a shaping protocol (homotopy-continuation), whereby the solution of an easier variant of the problem is used as initial guess for a harder variant of the problem, gradually finding a solution to the original domain.

In dynamic environments, effective behavior requires feedback control, which provides appropriate reaction to changing circumstance. We present tools that generate adaptive autonomous behavior through local optimization, and scale well to domains with over 60 state-space dimensions. We present a series of experiments that demonstrate robust and reactive motor behavior in domains of increasing complexity. First, a multi-joint swimmer which can follow a moving target, withstand violent perturbations, and avoid moving obstacles. Second, a one-legged hopper that can maintain its gait through both perturbations and morphological alteration. And finally, two bipeds: a planar walker, and a 3D runner that uses its arms to balance its gait.

Acknowledgments

More people than I can count contributed to this project, and it's hard to express my gratefulness to all of them. First and foremost, I owe my family infinite gratitude for their support and continuing inspiration: my grandfather, who was a biology teacher, my parents, who are avid readers and thinkers, and my wife, Anna, who shares my passion and devotion to the academic world.

I feel fortunate to have spent the past few years in St. Louis, and I am grateful for many things: the communal lifestyle, the extended social circle of the MFA students in creative writing, and the overall amiable atmosphere. I felt at home in the Media and Machines lab, where I had many insightful conversations with friends and colleagues, including (but not limited to) Michael Dixon, Nathan Jacobs, and Austin Abrams. The superbly-managed CSE department always took good care of me, and I thank Madeline Hawkins, Kelly Eckman, Myrna Harbison, Sharon Matlock, and Ron Cytron.

My path into the world of studying artificial behavior was jump-started when my father gave me "Godel Escher Bach" [Hofstadter, 1979], a book that still shapes my view on intelligence, artificial or otherwise. In particular, the Copycat model of Melanie Mitchell [Hofstadter, 1996] solidified my ambition to study open-ended behavior in simulation models.

My partnership with Yuval Tassa was the bedrock of this research. We became friends in the corridors of the Interdisciplinary Center for Neural Computation in 2003, and ever since then he taught me so much: he introduced me to Doya [2000], the swimmers of Coulom [2002], and the vast world of control theory and motor learning. We went to our first conference[1] together, advised on each other's careers, and talked almost every day for the past 7 years.

I was fortunate to receive professional support and mentorship from many wise people. Sorin Solomon and Uri Hershberg initiated my academic immersion, first in Jerusalem and then in Turin; Robert Pless was infinitely generous with his time, attention,

[1] http://iridia.ulb.ac.be/~bersini/Varela/

and resources; Herman Pontzer welcomed me to his lab and introduced me to the world of biomechanics; Dan Moran offered heaps of biomechanical motivation. I am particularly grateful to Emanuel Todorov, whose profound scientific insight and kind nature provided me with guidance and tools at a critical moment of the project. Last but not least, my advisor and friend, Bill Smart — his inspiring research brought me to St. Louis, his unending faith kept me funded and focused, his comments and encouragement helped me stay on track. I wish that one day I will be able to repay this debt by sharing what I learned with others.

Tom Erez

Washington University in Saint Louis
May 2011

This work was partially supported by the National Science Foundation, under awards BCS-0924609 and IIS-0917199, and by the Washington University School of Arts and Sciences.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We are interested in developing tools that can endow artificial agents with motor intelligence — the capacity to behave autonomously in a complex physical environment, anticipating the results of your actions, achieving goals and avoiding pitfalls. Many of the robotic systems built today derive their motor controllers from a patchwork of heuristics, hard-coded by a dedicated team of control engineers. This approach can generate effective and robust feedback control, such as Boston Dynamics' Big Dog robot, which can recover its gait even when violently perturbed. In contrast, we wish to develop algorithms that can generate reactive motor behavior from first principles, and can be applied to a variety of domains without demanding human input and excessive manual tuning. The optimization methods presented here allow for the emergence of robust controllers in a variety of domains, and our experiments show simulated robots that can demonstrate "motor creativity", i.e., open-ended behavior, which was not pre-programmed.

We take a model-based approach, studying tasks which are defined by a general-form cost function[2] and a dynamic model, both treated as a black box (meaning that no analytic manipulation of these functions is needed). This is an appealing paradigm, because it offers an intuitive scheme — it is easier to specify which states are desirable than to directly craft the policy that realizes these goals. We focus on optimizing behavior in continuous, high-dimensional domains, and particular attention is given to legged locomotion, which is studied as a test-case for domains with contacts. In such cases, the dynamics change dramatically according to whether a contact is made or broken, giving rise to particularly challenging optimization problems.

[2] In this dissertation I make the arbitrary choice of defining optimization in terms of minimizing costs, instead of the equivalent formulation of maximizing rewards.

This work is couched in two sibling domains: Computational Reinforcement Learning [Sutton and Barto, 1998], a theoretical framework for optimizing sequential decision-making (i.e., behavior), and Optimal Control [Stengel, 1994], a discipline within control theory that deals with a similar set of questions. The methods presented in this dissertation draw on the knowledge of both fields in equal amounts, and the terminologies of both are used interchangeably.

1.1 Three challenges

This work stems from the identification of three major challenges which must be overcome when applying optimal control to high-dimensional domains with contacts. Here I explain the fundamental difficulty in each case, and review the guiding principles I propose to handle them. The main contribution of this dissertation is the development of methods that are effective in the face of these issues.

1.1.1 The curse of dimensionality

Many techniques of reinforcement learning and optimal control identify the globally-optimal policy by applying some form of dynamic programming to an enumeration of all possible states. However, this global approach does not scale well to continuous domains, whose states cannot be enumerated; worse yet, the volume of a continuous state space (i.e., the "number" of states) grows exponentially with each added degree of freedom, a phenomenon known as the *curse of dimensionality*. Therefore, finding a globally-optimal policy is prohibitively hard in high-dimensional, nonlinear continuous models in the general case.

In light of this observation, my dissertation develops techniques for identifying *locally optimal* behavior, by focusing on optimizing a single trajectory. I argue that this compromise is not only computationally imperative, but that it is also stands in agreement with the way motor behavior develops in the natural world, whether through phylogeny (where genetic variations occur locally and independently) or ontogeny (where radically novel ways of performing a certain task are rarely introduced, if ever).

1.1.2 Physics modeling

The algorithms presented in this dissertation are *model-based* — the behaving agent has access to a model of the world's physics, and this model is used to identify optimal behavior. In this class of algorithms, simulating the dynamics is the most computationally-intensive component of the optimization. As such, an efficient way to compute the dynamics is crucial to enable effective optimization. This dissertation makes use of MuJoCo, a new software tool developed by Todorov [2011a], which is described in more detail in section 2.4.1.

In order to study terrestrial locomotion, the reaction forces between the ground and the feet must be modeled. However, these collisions have a discontinuous nature, which may cause standard algorithms of numerical optimization to fail. Various approaches have been proposed to simulate the reaction forces due to contact. Adding a spring-damper unit at every contact point yields a continuous-state, continuous-time approximation, but the parameters of the underlying equations are notoriously hard to tune, and the simulation of the resulting stiff differential equation can be numerically challenging. One alternative is to use a hybrid model, where the state space is augmented with discrete variables indicating the contact state, and the continuous dynamics change for every possible contact state. Hybrid modeling is commonly used in the control theory community for bipedal locomotion [e.g., Chevallereau et al., 2003], but standard algorithms of optimal control cannot tackle the hybrid state space. Another alternative works is to pose a Linear Complementarity Problem (LCP) at every time-step (section 2.4.2). LCP is widely used in the graphics community [e.g., Wang et al., 2009, Muico et al., 2009], but results in discontinuous dynamics, again disallowing the use standard continuous optimization.

The work presented here takes a different approach to contact modeling and employs two different algorithms for contact smoothing. This smoothing is grounded in the work of Tassa and Todorov [2010] (reviewed in section 2.4.2), who show that when considering LCP in the presence of stochastic dynamics, the expected next state is a smooth function of the current state. This leads to a model of smooth dynamics in a discrete-time, continuous-state system, thereby allowing the use of standard optimization tools. In addition to stochastic LCP, section 6.3.5 reviews a different smooth contact model which follows a similar scheme, and has the added benefit of allowing inverse computations.

1.1.3 Sparse success

In addition to the curse of dimensionality and the challenge of modeling contacts, another source of difficulty can be identified by considering the variability of performance across the space of all possible policies.

In the swimming task discussed in chapter 3, for almost all possible body configurations there exists an action that would decrease the cost. Furthermore, the viscous interaction between swimmer and medium ensures that spurious actions will be quickly dampened. These factors give rise to a situation where, if we are given some good policy, we can expect that a minor perturbation of this policy would result in a minor perturbation of the system's performance. This means that there are entire areas of policy space which yield acceptable performance. In short — you cannot fall when you swim.

This is not the case for bipedal locomotion. Here, the reaction forces provided by the environment are limited by friction cones, and would not dampen spurious actions; instead, balance may be lost when the center of mass moves outside the range of support by the feet, at which stage falling might be unavoidable. In such cases, minor perturbations to a good policy may have catastrophic results, and so the subset of policies which yield acceptable behavior is *sparse* in policy space. This gives rise to an optimization problem that is riddled with local minima.

This can be overcome through *shaping*. This concept is drawn from the vocabulary of behaviorist psychology, and it describes a process in which a solution to a difficult task is found by first solving a simple variant of this task, and then using that solution as a first guess for solving progressively harder variants of the task; the equivalent mathematical notion is a homotopy-continuation method [Allgower and Georg, 2003]. Every aspect of the task can be subject to shaping, from the physical properties of the domain to the parameters of the cost function (see section 2.5); all the legged behaviors presented in this dissertation are the result of shaping.

1.2 Topics not covered in this dissertation

Before delving into the body of this dissertation, I would like to briefly mention what this dissertation is *not* about.

First, I am not making the claim that the control algorithms described here have direct neural correlates. While I believe that a phenomenological study of motor behavior would prove useful in the quest for a mechanistic model of motor behavior, this would be a colossal undertaking, which falls outside the scope of this dissertation.

Generating novel motion patterns in a biomechanically accurate model could be of great scientific merit, and the algorithms presented here are designed with this goal in mind. However, careful tuning would be required to make the model a valid representation of reality, and therefore biomechanical realism is left for future work.

Policy search methods are a popular approach to designing motor behavior in high-dimensional systems [Theodorou et al., 2010, Wang et al., 2009, Yin et al., 2007, Riedmiller et al., 2007]. However, these methods require the parameterization of the policy to be carefully designed in advance. This approach can be effective in many particular cases, since the parameterization of the policy allows the designer to express prior knowledge on the system and the expected behavior. Yet, in this dissertation I try to take a more general approach, describing algorithms that can work even in the absence of prior expectations on the optimal behavior.

The unobservability of the system's true state is an important aspect of real-world behavior that is also not considered here. When physical objects move in space, their dynamic properties (position and velocity) are often unknown, and must be inferred from a limited set of noisy observations, whether visual, somatosensory or electronic. The algorithms I discuss here assume that the state is fully known, and leave state estimation as an independent problem. This separation can be justified by the principle of *certainty equivalence*, which guarantees that the most likely state (as identified by an independent estimation process) can reliably serve for feedback control. In order to maintain a coherent narrative, I do not include in this dissertation my studies on control in partially observable domains; the interested reader may find more information in the work of Erez and Smart [2010].

Finally, while the systems studied in this dissertation are stochastic, I am not employing risk-sensitive and adversarial techniques, which may offer an additional level of robustness in some cases; for further reading, the interested reader is referred to Erez and Smart [2009].

1.3 Outline

The remainder of this document is organized into 6 chapters.

- Chapter 2 reviews the mathematical theory and terminology that underpins this work. I define the various mathematical terms and the optimization criteria used throughout the dissertation, and review the relevant aspects of physics modeling. I conclude with a discussion about computational shaping.

- Chapter 3 describes our experiments with Differential Dynamic Programming (DDP), a sequential algorithm for trajectory optimization. DDP is put to use in the control of a robotic swimmer in two different settings: first, a library of optimal trajectories is constructed offline, and a locally-linear feedback controller is employed to control the swimmer simulation in real time. Second, DDP is used online for Model-Predictive Control (MPC). This chapter is based on research that is presented by Tassa, Erez, and Smart [2008] and Tassa, Erez, and Todorov [2011b].

- Chapter 4 presents a Bayesian approach to trajectory optimization, casting the problem in terms of likelihood estimation. This leads to simultaneous optimization of an entire trajectory, and allows for the optimization of limit cycles, as well as the construction of locally-linear feedback controller around the optimal trajectory. The effectiveness of this approach is demonstrated by generating a stable forward simulation of planar walking and running. This chapter is based on the work presented by Tassa, Erez, and Todorov [2011a].

- Chapter 5 presents an algorithm called Infinite-Horizon Model-Predictive Control (IHMPC), which combines offline limit-cycle optimization (using Bayesian estimation) and online trajectory optimization (using DDP). This chapter shows how the approximation of a quadratic value function around an optimized limit cycle can be employed online for augmenting the effective planning horizon of MPC. IHMPC is demonstrated by controlling a planar hopping task; we derive a *complete* solution to the hopping domain, in the sense that the hopper can effectively recover from any perturbation. As further evidence of robustness, I present experiments where intentional modeling errors are introduced, and robust behavior is maintained despite the discrepancy between planning and simulation. This chapter is based on work that is presented by Erez, Tassa, and Todorov [2011].

- Chapter 6 presents an alternative algorithm for simultaneous trajectory optimization via inverse dynamics. By relying on the invertability of the convex contact model described therein, this algorithm allows for rapid and efficient limit cycle optimization, as demonstrated by the design of a running gait for a 3D humanoid. This chapter is based on work that is presented by Erez and Todorov [2011].

- Finally, chapter 7 concludes with a summary of the dissertation, and offers an array of future research directions that extend the presented work.

Chapter 2

Background

This chapter provides the mathematical grounding for the algorithms described throughout the dissertation. We start by making the following definitions (for more in-depth discussion, the reader may refer to Sutton and Barto [1998] or Stengel [1994]):

State: The systems studied here occupy a unique state $\mathbf{x}$ at any given time. In principle, the state-space may be discrete or continuous, finite or infinite; here we focus on the case where the state space is continuous and n-dimensional: $\mathbf{x} \in \mathbb{R}^n$. Often the state is a concatenation of generalized coordinates $\mathbf{q}$ (usually representing a Cartesian position and joint angles) and the associated velocities vector $\dot{\mathbf{q}}$ (see section 6.3.2).

Time: We focus on fixed-step integration of continuous equations of motion (section 2.4.1). As a general rule, time index is denoted as superscript $\mathbf{x}^k$. In the handful of cases where superscript is needed to denote something different (e.g. matrix inversion), the time index is relocated to a subscript (e.g., Σ_k^{-1}).

Action/Control signal: The *agency* of the behaving agent is realized by the choice of an action $\mathbf{u}^k$ at every time step, which affects the next state. In this context, we focus on the continuous, m-dimensional control signal $\mathbf{u} \in \mathbb{R}^m$, often representing joint torques.

Dynamics: The evolution of the system's state over time is a function of the current state and action. In the context of discrete-time systems, the forward dynamics are described by a time-stepping function $\mathbf{f}$, specifying the next state $\mathbf{x}'$ as a function of the current state and control:

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{u}). \tag{2.1}$$

Stochasticity: We may also consider the case where the state dynamics is subject to a random perturbation:

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{u}) + \xi,$$

where $\xi \sim \mathcal{N}(\mathbf{0}, \Sigma)$ is an n-dimensional random variable whose distribution is assumed to be multivariate normal with mean zero and a constant covariance matrix Σ.

Trajectory: Given an *initial state* $\mathbf{x}^1$ and a *control sequence* $\mathbf{U} = \mathbf{u}^{1:N-1}$, we can compute the resulting *trajectory* by integrating the dynamics forward in time from $\mathbf{x}^1$ through $N-1$ repeated application of the dynamics (2.1), giving rise to the trajectory $\mathbf{X} = \mathbf{x}^{1:N}$.

Policy: the agent's behavior is defined in terms of a plan of action. We focus on the case of a deterministic policy, mapping every state to a particular action: $\mathbf{u} = \pi(\mathbf{x})$. The conjunction of a policy and an initial state serve to define a trajectory according to $\mathbf{x}^{k+1} = \mathbf{f}\big(\mathbf{x}^k, \pi(\mathbf{x}^k)\big)$.

2.1 Optimality Criteria

The notion of optimal behavior depends on the definition of a cost function $\ell(\mathbf{x}, \mathbf{u})$, specifying the penalty the agent incurs from every state-action pair. In addition, a terminal cost function $\ell_N(\mathbf{x})$ may be evaluated at the last state of a trajectory.

For a given policy $\pi(\mathbf{x})$, we define the scalar *value* function (or the *cost-to-go* function) $V^\pi(\mathbf{x})$, which denotes the future cumulative cost the agent expects to collect when starting at state $\mathbf{x}$ and acting according to π.

Given a dynamical system and a cost function, the *optimal* behavior is realized by the policy π^*, which prescribes at every state the action that would *minimize* future costs. The literature considers several alternative definitions for this cumulative cost, which are applicable to different circumstances, and may yield different behaviors. Here we present *finite-horizon* and *infinite-horizon average cost*; finite-horizon optimization is used in chapter 3, and also in chapter 5 as part of the IHMPC algorithm. Infinite horizon average cost is used in chapters 4, 5 and 6. For other criteria, such as discounted infinite-horizon or first-exit, the reader may refer to Sutton and Barto [1998].

2.1.1 Finite-horizon optimality

In this case, we seek the policy which minimizes the N-step simple cumulative cost. Given a state $\mathbf{x}^1$, the dynamics are integrated for $N-1$ steps, and the *value* of $\mathbf{x}^1$ is simply the sum of all costs along this trajectory:

$$V_1^{\pi}(\mathbf{x}^1) = \sum_{k=1}^{N-1} \ell\left(\mathbf{x}^k, \pi(\mathbf{x}^k)\right) + \ell^N(\mathbf{x}^N). \tag{2.2}$$

Note that the value is time-indexed, because the planning horizon is different for every state along the trajectory: while the first state sees the full N-step planning horizon, state k sees only $(N-k)$steps ahead, which may lead to myopic behavior.

We can define the value recursively:

$$V_k^{\pi}(\mathbf{x}) = \ell\big(\mathbf{x}, \pi\left(\mathbf{x}\right)\big) + V_{k+1}^{\pi}\big(\mathbf{f}(\mathbf{x}, \pi(\mathbf{x})\big), \tag{2.3}$$

with the boundary condition $V^N(\mathbf{x}) = \ell^N(\mathbf{x})$.

The *optimal* value function $V^*(\mathbf{x})$ describes the minimal cost-to-go from any state:

$$V_k^*(\mathbf{x}) = \min_{\mathbf{u}}\Big[\ell(\mathbf{x}, \mathbf{u}) + V_{k+1}^*\big(\mathbf{f}(\mathbf{x}, \mathbf{u})\big)\Big], \tag{2.4}$$

which is a Bellman equation (see section 2.2). By writing the derivative with respect to $\mathbf{u}$ and equating to zero, we recover the optimal policy: $\mathbf{u}^* = -\ell_{\mathbf{uu}}^{-1}(\ell_{\mathbf{u}} + \mathbf{f}_{\mathbf{u}} V_{\mathbf{x}}^*)$.

We define the *Hamiltonian*, also known as *Q-function* [Watkins, 1989], over state-action pairs:

$$Q_k^{\pi}(\mathbf{x}, \mathbf{u}) = \ell(\mathbf{x}, \mathbf{u}) + V_{k+1}^{\pi}\big(\mathbf{f}(\mathbf{x}, \mathbf{u})\big), \tag{2.5}$$

which considers the current action as a free variable, and follows the designated policy thereafter. The Bellman equation can be written in terms of the Q-function:

$$V^*(\mathbf{x}) = \min_{\mathbf{u}} Q^*(\mathbf{x}, \mathbf{u}). \tag{2.6}$$

2.1.2 Infinite-Horizon Average Cost

In this case, the optimal policy π^* minimizes the average cost along the trajectory in the limit of infinitely-long horizon:

$$c^* = \min_{\pi} \lim_{\mathfrak{N} \to \infty} \frac{1}{\mathfrak{N}} \sum_{k=1}^{\mathfrak{N}} \ell\Big(\mathbf{x}^k, \pi\big(\mathbf{x}^k\big)\Big). \tag{2.7}$$

Given a policy, the value function of the infinite-horizon problem is defined as the total deviation of the future cumulative cost from the average:

$$\tilde{V}^{\pi}\big(\mathbf{x}^1\big) = \lim_{\mathfrak{N} \to \infty} \left[\sum_{k=1}^{\mathfrak{N}} \ell\Big(\mathbf{x}^k, \pi\big(\mathbf{x}^k\big)\Big) - \mathfrak{N} c^{\pi} \right]. \tag{2.8}$$

In this case, the recursive definition of the optimal value function is not time dependent:

$$c + \tilde{V}^*(\mathbf{x}) = \min_{\mathbf{u}} \Big[\ell(\mathbf{x}, \mathbf{u}) + \tilde{V}^*\big(\mathbf{f}(\mathbf{x}, \mathbf{u})\big) \Big]. \tag{2.9}$$

A terminal cost is inapplicable in this case (since the trajectory never ends). This infinite sum can be difficult to compute in the general case; however, when the optimal behavior results in a limit cycle, the infinite-horizon average cost is simply the average cost along the closed trajectory.

2.1.3 Comparing optimality criteria

The choice of an optimality criterion has significant implications on the required computational procedure and the resulting behavior. The finite-horizon criterion is easy to evaluate, but as mentioned above, the optimal policy may exhibit myopic behavior: towards the end of the trajectory, the planning horizon is limited, and this reduces the motivation to take actions of long-term benefit, resulting in greedier optimal behavior (see figure 3.2). The optimization criterion of infinite-horizon average-cost is immune to the myopia that plagues the finite-horizon formulation. In the general case, it is difficult to compute, but in this dissertation the infinite-horizon criterion is only used in the context of limit-cycle optimization, where it is reduced to minimizing the total cost along the closed trajectory.

2.2 Value-Function Approximation

The finite-horizon domain has the key property of *optimal substructure*: given the current state, the optimal action (in the finite-horizon sense) is independent of any past states or actions, and the action taken at this time step can only affect costs (and states) in the future. Therefore, for a given state $\mathbf{x}^k$ (and horizon $N-k$), the problem of finding the optimal behavior in the remaining time is well-defined, and can be considered independently of the history which led to this state.

This identifies the problem of finding the optimal behavior as amenable to *dynamic programming*. In domains with a discrete and finite state space, the finite-horizon optimal policy may be found by tabulating all state-action pairs, and filling in the Q-values for every pair. If the domain has terminal states, this can be done through a single backwards-pass from the terminal states (where the cost-to-go is simply the immediate cost) through the transition graph, integrating the optimal cost-to-go backwards:

$$Q^*(\mathbf{x},\mathbf{u}) = \ell(\mathbf{x},\mathbf{u}) + \min_{\mathbf{u}'} Q^*\big(\mathbf{f}(\mathbf{x},\mathbf{u}),\mathbf{u}'\big).$$

If there are no terminal states, we may still solve the domain by creating a tabular approximation $\tilde{Q}$ for every state-action pair, and update it using the Q-learning rule:

$$\tilde{Q}(\mathbf{x},\mathbf{u}) \leftarrow (1-\alpha)\tilde{Q}(\mathbf{x},\mathbf{u}) + \alpha\Big[\ell(\mathbf{x},\mathbf{u}) + \min_{\mathbf{u}'} \tilde{Q}\big(\mathbf{f}(\mathbf{x},\mathbf{u}),\mathbf{u}'\big)\Big], \tag{2.10}$$

where $0 < \alpha \leq 1$ is the learning rate. For a proper choice of α, this technique is guaranteed to converge to the optimal Q-function [Watkins, 1989].

In order to apply this method to a continuous domain, we can discretize the state- and action- spaces, and build the Q-table over these finite sets. In the limit of infinitely-fine discretization, the tabular approach is guaranteed to converge to the optimal policy [Sutton and Barto, 1998]. However, note that this is a *global* approach, as it finds the optimal policy for every element of state space. Naturally, the complexity of a tabular approach scales linearly with the size of the table. Yet, the number of elements in such a table grows exponentially with the number of state dimensions. Therefore, such a discretization-based table becomes too big to manage in high-dimensional domains.

One way to alleviate this exponential complexity is through the application of adaptive mesh techniques [e.g., Munos and Moore, 1999]. However, the fundamental difficulty remains, as the volume of state space (and hence, for example, the number of nodes to prune) still grows exponentially with dimensionality.

2.3 Local methods

Since the globally-optimal policy is computationally intractable in the general case, the only viable alternative is to focus on techniques that identify locally-optimal policies. Local algorithms of optimal control fall into two broad categories [Diehl et al., 2009]: *Sequential* and *Simultaneous*, named according to the way the search space is represented.

2.3.1 Sequential representation

In the finite-horizon case, we can use a sequential approach, and optimize only the action sequence $\mathbf{U}$. Given a fixed initial state $\mathbf{x}^1$, we can measure the total cost of any given action sequence. For the *locally-optimal* $\mathbf{U}^*$, any perturbation of the action sequence $\delta\mathbf{U}$ would result in a higher cumulative cost. It is important to note that this framework cannot represent (nor optimize) limit cycles, because only specific action sequences result in an identity between the initial state and the terminal state (i.e., closing the loop). Sequential representation underpins the DDP algorithm (described in chapter 3, and used again in chapter 5).

2.3.2 Simultaneous Representation

As an alternative, we may apply a simultaneous approach by explicitly representing the entire trajectory $\mathbf{X}$ (and perhaps also the action sequence $\mathbf{U}$). This leads to the formulation of an optimization problem of the form:

$$\min_{\mathbf{X},\mathbf{U}} \sum_{k=1}^{N} \ell\big(\mathbf{x}^k, \mathbf{u}^k\big) \quad \text{s.t. } \forall k : \psi\big(\mathbf{x}^k, \mathbf{u}^k, \mathbf{x}^{k+1}\big) = 0,$$

where the function

$$\psi(\mathbf{x}, \mathbf{u}, \mathbf{x}') = \mathbf{x}' - \mathbf{f}(\mathbf{x}, \mathbf{u})$$

imposes dynamical consistency between every consecutive pair of states, treating equation 2.1 as a constraint. In order to apply this approach to a limit cycle of period N, we define $N + 1 \equiv 1$ so that the last state $\mathbf{x}^N$ is followed by the first state $\mathbf{x}^1$.

Local methods of optimization that use a simultaneous representation include multiple shooting [Bock and Plitt, 1984] and space-time constraints [Witkin and Kass, 1988]. This approach has also been applied to gait design [Wampler and Popovic, 2009, Mombaur et al., 2010, Wu and Popović, 2010]. Simultaneous representation underpins the algorithms discussed in chapters 4, 5 and 6.

2.3.3 Comparing local methods

So which representation is better? The answer, unsurprisingly, depends on the desired goal. The sequential approach often employs fewer variables, and the resulting trajectories are guaranteed to be dynamically consistent. Furthermore, dynamic programming may be applied in this context, leading to efficient computation. However, sequential algorithms only work with the finite-horizon optimality criterion, and are therefore inherently inadequate for periodic tasks where a ground contact has to be planned in advance. The simultaneous approach allows the optimization to consider infeasible trajectories; this may prevent getting stuck in some local minima, but requires machinery for enforcing dynamical consistency, and might result in slower convergence.

2.4 Modeling the dynamics

As stated in section 1.1.2, all the algorithms presented here rely on efficient computation of the dynamics. This involves two components — simulating the continuous multibody dynamics, and simulating the collisional interaction between feet and ground.

2.4.1 Multibody simulation

For the mechanical systems considered here, the time-stepping dynamics equation can be constructed using semi-implicit Euler integration of the equations of motion:

$$\dot{\mathbf{q}}' = \dot{\mathbf{q}} + hM^{-1}(\mathbf{r} + B\mathbf{u}), \tag{2.11a}$$

$$\mathbf{q}' = \mathbf{q} + h\dot{\mathbf{q}}', \tag{2.11b}$$

where h is the time-step, $\mathbf{q}$ is the configuration, $M_k = M(\mathbf{q}^k)$ is the mass-matrix, $\mathbf{r}^k = \mathbf{r}(\mathbf{q}^k, \dot{\mathbf{q}}^k)$ is the vector of total Coriolis, centripetal and other intrinsic forces, and B is the $n \times m$ matrix determining which degrees of freedom are actuated (with $m < n$ for underactuation). The first equation is an Euler integration of the velocity following Newton's second equation $\ddot{\mathbf{q}} = M^{-1}F$; the second equation is an Euler integration of the position using the *new* velocity.[3]

This dissertation makes use of several physics engines that are written in Matlab. However, chapters 3, 5 and 6 make use of MuJoCo, a new physics engine created by

[3] In *explicit* Euler, we use the current velocity to integrate the current position; in *semi-implicit* Euler, the new velocity is computed according to the current position, but the position is updated using the new velocity; in *implicit* Euler, the new position determines the new velocity, which determines the new position.

Todorov [2011a]. MuJoCo is extremely fast for multibody computation due to some unique properties: it is written in C with no dependency on external libraries; all the memory space in use is pre-allocated; and, most importantly, it is thread-safe. This allows us to make maximal use of multi-core processors, and yields substantial performance gain over most other simulation software.

The heart of the multibody physics computation is the calculation of the mass matrix $M(\mathbf{q})$ and the intrinsic forces $\mathbf{r}(\mathbf{q}, \dot{\mathbf{q}})$; for more information on how these are computed in MuJoCo, see Todorov [2011a]. Most significantly, the complexity of a single call to the dynamics engine is dominated by the factorization and back-substitution of M at $\sim O(n^3)$ (where n is the dimension of $\mathbf{x}$). The algorithms presented here make extensive use of the derivatives of the dynamics; we estimate these derivatives through finite-differencing. Note that the mass matrix is independent of the velocity; this can be used to shortcut some of the finite-difference computations.

2.4.2 Stochastic Linear Complementarity

The simulation of ground reaction forces is an open research question; as discussed in section 1.1.2, we are interested in contact smoothing methods, since smooth dynamics enable the use of standard optimization tools.

A contact model is expected to fulfil several conditions. Most importantly, the contact forces are expected to fall within the friction cone; the *linear complementarity* property [Cottle, 1992] states an additional requirement: a reaction force may only be applied between objects that are in contact (prohibiting "action-at-a-distance"). This is a difficult constraint to enforce, because the next contact state depends on the reaction forces, which depend on the contact state. Modeling this relationship requires the use of the discontinuous operator $\min(a, b)$, which results in contact forces that are a non-smooth function of the state.

The theory of Stochastic Linear Complementarity Problems [Tassa and Todorov, 2010] shows that in the context of a stochastic dynamical system, the *expected* dynamic transition is a smooth function of the state. Here we only repeat the main result: instead of finding the contact force by solving a linear complementarity problem (which can be discontinuous in the inputs), the expected contact force in a stochastic system can be found by minimizing the SLCP residual, which is a smooth function of the state. This implies that the minimizing solution (i.e., the reaction force) is a smooth function of the state.

As mentioned above, the algorithms presented here rely on computing the derivatives of the dynamics function through finite-differencing, which involve re-computing the dynamics at a "cloud" of points around the current state. In the context of SLCP, a direct computation of the dynamics at such a cloud of very close states involves re-minimization of the SLCP residual; by warm-starting the SLCP optimization with the previous solution, this can be made computationally efficient. However, finite-differencing the output of an optimization function is numerically tricky, since numerical optimization involves setting thresholds and termination criteria, and therefore the result cannot be expected to be accurate in the bits that are below the threshold. Yet, since all the states in the cloud are very similar, the optimized reaction forces identify on the leading bits. Therefore, full re-optimization may lead to numerical inaccuracies and underflow artefacts.

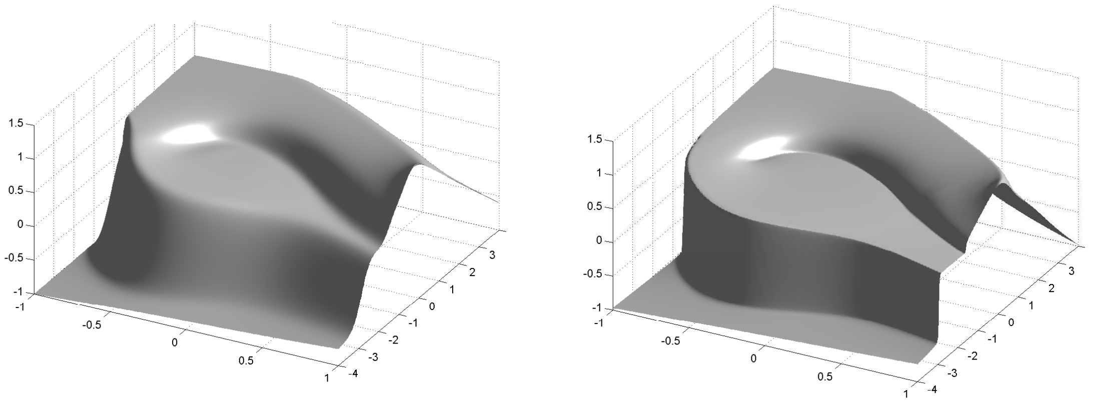
This numerical challenge motivates an alternative approach for finite-differencing the contact forces. Instead of performing full optimization for every point in the cloud, the gradient is computed in two steps: first, we find the contact force at the current state (the "center" of the cloud) by minimizing the SLCP residual as before; then, for every state in the cloud we take a fixed number of optimization steps, starting from the contact force at the center of the cloud. This ensures that all the points in the cloud are subject to the same numerical conditioning, and yields smooth, accurate derivatives.

2.5 Shaping

As discussed in section 1.1.3, the optimization problems formulated here often exhibit many local minima; in such a case, a local search is prone to failure, in the sense that the resulting local optimum might be prohibitively sub-optimal. In order to tackle this issue, Erez and Smart [2008] discuss the different ways continuation-homotopy methods may be used to facilitate the solution of optimal control problems.

The key observation is that a good initialization of the optimization process is critical in non-convex optimization problems of the type encountered here. This is the cornerstone for the entire endeavor of shaping: the basic premise is to first solve an easy variant, and then change the task and use the previous solution as an initial guess. Thus, one tracks a homotopy path through the space of all possible domains, finding a solution to a sequence of problems, and eventually producing a solution to the hard variant. While such an approach is inherently limited to finding local optima, it can enable the solution otherwise prohibitively-hard problems.

The standard shaping protocol involves a modification of the cost function, first rewarding behavior that only roughly resembles the desired motion pattern, and gradually making the cost less forgiving. For example, the runner presented in section 4.5.4 is optimized by first solving for a walking behavior (section 4.5.2), and then modifying the desired velocity. In some domains, the level of difficulty is related to some specific parameters of the model (for example, staying upright is probably easier in lower gravity, or when your center of mass is closer to the ground). The identification of such bottleneck parameters can allow us to manipulate the difficulty of the domain through changing the physics. As a specific example of morphological shaping, the level of noise assumed to affect the dynamics can greatly affect the expected solution, as the expected presence of noise leads to smoother and more regular problems (see figure 2.1). In particular, assuming more noise leads the SLCP algorithm (section 2.4.2) to generate smoother ground reaction forces. We used this form of shaping in the experiments described in chapters 4 and 5.



(a) The smooth value function ($\Sigma = 0.02\mathbf{I}$) (b) The deterministic value function ($\Sigma = \mathbf{0}$)

Figure 2.1: Approximation of the optimal value function for the car-on-the-hill problem when the domain is **(a)** stochastic and **(b)** deterministic. Notice how the sharp discontinuities in the deterministic case are smoothed out in the stochastic case. These results are from Tassa and Erez [2007].

Chapter 3

Differential Dynamic Programming

3.1 Introduction

This chapter describes our experiments with Differential Dynamic Programming (DDP), a sequential algorithm (section 2.3.1) for optimizing behavior under the finite-horizon criterion (section 2.1.1). DDP is a second-order algorithm first described by Jacobson and Mayne [1970], which makes use of the optimal substructure property (section 2.2), allowing for efficient computation and rapid convergence. In order to overcome the limitations of finite-horizon optimization, we present a receding-horizon scheme (section 3.5.1).

This chapter describes how DDP can be used to optimize behavior in a dynamic environment, studying a domain of robotic swimming with 10 to 27 dimensions, which is described in section 3.6. We present two methods for deriving feedback controllers from DDP: first, we use an offline receding-horizon scheme to construct a locally-linear global feedback controller (section 3.5.1). We then apply the same receding-horizon principle in an online fashion for Model-Predictive Control (section 3.5.3). In both cases, the receding horizon structure gives rise to non-myopic behavior, generating an unlimited diversity of swimming behaviors on-the-fly in a variety of environmental conditions. Our results are best illustrated by a series of movies [4].

[4] Available at:
http://www.youtube.com/watch?v=3LagQ5_0Q0g
http://www.youtube.com/watch?v=H05TW-UfehQ
http://www.youtube.com/watch?v=m0PboYKQTCc

3.2 The basic algorithm

DDP finds the optimal N-step trajectory emanating from a fixed state $\mathbf{x}^1$. Every iteration of DDP is composed of two steps: first, the *backward pass* starts from a nominal trajectory, integrating a quadratic approximation of the value function (2.4), and deriving a modification to the policy along the way; subsequently, the *forward pass* applies this modification, hopefully tracing a better trajectory. This basic structure is shared with other algorithms, such as the ones presented by Bryson and Ho [1975] and Dunn and Bertsekas [1989]; for a comparison, see Mizutani and Dreyfus [2005].

3.2.1 The backward pass

The input to the backward pass (BP) is some nominal action sequence $\mathbf{U} = \{\mathbf{u}^k\}_{k=1}^{N-1}$, and the corresponding nominal trajectory $\mathbf{X} = \{\mathbf{x}^k\}_{k=1}^{N}$, as well as the total cumulative cost (2.2):

$$C(\mathbf{U}) \triangleq V(\mathbf{x}^1) = \sum_{k=1}^{N-1} \ell(\mathbf{x}^k, \mathbf{u}^k) + \ell^N(\mathbf{x}^N).$$

The goal of the BP is to find a modification of the action sequence that would lead to a trajectory with a lower cumulative cost. The BP constructs a quadratic approximation of the value function around every time step:

$$V^k(\mathbf{x}^k + \delta\mathbf{x}) \approx V_0^k + \delta\mathbf{x}^\mathsf{T} V_\mathbf{x}^k + \delta\mathbf{x}^\mathsf{T} V_{\mathbf{xx}}^k \delta\mathbf{x}, \tag{3.1}$$

and uses it to compute the locally-optimal action sequence $\mathbf{U}^* = \mathbf{U} + \Delta^*\mathbf{U}$. As the name suggests, the backward pass builds these quadratic models by stepping backward along the nominal trajectory.

Every step of the BP derives a quadratic model $\{V_\mathbf{x}, V_{\mathbf{xx}}\}$ around $\mathbf{x}$ from the quadratic model $\{V'_\mathbf{x}, V'_{\mathbf{xx}}\}$ around the *next* state $\mathbf{x}'$. At every time-step, we construct a second-order Taylor expansion of the Q-function (2.5) around $\mathbf{x}$ and $\mathbf{u}$:

$$Q^k(\mathbf{x} + \delta\mathbf{x}, \mathbf{u} + \delta\mathbf{u}) \approx \frac{1}{2}[1 \ \delta\mathbf{x}^\mathsf{T} \ \delta\mathbf{u}^\mathsf{T}]^\mathsf{T} \begin{bmatrix} Q_0 & Q_\mathbf{x} & Q_\mathbf{u} \\ Q_\mathbf{x} & Q_{\mathbf{xx}} & Q_{\mathbf{xu}} \\ Q_\mathbf{u} & Q_{\mathbf{ux}} & Q_{\mathbf{uu}} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix}. \tag{3.2}$$

We can write the coefficients of (3.2) around the current state $\mathbf{x}$ in terms of the quadratic model around the next state $\mathbf{x}'$:

$$\begin{aligned} Q_0 &= \ell + V'_0, & Q_{\mathbf{xx}} &= \ell_{\mathbf{xx}} + F_{\mathbf{x}}^{k} V'_{\mathbf{xx}} F_{\mathbf{x}} + V'_{\mathbf{x}} F_{\mathbf{xx}}, \\ Q_{\mathbf{x}} &= \ell_{\mathbf{x}} + V'_{\mathbf{x}} F_{\mathbf{x}}, & Q_{\mathbf{uu}} &= \ell_{\mathbf{uu}} + F_{\mathbf{u}} V'_{\mathbf{xx}} F_{\mathbf{u}} + V'_{\mathbf{x}} F_{\mathbf{uu}}, \\ Q_{\mathbf{u}} &= \ell_{\mathbf{u}} + V'_{\mathbf{x}} F_{\mathbf{u}}, & Q_{\mathbf{xu}} &= \ell_{\mathbf{xu}} + F_{\mathbf{x}} V'_{\mathbf{xx}} F_{\mathbf{u}} + V'_{\mathbf{x}} F_{\mathbf{xu}}. \end{aligned} \tag{3.3}$$

Using this quadratic approximation, we can perform the minimization of (2.6) analytically, by taking the derivative of (3.2) with respect to $\delta\mathbf{u}$ and equating to zero:

$$0 = Q_{\mathbf{u}} + \delta\mathbf{u}^{\mathsf{T}} Q_{\mathbf{uu}} + \delta\mathbf{x}^{\mathsf{T}} Q_{\mathbf{xu}}, \tag{3.4}$$

yielding a closed-form expression for the minimizing control:

$$\Delta^*\mathbf{u} = -Q_{\mathbf{uu}}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\delta\mathbf{x}). \tag{3.5}$$

Note that this expression has two terms, the first being a modification to the open-loop policy, and the second being a feedback term describing how the optimal control $\mathbf{u}^*$ should adapt to changes in the state. Therefore, we can rewrite it as:

$$\Delta^*\mathbf{u} = l + L\delta\mathbf{x}, \tag{3.6}$$

with $l = -Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}}$ and $L = -Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}$.

Using the quadratic model of Q, we can approximate the expected change in the value due to switching from the nominal control $\mathbf{u}$ to the approximately-optimal one $\mathbf{u}^* = \mathbf{u} + \Delta^*\mathbf{u}$:

$$V_0 = Q_0 - \frac{1}{2} Q_{\mathbf{u}}{}^{\mathsf{T}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}}.$$

By taking the first and second derivatives of this equation with respect to $\mathbf{x}$, we obtain the following equations for a quadratic model around the current state:

$$\begin{aligned} V_{\mathbf{x}} &= Q_{\mathbf{x}} - Q_{\mathbf{xu}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}}, \\ V_{\mathbf{xx}} &= Q_{\mathbf{xx}} - Q_{\mathbf{xu}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}. \end{aligned} \tag{3.7}$$

This quadratic model may now be used in equation 3.1 to take the next step back. The BP is initialized by setting $V_{\mathbf{x}}^N = \ell_{\mathbf{x}}^N, V_{\mathbf{xx}}^N = \ell_{\mathbf{xx}}^N$, and steps backward until the quadratic model around $\mathbf{x}^1$ is computed.

3.2.2 The Forward Pass

The second phase of every DDP iteration involves integrating a new trajectory using the optimal control modifications calculated during the backward pass. Starting from the fixed initial state $\check{\mathbf{x}}^1 = \mathbf{x}^1$, we generate a new trajectory $\check{\mathbf{X}} = \check{\mathbf{x}}^{1:N}$ and a new control sequence $\check{\mathbf{U}} = \check{\mathbf{u}}^{1:N-1}$ by applying the following equations $N-1$ times:

$$\check{\mathbf{u}}^k = \mathbf{u}^k + l^k + L^k(\check{\mathbf{x}}^k - \mathbf{x}^k), \tag{3.8a}$$

$$\check{\mathbf{x}}^{k+1} = \mathbf{f}(\check{\mathbf{x}}^k, \check{\mathbf{u}}^k). \tag{3.8b}$$

Once the new trajectory is complete, we compute the sum of costs along the new trajectory:

$$C(\check{\mathbf{U}}) = \sum_{k=1}^{N} \ell(\check{\mathbf{x}}^k, \check{\mathbf{u}}^k) + \ell^N(\check{\mathbf{x}}^N).$$

If $C(\check{\mathbf{U}}) \leq C(\mathbf{U})$, i.e., the total cost of the new trajectory is lower than the total cost of the current one, we accept the iteration, and begin a new backward pass along the new nominal trajectory.

When the change in cost $C(\check{\mathbf{U}}) - C(\mathbf{U})$ and/or the magnitude of the open loop modification $\sum \left|\left|l^k\right|\right|$ reach below some threshold, DDP halts, and the nominal trajectory is pronounced locally-optimal (see section 3.3.1).

3.3 Analysis

3.3.1 Convergence Properties

The convergence properties of DDP can be compared with a naive approach, treating $\mathbf{U}$ as one single multidimensional variable, and taking Newton steps with an exact

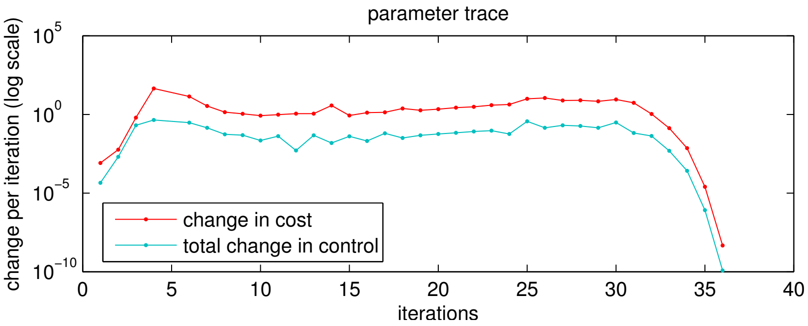


Figure 3.1: An illustration of the quadratic convergence of DDP (drawn from the simulation of a 4D cart-pole problem). After about 30 iterations, both the change in cost and the average change in the control sequence suddenly drop (note the logarithmic scale).

$Nm \times Nm$ Hessian. As shown by Liao and Shoemaker [1992], DDP achieves similar or better performance, due to the incorporation of the feedback term L. As figure 3.1 shows, the size of the improvement plummets super-exponentially near the minimum, indicating that the Hessian is exact. In practice, this quadratic convergence provides a natural stopping criterion, as it is evident when DDP has found a local minimum.

3.3.2 Computational considerations

Given a quadratic model of $\mathbf{f}$ and ℓ (whose terms are used in equation 3.3), the computational complexity of one iteration of DDP itself is $O(Nm^3)$, as it is dominated by the cost of inverting $Q_{\mathbf{uu}}$ at every step of the backward pass. However, the most significant part of the computational effort in practice is devoted to the finite-differencing approximation of the quadratic model of $\mathbf{f}$. Evaluating the first and second derivatives of the dynamics function through finite-differencing requires $O(n^2)$ calls to the dynamics function (where n is the dimension of the state-space, as in chapter 2) for each of the N states, where every dynamics computation is approximately $O(n^3)$ (section 2.4.1), leading to a total complexity of $O(Nn^5)$.

To see if we can reduce the burden of computing second derivatives, we may consider equation 3.3 again. Since mechanical systems are control-affine, $\mathbf{f}_{\mathbf{uu}} = 0$ identically, and does not need to be computed. On the other hand, $\mathbf{f}_{\mathbf{xx}}$ is the largest data structure (an n^3 tensor) and the most expensive to compute. The iLQG algorithm described by Todorov and Li [2005] is a DDP variant that relies on the assumption that $\mathbf{f_{xx}} = 0$. Here we make the following observation: if first derivatives $\mathbf{f_x}$ are computed with central differences, then the computation of the diagonal of $\mathbf{f_{xx}}$ is free. This reduces the complexity of computing the finite-difference gradients to $O(Nn^4)$, still the dominating term, but significantly less expensive. In the experiments below we explore the tradeoff of using full versus diagonal $\mathbf{f_{xx}}$.

Finally, a significant speedup can be achieved by parallelizing the finite-difference computation across several cores on a modern multi-core processor. Here, we used MuJoCo, a new physics simulator created expressly for this purpose, as described in section 2.4.1.

In practice, when using MuJoCo, the computational effort for computing the derivatives of the dynamics (in parallel) takes about 90% of CPU time. The backward pass itself constitutes about 10% of the computational load, and the forward pass takes only an order of 1% of the computational resources. With a Matlab-based physics engine the computational effort for computing the derivatives of the dynamics is even bigger, taking more than 99.9% percent of the computation time.

3.4 Modifications to the basic algorithm

3.4.1 Value divergence and regularization

In equation 3.5 we identify the optimal modification l to the control signal $\mathbf{u}$ according to the quadratic model (3.2). However, this computation is only valid if the matrix $Q_{\mathbf{uu}}$ is positive-definite. When this is not the case, the quadratic model is suggesting that there is a direction of $\delta\mathbf{u}$ where the value is unbounded from below, and that the greater the change in that direction, the better. This *divergence* of the value function plagues the calculations of the backward pass.

In order to complete the backward pass, we must ensure that $Q_{\mathbf{uu}}$ is positive-definite, and therefore l, as computed in equation 3.5, is indeed a descent direction. The common regulation scheme presented in the literature is to add some scaled identity matrix $\lambda\mathbf{I}$ to $Q_{\mathbf{uu}}$, so that $l = -(Q_{\mathbf{uu}} + \lambda\mathbf{I})^{-1}Q_{\mathbf{u}}$. This has the effect of increasing all of $Q_{\mathbf{uu}}$'s eigenvalues by λ; for a large-enough value of λ, this ensures that $Q_{\mathbf{uu}}$ is positive-definite.

Recall that according to equation 3.3, $Q_{\mathbf{uu}}$ is additive in $\ell_{\mathbf{uu}}$. If we use a modified cost function $\ell^{\lambda}(\mathbf{x},\mathbf{u}) = \ell(\mathbf{x},\mathbf{u}) + \frac{1}{2}\lambda\left|\left|\mathbf{u} - \mathbf{u}^{k}\right|\right|^{2}$, we would get $\ell^{\lambda}_{\mathbf{uu}} = \ell_{\mathbf{uu}} + \lambda\mathbf{I}$. Therefore, adding $\lambda\mathbf{I}$ to $Q_{\mathbf{uu}}$ is equivalent to adding a cost term which incurs a quadratic penalty for deviations from the current nominal control $\mathbf{U}$.

This interpretation motivates an alternative regularization method: instead of adding a cost term on deviations from $\mathbf{u}^{k}$, we introduce a similar cost term that incurs a quadratic penalty for deviating from the current state $\mathbf{x}^{k}$. The modified cost function $\ell^{\lambda}(\mathbf{x},\mathbf{u}) = \ell(\mathbf{x},\mathbf{u}) + \frac{1}{2}\lambda\left|\left|\mathbf{x} - \mathbf{x}^{k}\right|\right|^{2}$ yields $\ell^{\lambda}_{\mathbf{xx}} = \ell_{\mathbf{xx}} + \lambda\mathbf{I}$, which additively modifies $Q_{\mathbf{xx}}$, and therefore $V_{\mathbf{xx}}$, affecting $Q_{\mathbf{uu}}$ (3.3) at the next step of the backward pass:

$$\begin{aligned}\tilde{Q}_{\mathbf{uu}} &= \ell_{\mathbf{uu}} + F_{\mathbf{u}}(V'_{\mathbf{xx}} + \lambda\mathbf{I})F_{\mathbf{u}} + V'_{\mathbf{x}}F_{\mathbf{uu}},\\ \tilde{Q}_{\mathbf{xu}} &= \ell_{\mathbf{xu}} + F_{\mathbf{x}}(V'_{\mathbf{xx}} + \lambda\mathbf{I})F_{\mathbf{u}} + V'_{\mathbf{x}}F_{\mathbf{xu}},\end{aligned} \tag{3.9}$$

and using these modified matrices in (3.5) to compute l and L.

Since we are altering the quadratic model through regularization, we must modify equation 3.7:

$$V_0 = Q_0 + {Q_{\mathbf{u}}}^{\mathsf{T}} l + \frac{1}{2} l^{\mathsf{T}} Q_{\mathbf{uu}}^{-1} l, \tag{3.10a}$$

$$V_{\mathbf{x}} = Q_{\mathbf{x}} + L^{\mathsf{T}} Q_{\mathbf{u}} + {Q_{\mathbf{ux}}}^{\mathsf{T}} l + L^{\mathsf{T}} Q_{\mathbf{uu}}^{-1} l, \tag{3.10b}$$

$$V_{\mathbf{xx}} = Q_{\mathbf{xx}} + L^{\mathsf{T}} Q_{\mathbf{ux}} + {Q_{\mathbf{ux}}}^{\mathsf{T}} L + L^{\mathsf{T}} Q_{\mathbf{uu}}^{-1} L. \tag{3.10c}$$

In the limit of large λ, $\Delta^{*}u \approx \lambda^{-1}Q_{\mathbf{u}}$, and the Newton step in equation 3.5 is reduced to a first-order gradient descent with a small step size λ^{-1}; consequently, a larger value of λ leads to smaller steps and slower convergence. In practice, we decrease λ after every successful iteration, and increase it only when a non-positive-definite $\tilde{Q}_{\mathbf{uu}}$ is encountered (see section 3.4.3).

3.4.2 Line search

The forward pass of DDP, given by equations 3.8, relies on the local models measured around $\mathbf{x}^k$. Yet, the accumulation of the prior modifications $\Delta\mathbf{u}^{1:k-1}$ might lead $\check{\mathbf{x}}^k$ to be very different from $\mathbf{x}^k$, where the local models approximated around the nominal trajectory are no longer valid. At which case, $\mathbf{u}^k + l^k$ may well lead to an increase of the total cost, and divergence can occur. In order to regulate the forward pass, we introduce a line-search parameter $0 < \alpha \leq 1$ into equation 3.8a:

$$\check{\mathbf{u}}^k = \mathbf{u}^k + \alpha l^k + L^k(\check{\mathbf{x}}^k - \mathbf{x}^k). \tag{3.11}$$

α also affects the feedback term implicitly, since a smaller step leads to a smaller deviation from the original trajectory, and hence smaller feedback terms. In the limit of $\alpha = 0$, the new trajectory is identical to the old one, and the feedback term remains zero throughout.

Equation 3.10a allows us to estimate the expected total cost of the new trajectory by summing the cost difference terms,

$$\Delta V^k = V_0^k - Q_0^k = Q_{\mathbf{u}}^{k\mathsf{T}} l^k + \frac{1}{2} l^{k\mathsf{T}} Q_{\mathbf{uu}}^{-1} l^k,$$

along the trajectory. In order to account for α, we modify this equation:

$$\Delta V^k(\alpha) = \alpha l^{k\mathsf{T}} Q_{\mathbf{u}}^k + \frac{\alpha^2}{2} l^{k\mathsf{T}} Q_{\mathbf{uu}}^k l^k. \tag{3.12}$$

By saving the linear and quadratic terms separately, we obtain a quadratic model of the expected reduction as a function of α.

As advocated in Jacobson and Mayne [1970], we may use this prediction during line search. We compare the expected reduction $\Delta V(\alpha) = \sum_k \Delta V^k(\alpha)$ and the actual reduction, measured as the difference between the total cost of the nominal trajectory and the total cost of the new one:

$$z = \frac{[C(\mathbf{U}) - C(\check{\mathbf{U}})]}{\Delta V(\alpha)}.$$

We accept the iteration only if this ratio is bigger than some minimal threshold γ:

$$\gamma < z. \tag{3.13}$$

This is similar to the classic Armijo condition in optimization, but uses the more accurate quadratic reduction model (3.12). In the experiments described below we used $\gamma = 0.5$.

At every forward pass, we start with $\alpha = 1$. If this does not lead to improvement, we perform a simple halving backtrack $\alpha \leftarrow \alpha/2$ and try the forward pass again. If α falls below 2^{-10} and no improved trajectory is found, we increase λ and repeat the backward pass.

3.4.3 Parameter Schedules

The fast and accurate modification of the regularization parameter λ turns out to be important due to three conflicting requirements. If we are near the minimum, we want λ to quickly go to zero to enjoy the quadratic convergence. If the approximation diverges (a non-positive-definite $\widetilde{Q}_{\mathbf{uu}}$), we want λ to be big enough to prevent divergence, which can sometimes be very large. Finally, if we are in a regime where some $\lambda > 0$ is required, we would like to accurately set it to be as close as possible to the minimum value (though not smaller). Our solution is to use a log-quadratic modification schedule. Defining some minimal value $\lambda_{\min}$ (we use $\lambda_{\min} = 10^{-6}$) and a minimal modification factor Γ_0 (we use $\Gamma_0 = 2$), we adjust λ as follows:

$$\begin{aligned}
&\text{increase } \lambda: \\
&\qquad \Gamma \leftarrow \max(\Gamma_0, \Gamma \cdot \Gamma_0) \\
&\qquad \lambda \leftarrow \max(\lambda_{\min}, \lambda \cdot \Gamma) \\
&\text{decrease } \lambda: \\
&\qquad \Gamma \leftarrow \min(\tfrac{1}{\Gamma_0}, \tfrac{\Gamma}{\Gamma_0}) \\
&\qquad \lambda \leftarrow \begin{cases} \lambda \cdot \Gamma & \text{if } \lambda \cdot \Gamma > \lambda_{\min}, \\ 0 & \text{if } \lambda \cdot \Gamma < \lambda_{\min}. \end{cases}
\end{aligned}$$

Whenever λ increases or decreases consecutively across iterations, the size of the change grows geometrically. If increase and decrease alternate, the change remains small.

3.4.4 Algorithm Summary

0. Initialization: Given an initial state $\mathbf{x}^1$ and an initial control sequence $\mathbf{U} = \{\mathbf{u}^k\}_{k=1}^{N-1}$, integrate the dynamics to obtain an initial trajectory $\mathbf{X} = \{\mathbf{x}^k\}_{k=1}^{N}$. Measure the total cost $C(\mathbf{U}) = \sum_{k=1}^{N-1} \ell(\mathbf{x}^k, \mathbf{u}^k) + \ell^N(\mathbf{x}^N)$.

1. Dynamics: Given a nominal trajectory $(\mathbf{X}, \mathbf{U})$, compute the derivatives of ℓ and $\mathbf{f}$ in the RHS of equation 3.3. Alternatively, compute the dynamics for all collocation vectors around each state $\mathbf{x}^k$. This can be done in parallel for all k.

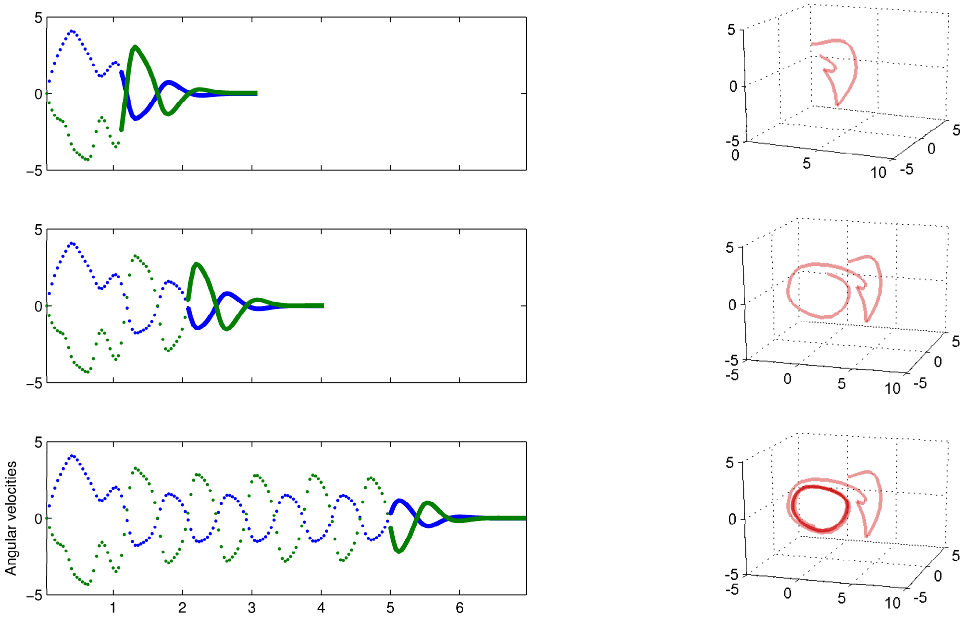
2. Backward pass: Iterate equations (3.3, 3.9, 3.10) for decreasing $k = N{-}1, \ldots 1$. If a non-positive-definite $\widetilde{Q}_{\mathbf{uu}}$ is encountered, increase λ and restart the backward pass. If successful and λ is not increased, decrease λ.

3. Forward pass: Set $\alpha = 1$. Iterate equations (3.11) and (3.8b) to compute a new nominal sequence. If the integration diverged or condition (3.13) is not met, decrease α and restart the forward pass.

4. Termination: If $\sum_k \left|\left|l^k\right|\right|$ or $C(\mathbf{U}) - C(\check{\mathbf{U}})$ are smaller than some threshold, stop. Otherwise, return to **1**.

3.5 DDP controllers

DDP, as presented above, cannot be directly applied to the general problem of synthesizing autonomous motor behavior in a changing environment, for several reasons. First, we wish to generate behavior over an arbitrarily-large time span, while DDP requires the horizon N to be set in advance. Second, computing the optimal trajectory in advance is impossible in a dynamic environment, because circumstances might change during execution time, rendering the original plan suboptimal. Finally,



(a) Time course of two angular velocities.

(b) State projection.

Figure 3.2: Receding horizon trajectories for a swimmer with 3 segments (section 3.6). **(a)** three snapshots of the receding horizon trajectory (dotted) with the current finite-horizon optimal trajectory (solid) appended, for two state dimensions. **(b)** Projections of the same receding-horizon trajectories onto the largest three eigenvectors of the full state covariance matrix. As described in section 3.6.1, the linear regime of the cost compels the receding horizon trajectories to a steady swimming gait – a limit cycle.

as discussed in section 2.1.3, finite-horizon optimization results in myopic behavior towards the end of the planning horizon. For example, the swimmer, discussed below in section 3.6, tends to stop pushing itself forward and just “coast” during the tail of the optimal trajectory (see the solid lines in figure 3.2). This results in behavior that is intuitively sub-optimal, even as it meets the numerical criterion. In order to address these difficulties, we propose a receding-horizon scheme.

3.5.1 Receding horizon

In receding-horizon DDP, we use only the first step of the optimal solution, and resolve the optimization problem from the second state of that trajectory. The first point of every trajectory optimizes a behavior with an N-step-lookahead planning horizon; by using only the first point, we ensure that all the controls we use are optimal for the same horizon, guaranteeing behavioral consistency.

Starting a run of N-step DDP from scratch at every step would be prohibitively expensive. We therefore propose the following: After obtaining the solution starting from $\mathbf{x}^1$, we save the local model at $k = 1$ and proceed to solve a new N-step problem where $\mathbf{x}^2$ is the fixed initial state, and the control sequence is initialized with the policy obtained on the previous run, shifted by one time-step: $\mathbf{U}' = [\mathbf{u}^2, \mathbf{u}^3 \cdots \mathbf{u}^N - 1, 0]$. Because this control sequence is very close to the optimal solution, the second-order convergence of DDP is in full effect and the algorithm converges in 1 or 2 iterations. Again saving the model at the first time step, we continue. We stress the that without the fast and exact convergence properties of DDP near the maximum, this algorithm would be far less effective.

3.5.2 Nearest-neighbor control with offline optimization

Here we show how we can use the receding horizon scheme offline to construct a feedback controller. In order to synthesize a global controller from many local controllers, it is essential that the different local components operate synergistically. In our context this means that local models of the value function must all model the same function. As mentioned before, this is not the case for the standard DDP solution, since the local quadratic models around the trajectory are approximations to $V^k(\mathbf{x}^k)$, the time-dependent value function. However, note that in a series of receding-horizon trajectories, the locally quadratic model of $V^1(\mathbf{x}^1)$ and a locally linear model of $\pi^1(\delta\mathbf{x}) = \mathbf{u}^1 + L^1\delta\mathbf{x}$ are manifestations of the same value function, since the receding horizon allows every $\mathbf{x}^1$ to "see" a full N-step planning horizon. Therefore, the values and policies computed by receding horizon can be safely mixed.

In particular, $\pi^1(\delta\mathbf{x}) = \mathbf{u}^1 + L^1\delta\mathbf{x}$ generalizes the open-loop policy to a feedback term, which generates a *tube*-shaped basin-of-attraction around the trajectory. Having lost the dependency on the time k with the receding-horizon scheme, we need some space-based method of determining which local gain model we select at a given state. The simplest choice, which we use here, is to select the nearest Euclidian neighbor.

Outside of the basin-of-attraction of a single trajectory, we can expect the policy to perform very poorly and lead to numerical divergence if no constraint on the size of $\mathbf{u}$ is enforced. A possible solution to this problem is to fill some volume of the state space with a library of local-control trajectories [Stolle and Atkeson, 2006], and consider all of them when selecting the nearest linear gain model.

3.5.3 Online model-predictive control

Online trajectory optimization, also known as Model Predictive Control (MPC), finds the optimal policy for only one particular state – the current one. The control is applied to the system, the clock advances, and the problem is re-solved in sync with (the simulation of) the real world.

MPC has been studied extensively (see reviews by Morari and Lee [1999], Bertsekas [2005], Mayne et al. [2000], Diehl et al. [2009], and references therein), but its application to robotic domains is only starting to gain popularity [Abbeel et al., 2007]. The challenge in applying MPC in robotics is that optimization must happen in real-time, matching the timescale of the robot. MPC was initially developed in the chemical process industry, where timescales of $10sec$ or more are typical [Diehl et al., 2002]. In robotics, a timescale of $10msec$ is not uncommon, demanding careful design of both algorithm and implementation.

Constructing a feedback controller offline (section 3.5.2) is necessary in the case that the optimization computations are too slow to allow for online application of DDP. Fortunately, computational hardware (as well as apt simulation software) is constantly becoming faster and cheaper, and has reached the point where DPP can be run at these timescales.

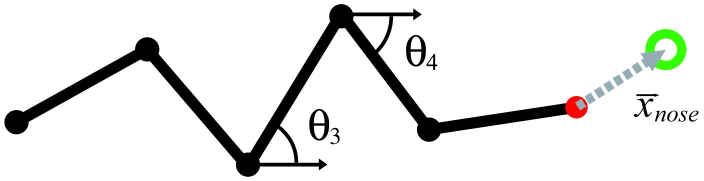


Figure 3.3: A *5-swimmer* with the "nose" point at its tip and a ring-shaped target.

3.6 The d-link Swimmer

We describe a variation of the *d-link swimmer* dynamical system, a domain that was first presented by Coulom [2002]. A *link* of length g, lying in a plane at an angle θ to some direction, parallel to $\hat{\mathbf{t}} = \left(\begin{smallmatrix}\cos(\theta)\\ \sin(\theta)\end{smallmatrix}\right)$ and perpendicular to $\hat{\mathbf{n}} = \left(\begin{smallmatrix}-\sin(\theta)\\ \cos(\theta)\end{smallmatrix}\right)$, moving with velocity $\dot{\mathbf{v}}$ in a viscous fluid, is postulated to admit a normal frictional force $-\kappa_n g \hat{\mathbf{n}}(\dot{\mathbf{v}} \cdot \hat{\mathbf{n}})$ and a tangential frictional force $-\kappa_t g \hat{\mathbf{t}}(\dot{\mathbf{v}} \cdot \hat{\mathbf{t}})$, with $\kappa_n > \kappa_t > 0$. The swimmer is modeled as a chain of d such links of lengths g_i and masses m_i. We define $\boldsymbol{\theta}$ as a d vector of link angles, $\mathbf{x}_{nose}$ as the Cartesian position of the tip of the first link, $\mathbf{q} = \left(\begin{smallmatrix}\mathbf{x}_{nose}\\ \boldsymbol{\theta}\end{smallmatrix}\right)$ as the vector of positions, and finally $\mathbf{x} = \left(\begin{smallmatrix}\mathbf{q}\\ \dot{\mathbf{q}}\end{smallmatrix}\right)$ is the $2d + 4$ state vector. The control signal $\mathbf{u}$ is a $d - 1$ vector of torque action-reaction pairs applied at each joint. These numbers hold for any tree topology of the links.

A 3D d-link model has 3 cartesian and $3d$ rotational degrees of freedom. In principle, we could have ignored the rotational degrees of freedom of the joints in the axial direction, using two hinges or even one hinge at each joint. However, using single aligned hinges would essentially embed a planar swimmer in 3D space, and multiple unaligned hinges can reach singular points (gimbal lock). We therefore chose to represent all joints as normalized quaternions (ball joints), which have no singular points. This means that each joint is described by 7 numbers, 4 for the normalized quaternion, and 3 for the angular velocities, for a total state dimension of $n = 6 + 7d$ for a 3D d-link model. The actuated states in this case are of dimension $m = 3(d-1)$.

3.6.1 The cost function

The cost function we use is:

$$
\begin{aligned}
\ell(\mathbf{x},\mathbf{u}) =& c_x \log\bigl(\cosh(\|\mathbf{x}_{nose}\|)\bigr) \\
&+ c_o \sum_j \mathcal{N}(\mathbf{x}_{nose}|\mathbf{o}_j,\mathbf{I}) \\
&+ c_u \|\mathbf{u}\|^2.
\end{aligned}
$$

The first term penalizes the norm of $\mathbf{x}_{nose} = [\mathbf{x}_1 \; \mathbf{x}_2]^\mathsf{T}$, the 2-vector from the tip of the first link to the target (the origin in internal space). This cost is minimized when the nose is brought to rest on the target. The functional form of the target-cost term is designed to be linear in $||\mathbf{x}_{nose}||$ when far from the target and approximately quadratic when close to it (figure 3.4). Because of the differentiation in equation 3.5, the solution is independent of V_0, the constant part of the value. Therefore, in the linear regime of the cost function, the solution is independent of the distance from the target, and all the trajectories are quickly compelled to converge to a one-dimensional manifold in state-space which describes steady-state swimming (figure 3.2(b)). Upon nearing the target, the swimmer must initiate a braking maneuver, and bring the nose to a standstill over the target. For targets that are near the swimmer, the behaviour must also include various turns and jerks, quite different from steady-state swimming, which maneuver the nose into contact with the target. This cost term should not be confused with the *desired state* cost of classical optimal control since values are specified only for 2 out of the $2d + 4$ coordinates. This effectively defines a target manifold (configurations where the nose is on the target), rather than a single target state. The freedom to reach this manifold on different paths manifests as diverse swimming and twisting behaviors.

The second term describes interaction between the swimmer's nose and a set of obstacles, represented by unit Gaussians $\mathcal{N}(\cdot|\mathbf{o}_j,\mathbf{I})$. Note that the means $\mathbf{o}_j$ can move along predefined trajectories, and so the controller must react predictively to their motion. Finally, the last term is a standard quadratic control-cost.

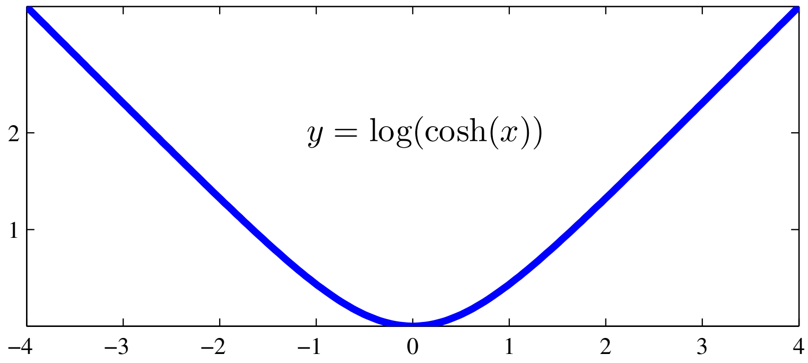


Figure 3.4: The functional form of the state-cost component.

3.6.2 Receding horizon DDP

In order to asses the controllers we constructed a real-time interaction package[5]. By dragging the target with a cursor, a user can interact with controlled swimmers of 3 to 10 links with a state dimension varying from 10 to 24, respectively. Even with controllers composed of a single trajectory, the swimmers perform well, and manage to turn, track and braking on approach to the target. These results are best illustrated by movies[6] which show real-time interaction with the swimmers.

All of the controllers in the package are for swimmers with unit link lengths and unit masses. The normal-to-tangential drag coefficient ratio is $k_n/k_t = 25$. The time-step length of the dynamics is $\Delta t = 0.05_s$. The receding horizon window is 40 time-steps, or 2 seconds.

When the state does not gravitate to one of the basins of attraction around the trajectories, numerical divergence can occur. This effect can be initiated by the user by quickly moving the target to a "surprising" location. Because nonlinear viscosity effects are not modeled and the local controllers are also linear, exponentially diverging torques and angular velocities can be produced. When adding as few as 20 additional trajectories, divergence is almost completely avoided.

[5] Available online at http://alice.nc.huji.ac.il/~tassa/pmwiki.php?n=Main.Code

[6] http://www.youtube.com/watch?v=3LagQ5_0Q0g (5-swimmer)
http://www.youtube.com/watch?v=HO5TW-UfehQ (10-swimmer).

3.6.3 DDP for MPC

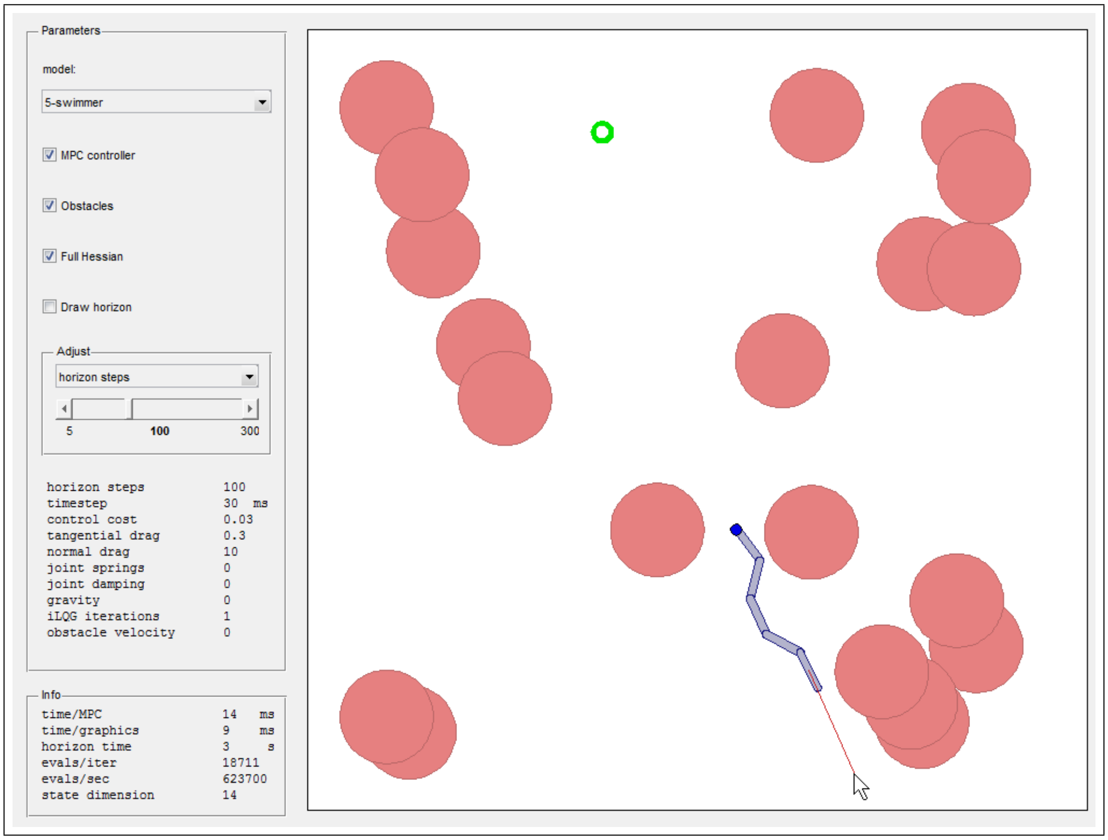
Below, we demonstrate MPC on systems with 12 to 27 state dimensions, updating the policy every $10 - 30ms$. We generate complex and robust swimming behavior on the fly, letting the user interact with the system and modify various parameters in real-time. These results are best illustrated by a movie[7] documenting online interaction with a variety of swimmer topologies and environments. As the movie shows, MPC can generate complex behavior in real-time, including: steady-state swimming, coasting, braking, and various contortional maneuvers.

One of benefits of MPC is that because there is no offline component, all the parameters can be changed in real time. To that end, we designed a rich graphical interface to allow a user to interact with the controller (figures 3.5, 3.6). The user can apply forces to the masses and move the target with the mouse (with the left and right buttons, respectively), and change a variety of parameters. These are:

- The number of time-steps N.
- The length of each time-step h.
- The control cost c_u.
- The normal and tangential drag κ_n and κ_t.
- Springs and dampers at the joints.
- Gravity.
- The number of optimization iterations at each time-step.
- The velocity of the obstacles.

Additionally, the user can choose to use only the diagonal of the Hessian, as described in section 3.3.2. In the lower left part of the window, the simulation displays the computational time of every iteration, allowing the user to quantify the performance implications of different parameter settings.

[7] http://www.youtube.com/watch?v=m0PboYKQTCc



Parameters

model: 5-swimmer

☑ MPC controller

☑ Obstacles

☑ Full Hessian

☐ Draw horizon

Adjust: horizon steps

5 100 300

```
horizon steps        100
timestep             30   ms
control cost         0.03
tangential drag      0.3
normal drag          10
joint springs        0
joint damping        0
gravity              0
iLQG iterations      1
obstacle velocity    0
```

Info

```
time/MPC             14    ms
time/graphics        9     ms
horizon time         3      s
evals/iter           18711
evals/sec            623700
state dimension      14
```

Figure 3.5: Screen capture of the user interacting with a swimmer. The nose point is in blue, the target is the green circle, and the pink discs are the obstacles. The user is pulling on the tail link with a linear spring (red line). A movie of the GUI in action is available at http://www.youtube.com/watch?v=m0PboYKQTCc

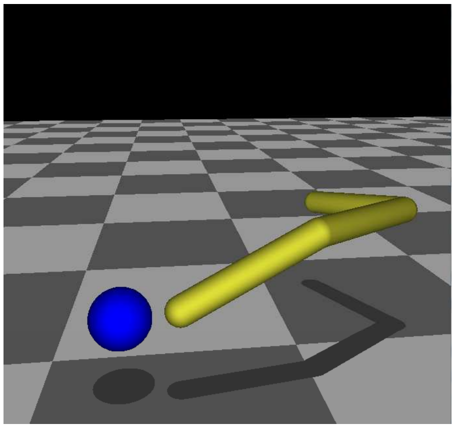


Figure 3.6: A snapshot of a 3D swimmer in action while the user is moving the target in real-time.

3.7 Epilogue: Why MPC is not enough

This chapter shows how finite-horizon optimization can bring about effective real-time behavior in a dynamic environment. The motor creativity exhibited by the swimmers demonstrates the power of optimal control in general, and specifically online MPC, as a general framework for simulating autonomous behavior. Unfortunately, iterative re-application of finite-horizon optimization is not strong enough to generate autonomous behaviors in all domains.

As mentioned above, the method of MPC is designed to avoid the problem of myopic behavior in finite-horizon planning: while an entire trajectory is planned, only the first action is executed, and planning starts again from the resulting new state. However, this process relies on warm-starting the optimization with the entire optimized trajectory from the previous iteration (shifted one time-step back). The effectiveness of MPC rests on the assumption that the solution at the previous timestep is similar to the solution at this timestep, and therefore the warm-start allows for efficient optimization.

This assumption is disrupted if the tail of the trajectory falls into a local optimum. This has no immediate effect, because only the first action is actually executed by the MPC, but as the planning horizon recedes, more and more of the trajectory is pushed into that local minimum. If at some stage the local optimality vanishes, the trajectory used to initialize the re-planning is no longer almost-optimal. This usually leads to the failure of MPC, since there is not enough time for re-optimization.

The problem of falling into inadequate local optima plagues all domains with ground collision. The hopping domain studied in section 5.5.2 can be used to illustrate this; this is a one-legged hopping robot whose task is to maintain a fixed horizontal velocity. Intuitively, the optimal behavior we expect in such a domain is a hopping gait. However, applying regular MPC to this domain results in catastrophic failure. To understand why, consider a trajectory where the last few states could involve ground collision. Such a collision would necessarily slow down the hopper, which would impede the performance of the task in the short term. As such, in the absence of an adequately long planning horizon, the optimal solution is myopic, causing the hopper to retract its leg; by avoiding the ground, the hopper tries to maintain its air velocity just a few timesteps longer. However, as the planning horizon of the MPC recedes, the locally-optimal avoidance maneouver becomes more complicated, and eventually ground impact becomes inevitable. At that stage, the MPC must plan the foot landing, but the optimization is initialized with a suboptimal trajectory that involves a bizarre contortion towards the end. Unwinding this suboptimal behavior is probably impossible within the time constraints of the robotic application, leading to the failure of MPC. Even when computation time is not a constraint (e.g., during offline optimization), recovering the new optimum when starting from a profoundly suboptimal behavior is often prohibitively hard, because the nominal trajectory, being far from optimal, would be plagued by value divergences (section 3.4.1).

In order to overcome this limitation of finite-horizon optimization, the next chapters of this dissertation discuss infinite-horizon optimization schemes which are shown to generate terrestrial locomotion behavior without falling into destructive local minima.

Chapter 4

Bayesian Estimation for Optimal Control

4.1 Introduction

Finite-horizon optimization methods (such as DDP, discussed in the previous chapter) suffer from myopic behavior (section 2.1.3), and the solution of receding-horizon re-optimization (section 3.5.1) is ineffective in domains with ground collision (section 3.7). Instead, we would like to have methods with similar efficiency but capable of solving infinite-horizon problems, in particular problems that give rise to complex periodic movements such as walking, running, or turning a screwdriver, all of which involve contact and collision with the environment.

This requires optimization over cycles. Such optimization is difficult to cast as an sequential decision problem because optimal substructure no longer holds in this case. In finite-horizon optimization, the Markov property guarantees that the future is independent of the past given the current state. However, in a limit cycle every state is both before and after every other state.

Here we overcome this difficulty by replacing the control problem with a dual Bayesian inference problem, and perform inference over a graphical model; this architecture allows for the representation of loops without requiring special hacks. First, we create a graphical model where every state along the trajectory is a node (see figure 4.1), and associate with each one a binary variable whose log-likelihood matches some performance criterion. Loosely speaking, these binary variables indicate the observation of “success”. We then postulate the observation of the value “1” at each of these binary nodes, and perform Bayesian estimation to find a likely set of assignments for the unobservable nodes given these observations. These assignments can be interpreted as the trajectory that maximizes the probability of success, hence maximizes the performance criterion.

This approach is different from the one employed in the previous chapter. In DDP we ask “what is the optimal control sequence?”; here, we ask “given that we observe success, what is the most likely route we took?”. At the same time, the method presented here uses similar computational machinery to DDP, finding local approximations of the dynamics and cost functions around a nominal trajectory. Therefore, when the dynamics are nonlinear we have to solve a sequential Bayesian inference problem: the solution at each iteration is used to re-approximate the system locally, and define the inference problem for the next iteration. When the algorithm converges, the mean of the posterior gives the locally-optimal trajectory, and the covariance of the posterior can be used to find the local feedback control law. Since computing the correct covariance is important here, we perform inference using the variational approach of Mitter and Newton [2004], leading to an algorithm based on sparse matrix factorization rather than loopy belief propagation (where only the mean is guaranteed to be correct, see Weiss and Freeman [2001]).

The estimation-control duality which is at the heart of our method arises within the recently-developed framework of linearly-solvable optimal control [Kappen, 2005, Todorov, 2009]. Several control algorithms exploiting this duality (sometimes implicitly) have been developed [Attias, 2003, Toussaint, 2009, Kappen et al., 2009], however they are limited to finite-horizon formulations – which DDP can already handle, with comparable (or, in our experience, better) efficiency. The algorithm presented here applies the estimation-control duality to general graph structures for the first time. We show how this method can lead to the optimization of a walking gait in a planar humanoid (figure 4.3).

4.2 Related work

The method presented below is related to three lines of research.

The first is sequential trajectory-optimization methods, such as DDP. It is possible to use these methods to solve for limit cycles by “attaching” the first and last states. This can be done either via penalty method, by imposing a large final cost over distance from the initial state, or exactly, by employing multipliers which enforce the state constraint, as in the method of Lantoine and Russell [2008]. We tried both of these approaches, and the inevitable result was a noticeable asymmetry around the attachment point, either in the state trajectory (when using final cost), or in the controls (when using multipliers). The main insight is that these algorithms assume the Markov property, which does not hold for a loop. One could also use a finite-horizon method with a very long horizon, that loops around the limit-cycle several times. By truncating the transients at both ends, we can get a decent approximation to the infinite-horizon solution. Clearly this is an inefficient use of computational resources, but could potentially serve as a useful validation procedure for our algorithm. However, note that such a long trajectory might be prohibitively-hard to optimize sequentially, mostly due to value divergence (section 3.4.1).

The second related body of work involves directly optimizing the total cost of a limit-cycle, while enforcing periodicity, formulating a simultaneous trajectory optimization problem. Wampler and Popovic [2009] and Ackermann and den Bogert [2010] are two recent examples from (respectively) the computer graphics and biomechanics communities. The log-likelihood that we end up maximizing below is indeed analogous to such a cost, however our method generates a feedback controller around the limit-cycle, rather than simply open-loop controls.

Finally, the last several years have seen research into the subclass of stochastic non-linear Optimal Control problems which are dual to Bayesian estimation. Specifically, Toussaint [2009] explores message-passing algorithms (Expectation Propagation) for the solution of Optimal Control problems. Murphy et al. [1999] and others have shown that when a graph has a loopy structure, message passing converges to the right mean but the wrong covariance. The procedure we describe below does not suffer from this drawback.

4.3 Optimal control via Bayesian inference

The basic intuition behind the duality we use here is that the negative log-likelihood in estimation corresponds to a state-dependent cost in control, and the difference (in terms of KL divergence) between the prior and the posterior corresponds to a control-dependent cost. The class of stochastic optimal control problems which have Bayesian inference duals in the above sense have received a lot of attention recently, because these problems have a number of other interesting properties, including the fact that the (Hamilton-Jacobi-) Bellman equation becomes linear after exponentiation [Kappen, 2005, Todorov, 2008].

4.3.1 Background on LMDPs and inference-control dualities

A linearly-solvable MDP (or LMDP) is defined by a state cost $q\left(\mathbf{x}\right) \geq 0$ and a transition probability density $p\left(\mathbf{x}'|\mathbf{x}\right)$ corresponding to the notion of passive dynamics. The controller is free to specify any transition probability density $\pi\left(\mathbf{x}'|\mathbf{x}\right)$ with the restriction that $\pi\left(\mathbf{x}'|\mathbf{x}\right) = 0$ whenever $p\left(\mathbf{x}'|\mathbf{x}\right) = 0\,.$ In infinite-horizon average-cost problems, p and π are further required to be ergodic. The cost rate function is:

$$\ell\big(\mathbf{x}, \pi\left(\cdot|\mathbf{x}\right)\big) = q\left(\mathbf{x}\right) + D_{\mathrm{KL}}\big[\pi\left(\cdot|\mathbf{x}\right)||p\left(\cdot|\mathbf{x}\right)\big].$$

The KL divergence term is a control cost which penalizes deviations from the passive dynamics. Defining the *desirability* function $z\left(\mathbf{x}\right) \triangleq \exp\big(-V\left(\mathbf{x}\right)\big)$ (where $V\left(\mathbf{x}\right)$ is the optimal value function), the optimal control is

$$\pi\left(\mathbf{x}'|\mathbf{x}\right) \propto p\left(\mathbf{x}'|\mathbf{x}\right) z\left(\mathbf{x}'\right),$$

and the exponentiated Bellman equation becomes linear in z. In particular, for finite horizon problems this equation is:

$$z^{k}\left(\mathbf{x}\right) = \exp\big(-q\left(\mathbf{x}\right)\big) \sum\nolimits_{\mathbf{x}'} p\left(\mathbf{x}'|\mathbf{x}\right) z^{k+1}\left(\mathbf{x}'\right),$$

with $z^N(\mathbf{x})$ initialized from the final cost (compare to equation 2.3). One can also define the function $r(\mathbf{x})$ as the solution to the transposed equation:

$$r^{k+1}(\mathbf{x}') = \sum\nolimits_{\mathbf{x}} \exp\bigl(-q(\mathbf{x})\bigr) p(\mathbf{x}'|\mathbf{x})\, r^k(\mathbf{x}),$$

with $r^1(\mathbf{x})$ being a delta function over the fixed initial state. Then the marginal density under the optimally-controlled stochastic dynamics μ can be shown to be:

$$\mu^k(\mathbf{x}) \propto r^k(\mathbf{x})\, z^k(\mathbf{x}).$$

The duality to Bayesian inference is now clear: z is the backward filtering density, r is the forward filtering density, p is the dynamics prior, $q(\mathbf{x})$ is the negative log-likelihood due to observation, and μ is the marginal of the Bayesian posterior. We can also write down the density p^* over trajectories generated by the optimally-controlled stochastic dynamics, and observe that it matches the Bayesian posterior over trajectories in the estimation problem:

$$p^*\left(\mathbf{x}^2, \mathbf{x}^3, \cdots \mathbf{x}^N | \mathbf{x}^1\right) \propto \prod\nolimits_{k=2}^{N} \exp\Bigl(-q\left(\mathbf{x}^k\right)\Bigr) p\left(\mathbf{x}_t | \mathbf{x}^{k-1}\right). \tag{4.1}$$

These LMDPs can be used to model continuous dynamics: for controlled Ito diffusions of the form:

$$dx = a(\mathbf{x})\, dt + B(\mathbf{x})\left(\mathbf{u}\, dt + \sigma d\omega\right), \tag{4.2}$$

and cost functions in the form:

$$\ell(\mathbf{x}, \mathbf{u}) = q(\mathbf{x}) + \frac{1}{2\sigma^2}\left\|\mathbf{u}\right\|^2, \tag{4.3}$$

the stochastic optimal control problem is a limit of continuous-state discrete-time LMDPs. The LMDP passive dynamics are obtained via explicit Euler discretization with time step h:

$$p(\mathbf{x}'|\mathbf{x}) = \mathcal{N}\left(\mathbf{x}' | \mathbf{x} + ha(\mathbf{x}),\ h\,\sigma^2 B(\mathbf{x})\, B(\mathbf{x})^{\mathsf{T}}\right), \tag{4.4}$$

where $\mathcal{N}$ denotes a Gaussian. The LMDP state cost is simply $hq(\mathbf{x})$. Note that equation 4.2 defines control-affine dynamics, and therefore the h-step transition probability of the controlled dynamics (with $\mathbf{u} \neq 0$) is a Gaussian with the same covariance as equation 4.4 but the mean is shifted by $h\,B\left(\mathbf{x}\right)\mathbf{u}$. Using the formula for KL divergence between Gaussians, the general KL divergence control cost reduces to a quadratic control cost, like the one used in section 3.6.1.

4.3.2 Periodic optimal control as Bayesian inference

Our goal now is to write down the trajectory probability p^* for infinite-horizon average-cost problems, and then interpret it as a Bayesian posterior. This cannot be done exactly, because equation 4.1 involves infinitely-long trajectories, which cannot even be represented unless they are periodic. Therefore we restrict the density to the subset of periodic trajectories with period N. This of course is an approximation, but the hope is that most of the probability mass lies in the vicinity of such trajectories. Then $p^*\left(\mathbf{x}^1, \mathbf{x}^2, \cdots \mathbf{x}^N\right)$ is the same as equation 4.1, except we have now defined $\mathbf{x}^1 = \mathbf{x}^N$.

While the trajectory probability for the control problem is no longer exact, equation 4.1 is still a perfectly valid Bayesian posterior for a graphical model with a loop. More precisely, $\exp\bigl(-q\left(\mathbf{x}\right)\bigr)$ are single-node potentials which encode evidence, while $p\left(\mathbf{x}'|\mathbf{x}\right)$ are pair-wise potentials which encode the prior. One caveat here is that, since the state space is continuous, the density may not be integrable. However, in practice we approximate $p\left(\mathbf{x}'|\mathbf{x}\right)$ with a Gaussian, so integrability comes down to making sure that the joint covariance matrix is positive definite – which can be enforced in multiple ways (see below).

Once the Bayesian posterior over limit-cycle trajectories is computed, we need to recover the underlying control law for the stochastic control problem. The obvious approach is to set

$$\pi\left(\mathbf{x}'|\mathbf{x}\right) = \frac{p^*\left(\mathbf{x}', \mathbf{x}\right)}{p^*\left(\mathbf{x}\right)},$$

where $p^*\left(\mathbf{x}', \mathbf{x}\right)$ and $p^*\left(\mathbf{x}\right)$ are the corresponding marginals of the trajectory probability, and then recover the control signal $\mathbf{u}\left(\mathbf{x}\right)$ by taking the mean of this distribution. However, this yields N different conditional distributions, and we need to somehow collapse them into a single conditional $\pi\left(\mathbf{x}'|\mathbf{x}\right)$ because the control problem we are solving is time-invariant. We have explored two alternative ways to do the combination: average the π's weighted by the marginals $p^*(\mathbf{x})$, or use the π corresponding to the nearest neighbor. Empirically we found that averaging blurs the density too much, while the nearest neighbor approach works well. An even better approach is to combine all the $p^*(\mathbf{x}', \mathbf{x})$ into a mixture density, and then compute the conditional $\pi(\mathbf{x}'|\mathbf{x})$ of the entire mixture. This can be done efficiently when the mixture components are Gaussians.

4.4 The algorithm

Probabilistic graphical models [Jordan, 1998] are an efficient way of describing conditional independence structures. A cycle-free directed graph (a tree) represents a joint probability as a product of conditionals

$$p(\mathbf{X}) = \prod_{\mathbf{x} \in \mathbf{X}} p\big(\mathbf{x}|\text{parents}(\mathbf{x})\big).$$

This equation represents the factorization properties of the joint probability function p. *Message Passing* algorithms, which involve sequentially propagating local distributions along directed graphs, provably converge to the true posterior in such acyclic graph structures.

An alternative to directed graphical models are Random Markov Fields, whose graph is undirected, and may contain cycles. The joint distribution of a Markov Field is given by

$$p(\mathbf{X}) \propto \prod_{c \in C} \psi_c(\mathbf{x}_c),$$

where C is the set of maximal cliques in the graph, and the ψ are called *potential functions*. In this type of model, message-passing algorithms are not guaranteed to converge correctly. In particular, for models where the nodes are distributed as Gaussians (as we assume below), Weiss and Freeman [2001] show that posteriors and marginals converge to correct means, but not to the correct variances.

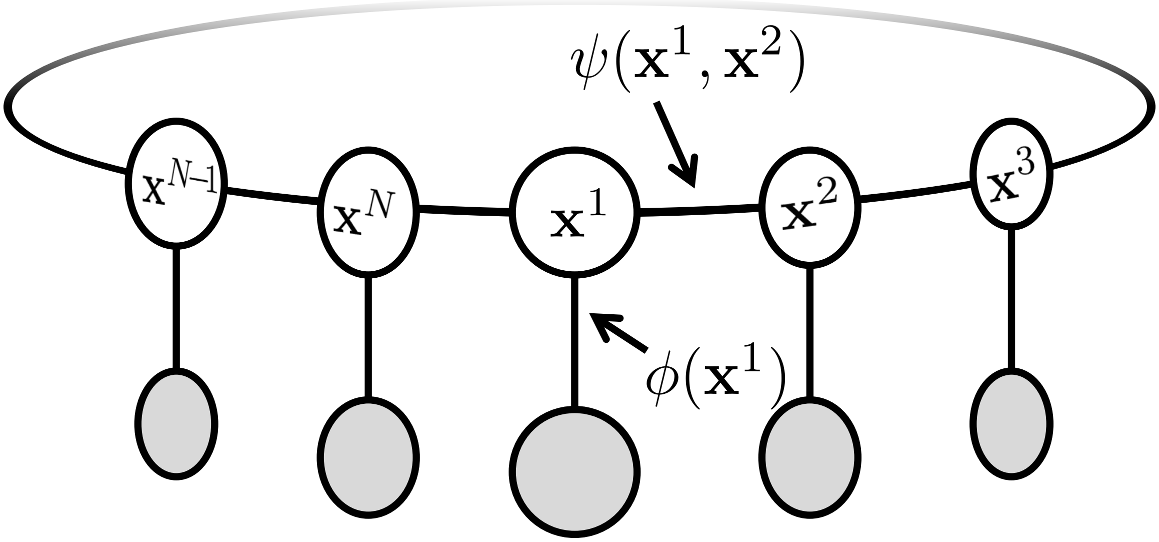


Figure 4.1: Illustration of the graphical model used in this chapter. State costs are encoded in the leaf potentials $\psi(\mathbf{x}^k)$. Dynamics and control costs are encoded in the edge potentials $\psi(\mathbf{x}^i, \mathbf{x}^j)$. See section 4.4.1.

4.4.1 Potential functions

Let $\{\mathbf{x}^k\}_{k=1}^N$ be a set of variables with the conditional dependency structure of a cycle, and let ij index over the pairs of sequential states $ij \in \{(1,2),(2,3),\ldots,(N,1)\}$. The discrepancy between the controlled dynamics and the discrete-time passive dynamics for each pair is

$$a^{ij} = \mathbf{x}^i + ha(\mathbf{x}^i) - \mathbf{x}^j. \tag{4.5}$$

The Gaussian noise leads to pairwise potentials, corresponding to $p(\mathbf{x}^j|\mathbf{x}^i)$,

$$\psi(\mathbf{x}^i, \mathbf{x}^j) = p(\mathbf{x}^j|\mathbf{x}^i) = \exp(-\tfrac{1}{2}a^{ij\mathsf{T}}\Sigma_i^{-1}a^{ij}).$$

Following equation 4.4, we might use $\Sigma_i = \; h\,\sigma^2 B\left(\mathbf{x}^i\right) B\left(\mathbf{x}^i\right)^{\mathsf{T}}$. However, in underactuated systems this formulation leads to a matrix Σ that is not full rank, and therefore cannot be inverted. In order to remedy that, we make B full-rank by allowing the agent to apply *helper forces*, which make the system fully-actuated. The vector of Ito-diffusion terms σ (4.2) is augmented accordingly with additional $n-m$ terms who take the value $1/\gamma^2$, which is equivalent to assigning a quadratic cost to helper force actuation (as in equation 4.3), weighed by the coefficient γ. In the limit $\gamma \to \infty$, the underactuated conditions are recovered.

The leaf potentials $\psi(\mathbf{x}^k)$ are composed of two parts, the state-cost $q(\mathbf{x}^k)$, and an optional prior on $\mathbf{x}^k$. This prior (not used below) could be useful when we wish to clamp certain states to specified values. For example, in the finite-horizon case where the graph is a chain, we could place a Gaussian prior on a known initial state

$$\phi(\mathbf{x}^1) = \exp(-q(\mathbf{x}^1))\mathcal{N}(\mathbf{x}^1|\mathbf{m}^1, \Sigma_1).$$

The joint distribution of the entire model is

$$p(\mathbf{X}) = p(\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N) \sim \prod_{k=1}^{N} \phi(\mathbf{x}^k) \prod_{ij=1}^{N} \psi(\mathbf{x}^i, \mathbf{x}^j),$$

where $\mathbf{X} = \text{stack}\{\mathbf{x}^k\} = [\mathbf{x}_1{}^{\mathsf{T}}\mathbf{x}_2{}^{\mathsf{T}} \cdots \mathbf{x}_N{}^{\mathsf{T}}]^{\mathsf{T}}$ is the stacked vector of all states, representing the entire trajectory. The negative log-likelihood of the trajectory is

$$l(\mathbf{X}) = \sum_k q(\mathbf{x}^k) + \sum_{ij} \tfrac{1}{2} a^{ij\mathsf{T}} \Sigma_i^{-1} a^{ij}. \tag{4.6}$$

The first term is the total state cost and the second term is the total control cost.

4.4.2 Gaussian approximation

Modeling $p(\cdot)$ as a Gaussian: $p(\mathbf{X}) \sim \mathcal{N}(\mathbf{X}|\bar{\mathbf{X}}, \mathcal{S})$, is equivalent to fitting a quadratic model to the negative log likelihood $l(\mathbf{X}) \approx \frac{1}{2}(\mathbf{X} - \bar{\mathbf{X}})^{\mathsf{T}}\mathcal{S}^{-1}(\mathbf{X} - \bar{\mathbf{X}})$. This quadratic form can be rewritten as:

$$\tfrac{1}{2}(\mathbf{X} - \bar{\mathbf{X}})^{\mathsf{T}}\mathcal{S}^{-1}(\mathbf{X} - \bar{\mathbf{X}}) = l_0 + \mathbf{X}^{\mathsf{T}}\mathbf{g} + \tfrac{1}{2}\mathbf{X}^{\mathsf{T}}H\mathbf{X}, \tag{4.7}$$

where the normalization term of the Gaussian $\frac{1}{2}\log(\det(2\pi\mathcal{S}))$ is folded into the constant l_0. The equations which translate between the two quadratic forms are:

$$\bar{\mathbf{X}} = -H^{-1}\mathbf{g}, \tag{4.8a}$$

$$\mathcal{S} = H^{-1}. \tag{4.8b}$$

It is important to note that the covariance matrix $\mathcal{S}$, as well as the Hessian H, are sparse. To see why, we focus again on equation 4.6: the terms which constitute the likelihood of the trajectory come from either individual states or consecutive pairs of states. This sparsity is key to the numerical effectiveness of the algorithm.

4.4.3 Iterative inference

Given a current approximation to the mean $\bar{\mathbf{X}}$ of the Bayesian posterior over trajectories, we expand $l(\bar{\mathbf{X}} + \delta\mathbf{X})$ to second-order in $\delta\mathbf{X}$ by computing H and $\mathbf{g}$, and then take a Newton's step to find a new mean:

$$\bar{\mathbf{X}}' = \bar{\mathbf{X}} + \operatorname*{argmin}_{\delta\mathbf{X}} l(\bar{X} + \delta\mathbf{X}) = \bar{\mathbf{X}} - H(\bar{\mathbf{X}})^{-1}\mathbf{g}(\bar{\mathbf{X}}), \tag{4.9}$$

until convergence. Again, this process can be interpreted either as repeated estimation of a joint Gaussian model, or sequential quadratic minimization of the total cost.

4.4.4 Dynamics model

We now turn to the computation of H and $\mathbf{g}$. In order to obtain a quadratic approximation for equation 4.6, we re-arrange the cost Hessians in a the sparse block-diagonal matrix $q_{\mathbf{XX}} = \mathrm{diag}\{q_{\mathbf{xx}}(\bar{\mathbf{x}}^k)\} \in I\!R^{nN \times nN}$ and stack the local cost gradients $q_{\mathbf{X}} = \mathrm{stack}\{q_{\mathbf{x}}(\bar{\mathbf{x}}^k)\} \in I\!R^{nN}$. With $q_0 = \sum_k q(\bar{\mathbf{x}}^k)$, the first term of (4.6) can be approximated as:

$$\sum_k q(\bar{\mathbf{x}}^k + \delta\mathbf{x}^k) \approx q_0 + \delta\mathbf{X}^\mathsf{T} q_{\mathbf{X}} + \tfrac{1}{2}\delta\mathbf{X}^\mathsf{T} q_{\mathbf{XX}} \delta\mathbf{X}.$$

In order to quadratize the last term of (4.6), we must approximate the nonlinear dynamics (or rather the dynamic discrepancies a^{ij}). We can do this using either a linear or a quadratic model. The former is faster to compute at each iteration, while the latter is more accurate and thus could yield convergence in fewer iterations. Which approach is better probably depends on the problem; in the examples given below we found that the quadratic model works better.

Linear dynamics approximation:

We expand the dynamic discrepancies to first order around our current approximation,

$$a^{ij}(\bar{\mathbf{x}}^i + \delta\mathbf{x}^i, \bar{\mathbf{x}}^j + \delta\mathbf{x}^j) \approx a_0^{ij} + a_{\mathbf{x}}(\bar{\mathbf{x}}^i)(\delta\mathbf{x}^i - \delta\mathbf{x}^j).$$

We construct the sparse matrix $A \in \mathbb{R}^{nN \times nN}$ as a stack of N block-rows of dimension $n \times nN$. For each pair ij, we place a negative identity matrix $-I_n$ on the j-th column-block and the dynamics Jacobians $a_{\mathbf{x}}(\mathbf{x}^i)$ on the i-th column-block. Additionally letting $\mathbf{a} = \text{stack}\{a^k\} \in \mathbb{R}^{nN}$, we have in matrix form

$$\mathbf{a}(\bar{\mathbf{X}} + \delta\mathbf{X}) = \mathbf{a}_0 + A\delta\mathbf{X}.$$

If we define $M = \text{diag}\{\Sigma_k^{-1}\} \in \mathbb{R}^{nN \times nN}$, the last term of (4.6) can be written as $\frac{1}{2}(\mathbf{a}_0 + A\delta\mathbf{X})^\mathsf{T} M(\mathbf{a}_0 + A\delta\mathbf{X})$, and the second-order expansion around $\bar{\mathbf{X}}$ becomes

$$l(\bar{\mathbf{X}} + \delta\mathbf{X}) = l(\bar{\mathbf{X}}) + \delta\mathbf{X}^\mathsf{T}(q_{\mathbf{X}} + A^\mathsf{T} M\mathbf{a}_0) + \tfrac{1}{2}\delta\mathbf{X}^\mathsf{T}(q_{\mathbf{XX}} + A^\mathsf{T} MA)\delta\mathbf{X}.$$

Comparison with equation 4.7 shows that

$$\mathbf{g} = q_{\mathbf{x}} + A^\mathsf{T} M\mathbf{a}_0 \tag{4.10a}$$

$$H = q_{\mathbf{xx}} + A^\mathsf{T} MA \tag{4.10b}$$

Quadratic dynamics approximation:

We can achieve a more accurate approximation by considering a quadratic model of the passive dynamics

$$a^{ij}(\bar{\mathbf{x}}^i + \delta\mathbf{x}^i, \bar{\mathbf{x}}^j + \delta\mathbf{x}^j) \approx a_0^{ij} + a_{\mathbf{x}}(\bar{\mathbf{x}}^i)\delta\mathbf{x}^i + \tfrac{1}{2}\delta\mathbf{x}_i{}^\mathsf{T} a_{\mathbf{xx}}(\bar{\mathbf{x}}^i)\delta\mathbf{x}^i - \delta\mathbf{x}^j,$$

where the left and right multiplications with the 3-tensor $a_{\mathbf{xx}}$ are understood as contractions on the appropriate dimensions. Though the gradient $\mathbf{g}$ is unaffected by the second order term, the Hessian picks up the product of second-order and 0th-order terms. Given the $n \times n$ matrix $\mathcal{Q}^{ij} = a^{ij\mathsf{T}}\Sigma_i^{-1} a_{\mathbf{xx}}(\bar{\mathbf{x}}^i)$ (contracting with the leading dimension of the tensor $a_{\mathbf{xx}}$), we define the block-diagonal matrix $\mathcal{Q} = \text{diag}\{\mathcal{Q}^{ij}\} \in$

$I\!\!R^{nN\times nN}$. The new approximation is now:

$$\mathbf{g} = q_{\mathbf{x}} + A^{\mathsf{T}} M \mathbf{a}_0, \tag{4.11a}$$

$$H = q_{\mathbf{xx}} + A^{\mathsf{T}} M A + \mathcal{Q}. \tag{4.11b}$$

In the experiments described below, the addition of the U term is found to significantly improve convergence.

4.4.5 Computing the policy

In this section, we describe how to obtain a local feedback control policy from the posterior marginals (the means $\bar{\mathbf{x}}^k$ and the covariance $\mathcal{S} = H^{-1}$). Let $S^k = \mathrm{cov}(\mathbf{x}^k)$ be the k-th diagonal $n \times n$ block of $\mathcal{S}$, and S^{ij} be the cross-covariance of $\mathbf{x}^i$ and $\mathbf{x}^j$ (the $n \times n$ block of $\mathcal{S}$ at the i^{th} n-column and j^{th} n-row), so that the mean of the conditional is

$$E[\mathbf{x}^j | \mathbf{x}^i] = \bar{\mathbf{x}}^j + S^{ij} S_i^{-1} (\mathbf{x}^i - \bar{\mathbf{x}}^i).$$

The feedback policy is the control which produces the expected controlled dynamics:

$$\mathbf{u}(\mathbf{x}^i) = B^{-1}\big(\bar{\mathbf{x}}^j + S^{ij} S_i^{-1} (\mathbf{x}^i - \bar{\mathbf{x}}^i) - a(\bar{\mathbf{x}}^i)\big). \tag{4.12}$$

4.4.6 Algorithm summary

0. Initialization: An initial approximation of $\bar{\mathbf{x}}$.

1. Local approximation: Compute $\mathbf{g}(\bar{\mathbf{x}})$ and $H(\bar{\mathbf{x}})$ with equation 4.11.

2. Estimation: Recompute $\bar{\mathbf{x}}$ with equation 4.9.

3. Termination: If $||\mathbf{g}||$ is smaller than some threshold, continue to **4**; otherwise, return to **1**.

4. Feedback policy: Compute $\mathcal{S}$ with equation 4.8b, and the feedback control law with equation 4.12.

4.5 Experiments

We demonstrate our algorithm on two simulated problems. A toy problem with two state dimensions and and a simulated walking robot with 23 state dimensions.

4.5.1 2D problem

The continuous diffusion consists of a non-linear spring damper system, subject to process noise in the velocity variable:

$$\begin{bmatrix} dx_1 \\ dx_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -(x_1^3 + x_2^3)/6 \end{bmatrix} dt + \begin{bmatrix} 0 \\ 1 \end{bmatrix} (udt + \sigma d\omega)$$

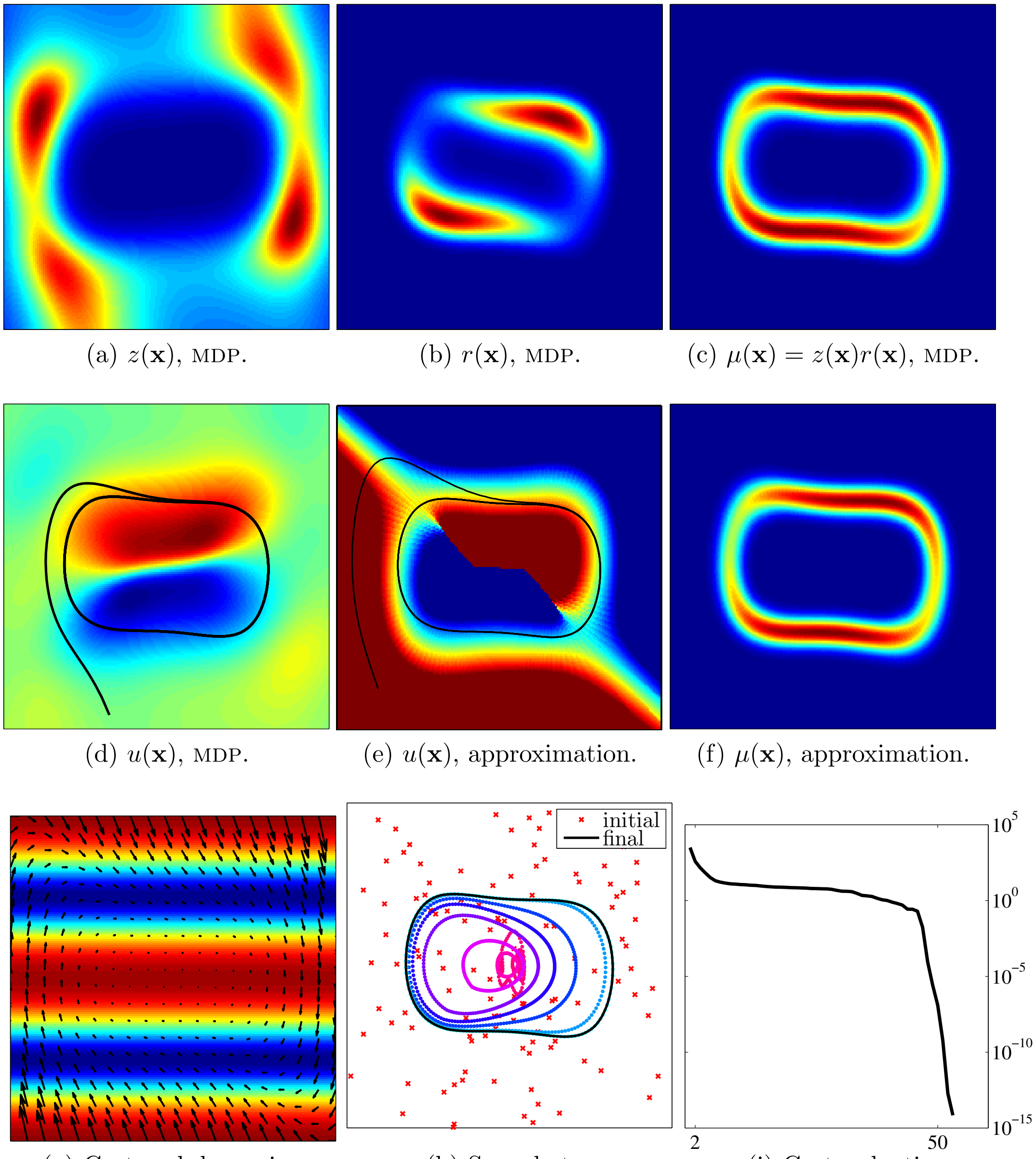
with the cost function:

$$\ell(x_2, u) = c_x \left(1 - e^{-(x_2-2)^2} - e^{-(x_2+2)^2}\right) + \frac{u^2}{2\sigma^2}$$

The state cost coefficient is $c_x = 4$ and the noise variance is $\sigma^2 = 1/4$ (and $\gamma = 1e4$). In Figure 4.1g, we show the cost function, overlayed by a vector plot of the drift, and one integrated trajectory of the passive dynamics.

We first solve this problem by discretizing the state-space and solving the resulting MDP. We use a 201×201 grid, leading to a 40401×40401 state transition matrix with 1.2×10^6 nonzeros. Discrete LMDPs can be solved by finding the leading eigenvector of a related matrix [Todorov, 2007]. The results are shown in figure 4.2 (a)-(d). Solving the MDP (using MATLAB's `eigs` function) took $86s$ on a computer powered by an Intel T9300 dual-core processor at 2.5GHz.

see caption on the next page



(a) $z(\mathbf{x})$, MDP.

(b) $r(\mathbf{x})$, MDP.

(c) $\mu(\mathbf{x}) = z(\mathbf{x})r(\mathbf{x})$, MDP.

(d) $u(\mathbf{x})$, MDP.

(e) $u(\mathbf{x})$, approximation.

(f) $\mu(\mathbf{x})$, approximation.

(g) Cost and dynamics.

(h) Snapshots.

(i) Cost reduction.

see caption on the next page

Figure 4.2: [See figures in the previous page. In all figures, red depicts higher values, blue depicts lower values. (a)-(h) show the area $[-4, 4]^2 \in (x_1 \times x_2)$.]
(a)-(d), MDP solutions for a discretized state-space. (e), (f), (h), (i), solution obtained by the proposed algorithm. (a) The exponentiated negative value function $z(\mathbf{x})$. (b) The forward filtering density $r(\mathbf{x})$. (c) The optimally controlled steady-state distribution, formed by element-wise product of $z(\mathbf{x})$ and $r(\mathbf{x})$. (d) The policy generated by the MDP solution, superimposed with one instantiation of the controlled dynamics. (e) The policy generated by the approximate solution. The color map is clipped to the values in (d), so saturated areas indicate mis-extrapolation. (f) The approximate distribution generated by our algorithm. For each pixel we measure the marginal of the state whose mean is nearest in the Euclidean sense. Note the similarity to (c). (g) The cost function, superimposed with a vector field of the passive dynamics. (h) Snapshots of the means for a particular convergence sequence, showing 8 configurations out of a total of 40. The red $\mathbf{x}$'s are the random initialization, followed by a jump to the center to decrease dynamical inconsistency, followed by a gradual convergence to the limit-cycle solution. (i) Convergence of the cost. Averaged over 15 runs with random initialization.

We then solve the same problem with our proposed algorithm. With 150 variables on the ring, the matrix H is 300×300, with 1800 nonzeros. Full convergence from a random initialization takes an average $0.3s$. A direct comparison is not relevant, since the MDP solver finds a global rather than local solution; yet the difference is striking. Once convergence of equation 4.9 is achieved, we compute the posterior with equation 4.8. In order to visualize the resulting distribution, we plot the value of the marginal of the closest Gaussian for every pixel in figure (f). The similarity of figures (c) and (f) is remarkable. Our proposed method only local, and by comparing figures (d) and (e), we see that the generated policy is valid only close to the limit-cycle. In figure (h), we see snapshots of the convergence process. Starting from a random initialization, the means jump to the center in order to decrease dynamical inconsistency, followed by a gradual convergence to the limit-cycle solution. In figure (i), we show the cost averaged over 15 runs, relative to the minimum cost achieved over all the runs. We see that all runs converged to the global minimum, with a quadratic convergence rate towards the end.

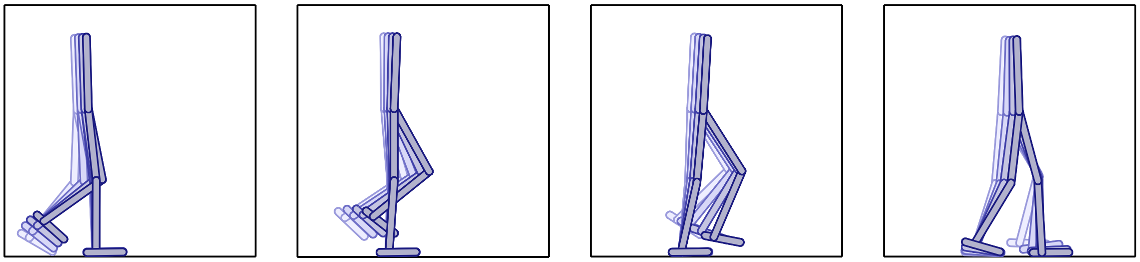


Figure 4.3: Frames from the limit-cycle solution of an optimal walking gait. See section 4.5.2.

4.5.2 Simulated walking robot

Our planar walking model is made of two legs and a trunk, each leg having three segments (thigh, shin and foot), and we imposed joint-angle constraints that ensure a biomechanically-realistic posture. The equations of motion are simulated using a planar physics engine[8] implemented in MATLAB; the parameter units are arbitrary, but can be assumed to have the appropriate units of a self-consistent system (e.g. MKS). The length of the foot segments is 1, and all other segments are of length 2. The segment masses are 0.1 for the foot, 0.4 for the shin, 1 for the thigh, and 4 for the trunk. A control signal of dimension 6 acts on the joints (hips, knees, ankles). The seven segment angles, together with the planar position of center-of-mass, make for a system with 9 degrees-of-freedom, or 18 state dimensions. In order to allow the gait to take a limit-cycle form, we remove the horizontal position dimension of the center-of-mass (see section 6.3.4), for a total of 23 state dimensions.

Ground reaction forces are computed using SLCP (section 2.4.2). However, the resulting dynamics are not control-affine, and therefore do not conform to the LMDP formulation (section 4.3.1). In order to recover control-affine dynamics, we augment the state space with 6 first-order filters of the control signal, with a time constant of 0.025. This adds 6 activation dimensions (which determine the instantaneous torques) to the state space.

We use 80 time-steps of length 1/80, for a step period of 1. The matrix H is of dimension 1840×1840, with 105840 non-zeros. Each iteration took 0.2 seconds on

[8] Developed by Yuval Tassa, available at http://alice.nc.huji.ac.il/~tassa/pmwiki.php?n=Main.Code

the same computer as above, with about 300 iterations until convergence. The initial trajectory is a fixed for all time-steps, positioned in a physically-unrealistic open-leg stand at some distance above ground.

We find that starting with a high value of γ leads to poor convergence rate; this is resolved by using a shaping protocol (section 2.5): we start from $\gamma = 0.01$ and optimize until convergence; we then multiply γ and re-solve, repeating until $\gamma = 1e4$. After the first optimization, subsequent shaping iterations usually converge within 10-20 iterations.

In order to produce upright walking, we use a cost function with three terms: First, a quadratic penalty for deviation of the center-of-mass's horizontal velocity $v_{\mathbf{x}}$ from a desired value of 2. Second, a linear reward for the vertical height of the trunk's upper tip h_T, to promote upright posture. Third, a cost term that is quadratic in the muscle activation dimensions c. The total weighted state-cost is

$$q(\mathbf{x}) = (v_{\mathbf{x}} - 2)^2 - 0.1 h_T + 0.01\|c\|^2. \tag{4.13}$$

Convergence for this problem is robust, with different initializations converging to the same solution. The resulting gait is demonstrated in Figure 4.3, as well as a movie, available at http://www.youtube.com/watch?v=HJdqOve5IEs.

4.5.3 Feedback control law

One of the main advantages of the algorithm presented here is that the generated control law (4.12) includes feedback terms, forming an effective basin-of-attraction around the optimal limit-cycle. In Figure 4.4, we illustrate convergence to the limit cycle from perturbed states, of the simulated walking robot. The means $\bar{\mathbf{x}}^k$ are projected onto $\{d_1, d_2\} \in \mathbb{R}^{23}$, the two leading eigenvectors of the covariance $\mathrm{cov}(\bar{\mathbf{x}}^k)$ (i.e. the two leading PCA directions). One state is then randomly perturbed, and used as an initial state for a controlled trajectory. The distribution of the perturbations is chosen so that most trajectories (solid) are within the basin-of-attraction and converge to the limit-cycle (successful walking), while several (dashed) diverge (stumble and fall).

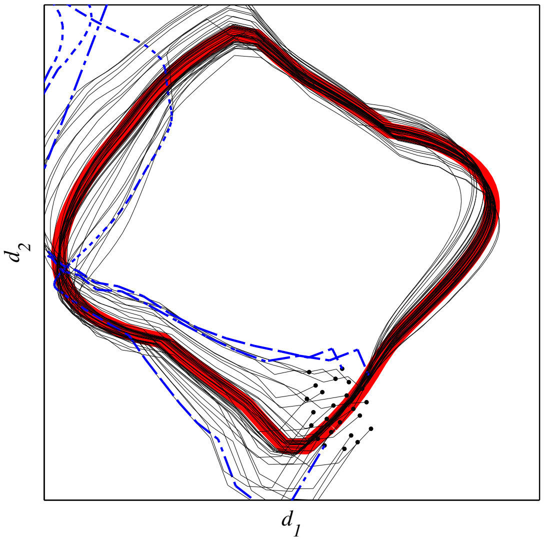


Figure 4.4: Robustness to perturbations of the simulated walking robot, under the feedback control law. The axes d_1 and d_2 are the largest eigenvectors of the covariance of the $\bar{\mathbf{x}}^k$. The optimal limit cycle is in thick red, superimposed with simulated trajectories. The trajectories' initial states are indicated by dots; converging trajectories are drawn with thin solid lines, and diverging ones are drawn with thicker dashed lines.

4.5.4 Running

After finding a walking gait that is optimal for a speed of 2m/s, we changed the desired speed in equation 4.13 to 4m/s, and ran the optimization again, initialized with the walker. The result is a running gait which can be seen in a movie, available at `http://www.youtube.com/watch?v=1So3xBfVP_k`.

4.6 Discussion

This chapter presents a method of solving optimal control problems with a periodic solution. Using the control-estimation duality which holds for problems of the form (4.2)-(4.3), we recast the problem as Bayesian inference, and maximize the likelihood of a Gaussian approximation. Importantly, the computational complexity of the methods scales polynomially with the state dimension.

The algorithm produces a local feedback control law around the trajectory, but the volume of the basin-of-attraction is limited by the validity of the Gaussian approximation. In order to expand the basin of attraction afforded by the feedback controller, we may combine this offline method with online methods like Model Predictive Control, by using the infinite-horizon cost-to-go as a final-cost for a finite-horizon trajectory optimizer. This idea is realized in the next chapter.

Chapter 5

Infinite-Horizon Model Predictive Control

5.1 Introduction

The *finite-horizon* criterion of optimality (section 2.1) seeks to minimize the future cumulative cost over some predefined planning horizon. The repeated online solution of the finite-horizon problem for an ever-receding horizon is called Model Predictive Control (MPC), as discussed in section 3.5.3. Online optimization is possible because this class of problems is relatively easy to solve, but may result in undesirable myopic behavior due to the limited planning horizon (section 3.5). In particular, it is ineffective in domains with contacts, as discussed in section 3.7.

In contrast, the *infinite-horizon* criterion (section 2.1) poses the optimization problem in terms of the average cost over an infinitely-distant horizon. This formulation is used in domains where there are well-defined absorbing states (where the system can remain forever), as well as domains where the solution is periodic and forms a limit cycle. This class of optimization problems is currently too computationally-intensive to be solved online.

The strengths of these two formulations can be combined by using the cost-to-go of an infinite-horizon solution as the terminal cost of the finite-horizon problem. This scheme is called Infinite-Horizon MPC (IHMPC).

Existing IHMPC algorithms can only tackle domains with fixed goal states (section 5.2). In contrast, the algorithm presented in this chapter tackles the more general problem where the optimal behavior results in a limit cycle. We use an offline optimization scheme to construct a locally-quadratic approximation of the infinite-horizon cost-to-go around the optimal limit-cycle. We then use this approximation as a terminal cost for online IHMPC.

We show how this approach can generate robust control for a simulated hopping robot (figure 5.3) in real time. The optimal limit cycle is found through offline optimization (chapter 4), and the infinite-horizon average-cost value function (section 5.4) is fitted around the closed trajectory; this approximation is used for IHMPC (section 5.5.2). The computation of the MPC optimization is fast enough to allow real-time simulation and control. The resulting controller yields robust behavior that can effectively recover from any perturbation.

Finally, we address the question of modeling errors. The method presented here uses model-based dynamic trajectory optimization both online and offline; however, such dynamic models would always be somewhat inaccurate for real robots. Robustness with respect to modeling errors is studied by altering the dynamics of the simulated plant, so that the optimization uses an incongruent dynamic model (section 5.5.3). Even in this case, our controller can generate the desired behavior (figure 5.5) and recover from perturbations.

5.2 Related work

Chen and Allgower [1998] were the first to suggest the combination of MPC with an infinite-horizon optimization problem, and this approach has been studied extensively in the past decade [Pannocchia et al., 2003, Almeida, 2008, Hu and Linnemann, 2002]. However, current IHMPC algorithms make the strong assumption that the task is specified in terms of reaching a goal state. In such a case, only a finite amount of time is spent away from this target, and so the conditions around the goal state dominate the infinite-horizon considerations. Even if the domain exhibits non-linear dynamics, a linear approximation can be used effectively in some small region around the goal state, which is bound to dominate the infinite horizon because once the system enters this region, it remains there forever. Together with a quadratic approximation of the cost function, the infinite-horizon problem takes the familiar form of a Linear-Quadratic Regulator (LQR), and can be solved using Ricatti equations. In contrast to existing IHMPC algorithms, the method proposed in this paper allows us to tackle domains with no single goal state.

The optimization criterion of infinite-horizon average-cost was first presented by Schwartz [1993] [see also Gosavi, 2004, Mahadevan, 1996, Singh, 1994], and was used for policy search [Baxter and Bartlett, 2001, Tedrake et al., 2004]. However, we are not aware of its use in conjunction with MPC.

5.3 Infinite-horizon model predictive control

In IHMPC, we compute the infinite-horizon value $\tilde{V}(\mathbf{x})$ of a given state $\mathbf{x}$ by breaking the infinite sum in equation 2.8 into two parts:

$$\lim_{\mathfrak{N}\to\infty}\left[\sum_{k=1}^{\mathfrak{N}}\ell\Big(\mathbf{x}^k,\pi\big(\mathbf{x}^k\big)\Big)-\mathfrak{N}c\right]=\sum_{k=1}^{N-1}\ell\Big(\mathbf{x}^k,\pi\big(\mathbf{x}^k\big)\Big)-(N-1)c$$
$$+\lim_{\mathfrak{N}\to\infty}\left[\sum_{k=N}^{\mathfrak{N}}\ell\Big(\mathbf{x}^k,\pi\big(\mathbf{x}^k\big)\Big)-\mathfrak{N}c\right].$$

Note that the first RHS sum is the finite-horizon value of the first state (2.2), the second term is constant, and the last sum is the infinite-horizon average-cost value function of the the N^{th} state (2.8). Therefore:

$$\tilde{V}\big(\mathbf{x}^1\big)=V\big(\mathbf{x}^1\big)+\tilde{V}\big(\mathbf{x}^N\big)+\text{constant}.$$

This identity implies a compositionality of controllers: if we use the infinite horizon value function $\tilde{V}(\mathbf{x}^N)$ as the terminal cost function ℓ_N in (2.2), the optimal finite-horizon behavior would also be the one minimizing the infinite-horizon average cost!

If we knew $\tilde{V}(\mathbf{x})$ for every state, there would be no need for MPC, because we could derive a globally optimal controller directly from the value function. Naturally, this is impractical, bacause $\tilde{V}(\mathbf{x})$ cannot be computed online, and in high-dimensional domains it is impossible to pre-compute $\tilde{V}(\mathbf{x})$ for the entire state space. However, if we could identify regions of state-space towards which the optimally-controlled system is likely to converge, we could focus our computational resources, and estimate $\tilde{V}$ only for that area of state space. This is likely to generate a good approximation, since most MPC trajectories would end in that region. Fortunately, the Bayesian inference algorithm of the previous chapter is geared exactly for the task of identifying regions of high posterior distribution under optimally-controlled dynamics. Therefore, we may estimate $\tilde{V}$ around the limit cycle found by the Bayesian algorithm, and use MPC with that approximation as terminal cost. This combination yields good performance as long as the MPC trajectory terminates close enough to the limit cycle.

How to approximate $\tilde{V}$ around the limit cycle? Previous studies of IHMPC focused on *tracking* tasks, where a well-defined goal state is available. In that case, LQG theory can be used to find $\tilde{V}$ analytically (as it is guaranteed to be a quadratic function around the goal state). However, we try to generate autonomous behavior without explicitly specifying a target state at every time step. In section 5.4 we present an algorithm for approximating $\tilde{V}$ that is based on minimizing least squares.

In summary, the IHMPC has three stages: first, we identify the optimal limit cycle using the Bayesian method of chapter 4 for offline trajectory optimization; then, we construct local quadratic approximations of $\tilde{V}$ around every state. Finally, we use that quadratic approximation onlime, as terminal cost in MPC optimization. As long as MPC can find a solution where $\mathbf{x}^N$ is close to that limit cycle, this constitutes a good approximation of the infinite-horizon average cost optimal behavior.

5.4 Approximating the infinite horizon value function

Given an optimized closed trajectory $\mathbf{X}$ and the corresponding open-loop sequence $\mathbf{U}$, we approximate the value function of the infinite-horizon problem locally as a quadratic function. This means that for each of the points $\mathbf{x}^k \in \mathbf{X}$ we seek the

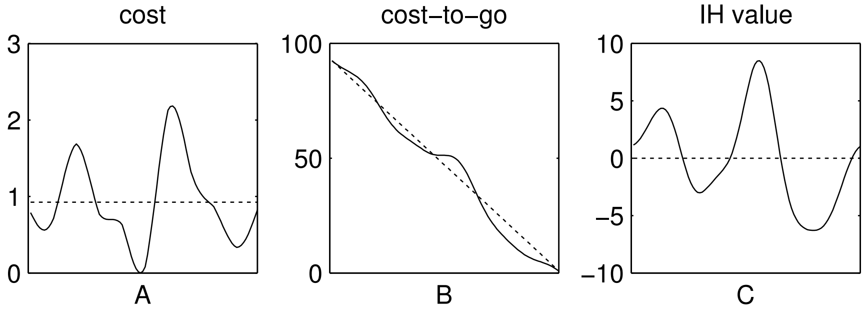


Figure 5.1: Illustrating the calculation of the 0^{th}-order term of the infinite-horizon value function along a limit cycle (described in section 5.4.1). A. the cost ℓ at every point along a limit cycle (solid), and the average cost (dashed). B. the finite horizon cost-to-go (solid) and the average cost-to-go (dashed), integrated along one cycle of the closed trajectory. C. the resulting infinite-horizon average-cost value function, shifted so that its mean over the entire limit cycle is zero.

coefficients $\tilde{V}_0^k, \tilde{V}_{\mathbf{x}}^k, \tilde{V}_{\mathbf{xx}}^k$ so that for small $\delta\mathbf{x}$,

$$\tilde{V}(\mathbf{x}^k + \delta\mathbf{x}) \approx \tilde{V}_0^k + \delta\mathbf{x}^{\mathsf{T}}\tilde{V}_{\mathbf{x}}^k + \frac{1}{2}\delta\mathbf{x}^{\mathsf{T}}\tilde{V}_{\mathbf{xx}}^k\delta\mathbf{x}.$$

The constant terms $\tilde{V}_0$ can be computed directly, while the coefficients $\tilde{V}_{\mathbf{x}}, \tilde{V}_{\mathbf{xx}}$ can be estimated using least-squares.

5.4.1 Computing the 0^{th}-order terms

Since we can measure the cost ℓ along every point of the limit cycle, we can calculate the average cost per step

$$c = \frac{1}{N}\sum_{k=1}^{N}\ell\big(\mathbf{x}^k, \mathbf{u}^k\big).$$

We define the cost-to-go along the limit cycle:

$$V^k = \sum_{k=i}^{N}\ell\big(\mathbf{x}^k, \mathbf{u}^k\big),$$

and the average cost-to-go:

$$\grave{V}^k = (N-k)c.$$

The infinite-horizon value function is the deviation of the cost-to-go from the average cost-to-go

$$\tilde{V}^k = V^k - \grave{V}^k + b$$

(see figure 5.1), shifted by the scalar b to enforce

$$\sum_{k=1}^{N} \tilde{V}^k = 0.$$

This computation is illustrated in figure 5.1.

5.4.2 Estimating the quadratic model

Equation 2.4 describes the relationship between the time-dependent, finite-horizon value functions around two consecutive states, and this relationship is applied in equations (3.3, 3.9, 3.10) of the backward pass (section 3.2.1). Equation 2.9 describes the same relationship between the time-independent, infinite-horizon value function around two consecutive states. However, a straightforward application of the backward pass equations to integrate $\tilde{V}$ backwards along the limit cycle does not result in a good approximation of the value function. This is because the computation used in the finite-horizon backward pass (section 3.2.1) relies on the dynamic-programming principle of *optimal substructure* (section 2.2): that the future cannot affect the past. This principle does not hold in limit cycles, where every state is visited both before and after every other state. We found that in practice, simple application of the backward pass along several rounds of the limit cycle requires a significant amount of regularization (section 3.4.1) since this trajectory is not optimal in the finite-horizon sense, and the resulting value function fails to produce robust behavior.

As an alternative, we can pose equation 2.9 in terms of least-squares minimization. The key idea is to define a scalar measure of *discrepancy* between every pair of consecutive quadratic model, and search for a combination of quadratic models that minimize this discrepancy over the entire limit cycle.

For every state k around the limit cycle, we represent the values of the quadratic model $\tilde{V}^k_{\mathbf{x}}, \tilde{V}^k_{\mathbf{xx}}$ as a single vector $\mathbf{V}^k$, concatenating the gradient vector $\tilde{V}_{\mathbf{x}}$ and the upper triangular part of the symmetrical matrix $\tilde{V}_{\mathbf{xx}}$ into a single search space. In order to compute $d(\mathbf{V}^k, \mathbf{V}^{k+1})$, the discrepancy between two consecutive models, we follow the following procedure: we expand $\mathbf{V}^{k+1}$ into the full quadratic model $\tilde{V}_{\mathbf{x}}^{k+1}, \tilde{V}_{\mathbf{xx}}^{k+1}$, and apply the backward pass equations (3.3, 3.7) *without* regularization to obtain a candidate quadratic model $\tilde{V}_{\mathbf{x}}^{k+1\to k}, \tilde{V}_{\mathbf{xx}}^{k+1\to k}$ around state $\mathbf{x}^k$, which can be represented as a single vector $\mathbf{V}^{k+1\to k}$. We now define the discrepancy using regular Euclidean norm in coefficient space:

$$d(\mathbf{V}^k, \mathbf{V}^{k+1}) = \left|\left|\mathbf{V}^k - \mathbf{V}^{k+1\to k}\right|\right|^2.$$

The total discrepancy of a set of quadratic models around the limit cycle is simply the sum of discrepancies for every consecutive pair. This is a nonlinear sum-of-squares optimization problem, whose variables are the coefficients of the quadratic models around the ring. Although the number of variables in this problem is quadratic in the dimensionality of the domain, note that only variables of consecutive pairs interact, leading to a sparse Hessian.

5.5 Results

We present results from two domains. First, we tackle the same toy 2D domain studied in section 4.5.1. This problem is small enough to afford a global solution through state-space discretization, which serves as ground truth. This allows us to evaluate the quality of our approximations. Then, we apply our method to a simulated domain of a one-legged hopping robot.

5.5.1 2D problem

Here we study the same 2D problem that is presented in section 4.5.1. We re-use the ground truth found by the MDP solution, and the local solution around the optimal limit cycle, which are reproduced at the top two panels of figure 5.2.

We identify quadratic approximations of the value function around every element of the limit cycle according to the algorithm presented in section 5.4. We then use these quadratic models as terminal cost in an MPC with various planning lengths, and compute the IHMPC policy over every state in the same 201x201 grid used for MDP. The two lower panels of figure 5.2 shows the resulting policies for different lengths of the MDP horizon. With a planning horizon of two steps, numerical artefacts are apparent (bottom left), but the policy is already somewhat similar to the ground truth policy. With a planning horizon of 12 steps (bottom right), the IHMPC policy becomes effectively equal to the true policy almost everywhere.

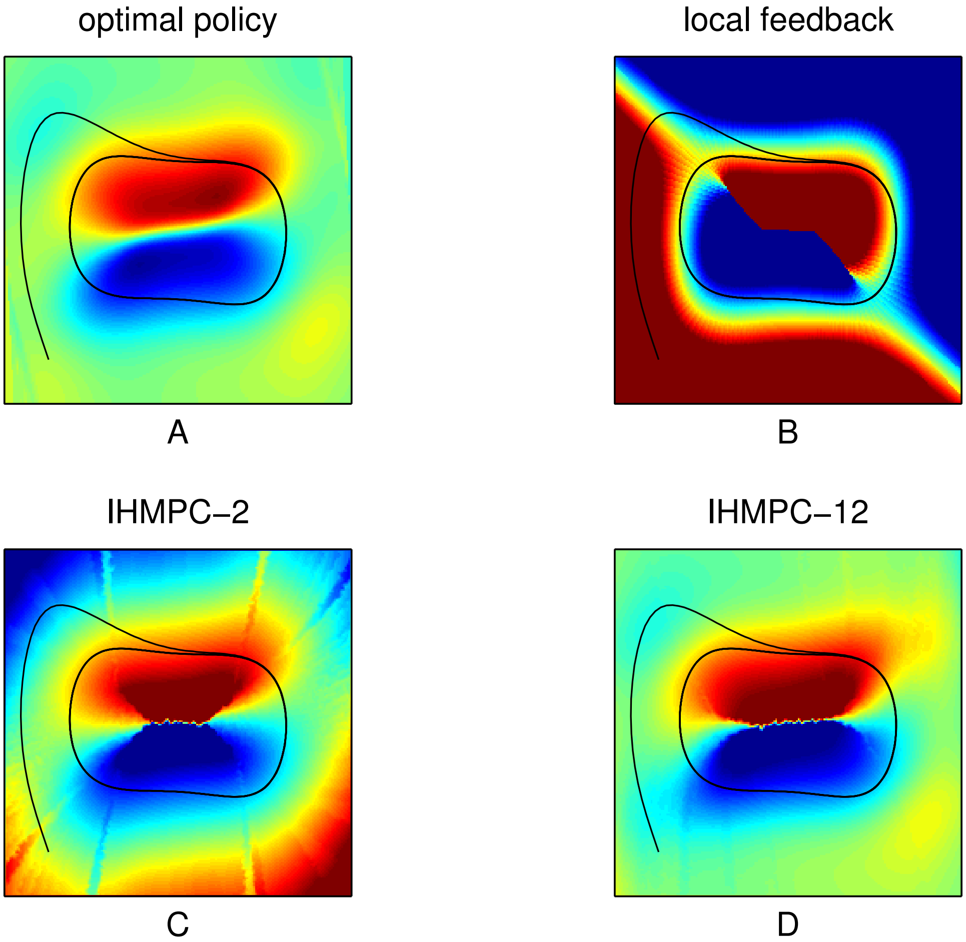


Figure 5.2: A. the ground-truth policy solved through MDP discretization. Compare to 4.1d. B. the locally-linear feedback policy computed around the limit cycle. Compare to 4.1e. C. IHMPC with a lookahead horizon of 2 steps. D. IHMPC with a horizon of 12 steps. The complete limit cycle is 150 steps long. In all figures, red is positive actuation, blue is negative actuation.

body	length (cm)	radius (cm)	mass (kg)
foot	50	7	8.77
lower leg	50	6	6.33
upper leg	50	5	4.32

Table 5.1: Morphological specification of the hopping robot

5.5.2 Planar hopping robot

The mechanical system of the hopping robot is composed of three body segments and two joints, and so this domain has a 9-dimensional state space and two-dimensional control space. The model is simulated using MuJoCo (section 2.4.1); the masses and segment lengths are specified in table I. The ground interaction forces are computed using SLCP (section 2.4.2) with $\sigma = 1$. The task requires the hopper's center of mass to maintain a fixed horizontal velocity $\dot{y}_{COM}$ of 1 m/s, while keeping its vertical position x_{COM} around 1 m:

$$\ell(\mathbf{x}, \mathbf{u}) = 10(\dot{y}_{COM} - 1)^2 + 0.1(x_{COM} - 1)^2 + 0.01||\mathbf{u}||^2.$$

We start by finding an optimal limit cycle with $N = 40$ steps of 20msec. The optimization of the hopping gait is implemented in MATLAB, and takes about an hour of computation on a dual-core (Intel T9300, 2.5GHz) processor. In order to initialize the optimization, all points along the limit cycle are set to a physically-unrealistic standing pose with the foot hovering over the ground. We follow the same shaping routine described in section 4.5.2, starting from $\gamma = 0.01$ and multiplicatively

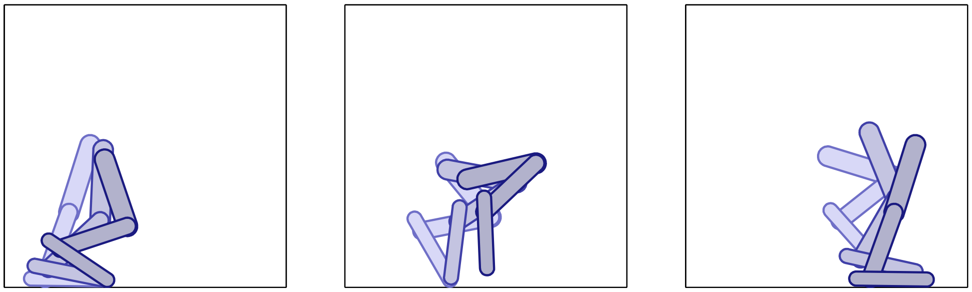


Figure 5.3: The hopping robot's limit cycle.

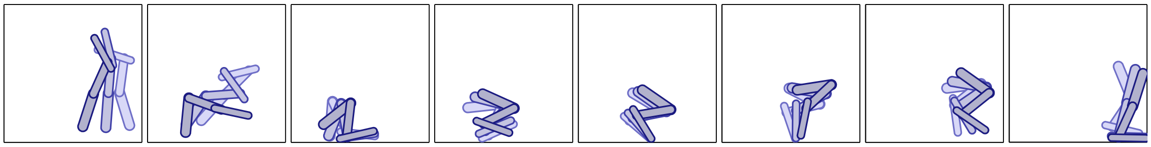


Figure 5.4: IHMPC recovering from a starting state that is radically far from the limit cycle. Note how in the first frame on the right, the hopping robot is flying upside-down, and yet manages to eventually get a foothold and push itself back up.

increasing until $\gamma = 1e4$. Fitting the quadratic approximation of the value function around the limit cycle takes about 5 minutes of computation. Both steps are run offline only once.

We then apply IHMPC with a planning horizon of 10 steps. Every MPC optimization loop was allowed at most 5 iterations. When the simulation is unperturbed, the MPC optimization converged in a single iteration in every timestep. This allows us to apply IHMPC in real-time. IHMPC was able to generate robust behavior over the entire state-space. The basin of attraction of the IHMPC effectively covers the entire volume of state space, and the hopping robot can recover from any perturbation and resume its gait. Even when the hopper is thrown to the ground (frames 1 and 2 in figure 5.4), it can find an appropriate motor sequence to get up (frames 3 and 4) and resume its hopping (frames 5-8).

5.5.3 Robustness to modeling errors

We use model-based optimization for both the offline and the online phases of our IHMPC algorithm. However, it is impossible to build a perfectly-accurate dynamical model of a physical robot. Therefore, in order to be useful for real-world robotic applications, any model-based control method must be robust to modeling errors. We examine this question by modifying the model used for plant simulation, making it incongruent with the model used for optimization. We increase the length of the top segment by 60%, from 50cm to 80cm. In this case, the original limit cycle is no longer the optimal gait for the modified morphology, and the MPC optimizations are using the wrong model of the dynamics. However, IHMPC is able to maintain effective hopping in this case as well (figure 5.5).

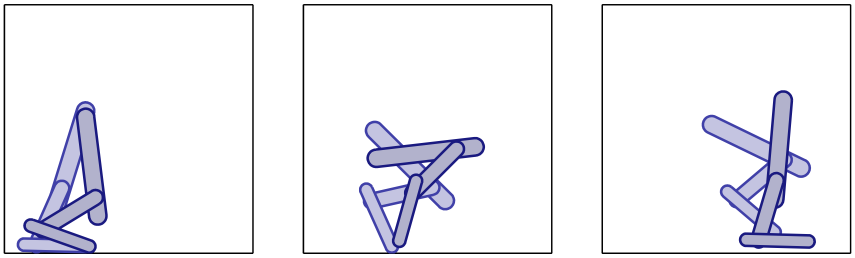


Figure 5.5: The limit cycle of a hopping robot with altered morphology (the top body's length is extended by 60% to 80 cm). IHMPC can recover stable hopping even as the planner still uses the original model.

5.6 Discussion

The main contribution of this chapter is the presentation of an algorithm for Infinite Horizon Model Predictive Control (IHMPC) for tasks with no specified goal state. We show that IHMPC can generate robust behavior in domains with discontinuities and unilateral constraints, even at the face of extreme external perturbations. We also show how our controller can maintain the desired behavior in the face of significant modeling errors.

Our results show a hopping robot that can get up when tossed to the ground. The controller's capacity to figure out how to get up is not part of the behavioral repertoire that was pre-computed offline. Instead, it is a result of MPC online trajectory optimization. This highlights how IHMPC allows for "motor creativity", as the online optimization can tackle any perturbation, as long as it can find a finite trajectory that terminates close enough to the optimal limit cycle. The capacity for such open-ended behavior is essential for robots who share the space with unpredictable humans and their ever-shifting environment.

Chapter 6

Inverse-Dynamics Trajectory Optimization

6.1 Introduction

In all the simulations presented in this dissertation thus far, the dynamics are modeled via a *forward* formulation, where the next state is computed from the current state and action. This computation uses numerical integration, which suffers from numerical stability issues — when the dynamics are stiff or non-smooth, small time-steps are necessary to prevent divergence. This presents a significant challenge to algorithms of trajectory optimization, which scale linearly with the number of time-steps.

As an alternative, this chapter uses *inverse* dynamics, where the current action is computed from the current and next state (section 6.3.2). Inverse dynamics does not rely on numerical integration, and is therefore immune to such problems of numerical instability, allowing for the representation of the same trajectory using a significantly smaller number of time-steps. Inverse dynamics can be employed in algorithms of *simultaneous* trajectory optimization, which use an explicit representation of the state-space trajectory.

In domains with contacts, inverse dynamics are particularly challenging, since the inverse-dynamics computation must be able to differentiate between the forces that are applied by the agent and the reaction forces. To see why, consider the effort required for voluntarily bringing your foot to a halt at a zero distance above the ground, compared with the kinematically-equivalent situation where the foot stops at zero height due to collision with the ground.

In order to resolve this issue, we employ an invertible contact model which is based on convex optimization (section 6.3.5). Like SLCP (section 2.4.2), this model provides contact smoothing, and allows us to formulate the contact-driven optimization problem in general terms, using only the continuous trajectory representation, and avoiding hybrid modeling or other contact-specific machinery. However, it offers an additional, unique feature — the capacity to compute inverse dynamics through the contact reaction forces.

Chapter 4 presents a method for optimizing limit cycles using forward dynamics using the linearly-solvable MDP framework (section 4.3.1), which only applies to domains that are control-affine. In order to use it with SLCP (which generates dynamics that are not control-affine), we had to use a third-order system with excitation-activation dynamics (section 4.5.2), which inflates the number of state dimensions. The method presented here applies directly to systems that do not conform to the LMDP formulation, and requires no additional dimensions.

We demonstrate the efficacy of this algorithm by solving a domain of bipedal running. This is an interesting domain because it includes a severely-underactuated flight phase, where the agent has no way to influence the trajectory of its center of mass. This makes it difficult to apply traditional gait design approaches (such as ZMP) to bipedal running. We optimize the motion pattern of a simulated 3D humanoid with 31 degrees of freedom, which is prohibitively-large for most standard optimization schemes. The resulting gait is best illustrated by a movie[9]. Most notably, a coordinated movement of arms and legs emerges, as the upper body motion balances the angular momentum created by leg swing.

6.2 Related work

Dynamics with contacts are a profound challenge to optimal control, and the standard way to overcome this obstacle is by using a hybrid representation. In a hybrid model,

[9] available at http://www.youtube.com/watch?v=CL11JTnH6P8

teh continuous state space is augmented with a discrete set of states representing the boolean contact state of every possible contact pair. Given the discrete contact state, the continuous dynamics can be computed by treating the contact as a constraint. Additional computation is used to ensure that the contact forces remain inside the friction cone, adjusting the discrete contact state as necessary.

Hybrid modeling requires explicit representation of the contact state and significantly increases the burden of modeling the dynamics. For example, Bessonnet et al. [2010] use inverse dynamics to optimize a gait pattern of a 3D robotic model. However, the dynamic model they use is not in general form, and their method requires an analytic formulation of a ZMP condition; therefore, their method cannot be easily extended to other morphologies, or a richer set of contacts. In another example, Ren et al. [2007] study a planar model of human walking, yet their model cannot compute ground reaction forces during double support, and these must be approximated via data-driven heuristics.

Inverse dynamics has been successfully applied to gait design using hybrid modeling.Wampler and Popovic [2009] optimize the trajectory using SQP, and optimize the contact times separately using Covariance Matrix Adaptation. Kim et al. [2008] apply inverse dynamics to a 3D biomechanical hybrid model by explicitly representing and optimizing the timing of each gait phase (single-support vs. double-support). Mombaur et al. [2010] also present an optimization method which solves simultaneously for body poses and contact times.

In contrast, the method we present here employs inverse dynamics in a fully-continuous state-space, eliminating the need for hybrid representation.

6.3 Inverse dynamics

6.3.1 Minimal representation

In order to represent the mechanical state of a multibody system, we need to specify both the positions and velocities along some generalized coordinates. However, in simultaneous trajectory optimization, the velocities are represented implicitly, by virtue of representing consecutive positions. Therefore, it is redundant to represent the trajectory in all dimensions of state space (including both position and velocity). This redundancy leads to potential incongruence, which is manifested as an implicit set of constraints between the various variables of our search space. This gives rise to a Hessian which is dominated by those over-specified directions, which results in a slower optimization procedure (since self-inconsistency is penalized independently at every step).

In order to eliminate the mutual dependencies, we wish to perform the same simultaneous optimization using a *minimal* representation. Since every consecutive pair of positions implicitly defines a velocity, we can rewrite the same optimization problem while explicitly representing only the positions around the limit cycle; this not only reduces the number of dimensions, but more importantly — eliminates mutual dependencies between the dimensions of the search space.

6.3.2 Computing the inverse dynamics

We assume that the state $\mathbf{x} \in \mathbb{R}^n$ is composed (at least) of a set of generalized coordinates $\mathbf{q} \in \mathbb{R}^q$ and their respective velocities $\dot{\mathbf{q}}$. In order to keep the syntax simple, we focus on the case where $\mathbf{x} = [\mathbf{q}^\mathsf{T}\ \dot{\mathbf{q}}^\mathsf{T}]^\mathsf{T}$ (so $n = 2q$); extending the optimization to additional state dimensions (such as muscle activation level or fiber length) can be done by defining the inverse dynamics of these quantities. The system is driven by a control signal $\mathbf{u} \in \mathbb{R}^m$; we are interested in the *underactuated* case ($m < n$), where some state dimensions can only be affected indirectly.

Given a general-form cost function $\ell(\mathbf{x}, \mathbf{u})$, we seek to compute the locally-optimal N-step trajectory which minimizes the total cumulative cost $\sum_k \ell(\mathbf{x}^k, \mathbf{u}^k)$. This optimization takes place over the minimally-represented search space $\mathbf{Q} = \text{stack}\{\mathbf{q}^k\} \in \mathbb{R}^{qN}$.

In a semi-implicit integration (section 2.4.1), the dynamics express a relation between the current state $(\mathbf{q}^k, \dot{\mathbf{q}}^k)$ and the velocity at the next time-step $\dot{\mathbf{q}}'$ (compare to equation 2.11a):

$$\dot{\mathbf{q}}' = \dot{\mathbf{q}} + hM^{-1}(\mathbf{r} + \hat{\mathbf{u}}), \tag{6.1}$$

where h is the time-step, $\mathbf{q}$ is the configuration, $M = M(\mathbf{q})$ is the mass-matrix, $\mathbf{r} = \mathbf{r}(\mathbf{q}, \dot{\mathbf{q}})$ is the vector of total Coriolis, centripetal and other intrinsic forces, and $\hat{\mathbf{u}}$ is the *full* control (i.e., the control signal in the fully-actuated system). During forward simulation (2.11a), $\hat{\mathbf{u}} = B\mathbf{u}$, where B is the $n \times m$ matrix determining which degrees of freedom are actuated.

The use of the full control $\hat{\mathbf{u}}$ is unavoidable during inverse dynamics — since we explicitly represent the entire trajectory, we must cope with trajectories that are infeasible in the underactuated system; however, the system controlled by $\hat{\mathbf{u}}$ is fully-actuated, and therefore every trajectory is dynamically-consistent by definition.

The full state-space trajectory can be easily recovered from a minimal representation. However, identifying the corresponding control sequence is not as simple. Forward dynamics answers the question "given the current position, velocity, and control, what is the next position and velocity?", mapping from $(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u})$ to $\dot{\mathbf{q}}'$ (where `prime` ($'$) stands for "next time-step" as in equation 6.1); inverse dynamics asks the question "given the current position and velocity, as well as the *next* velocity, what is the control that is applied?", mapping from $(\mathbf{q}, \dot{\mathbf{q}}, \dot{\mathbf{q}}')$ to $\hat{\mathbf{u}}$.

It is convenient to define the *expanded* representation $\boldsymbol{\varphi}$ as a concatenation of these vectors: $\boldsymbol{\varphi} \triangleq [\mathbf{q}^\mathsf{T}\ \dot{\mathbf{q}}^\mathsf{T}\ \dot{\mathbf{q}}'^\mathsf{T}]^\mathsf{T}$. Note that the expanded representation $\boldsymbol{\varphi}$ is a linear function of three consecutive positions along a minimally-represented trajectory:

$$\boldsymbol{\varphi}' = \Upsilon[\mathbf{q}^\mathsf{T}\ \mathbf{q}'^\mathsf{T}\ \mathbf{q}''^\mathsf{T}]^\mathsf{T},$$

where:

$$\Upsilon = \begin{bmatrix} 0 & \mathbf{I} & 0 \\ -\mathbf{I}/h & \mathbf{I}/h & 0 \\ 0 & -\mathbf{I}/h & \mathbf{I}/h \end{bmatrix} \tag{6.2}$$

6.3.3 Penalizing helper forces

The inverse dynamics computes the full control $\hat{\mathbf{u}}(\boldsymbol{\varphi})$. In order to identify the control of the original underactuated system, we decompose $\hat{\mathbf{u}}$ into the sum of two terms, one representing the standard actuation and one representing the *helper* forces that apply to the originally-unactuated dimensions. Given the m-rank control matrix B of the original underactuated system, we can construct a second matrix $\breve{B}$ whose columns span the kernel of B. This allows us to uniquely decompose $\hat{\mathbf{u}}$:

$$\hat{\mathbf{u}}(\boldsymbol{\varphi}) = B\mathbf{u}(\boldsymbol{\varphi}) + \breve{B}\breve{\mathbf{u}}(\boldsymbol{\varphi}). \tag{6.3}$$

We may now use the original (underactuated) control signal $\mathbf{u}$ in the computation of $\ell(\mathbf{x}, \mathbf{u})$ as before. In order to prevent reliance on helper forces, we use a penalty method and compute a cost term on $\breve{\mathbf{u}}$ which constrains it to 0; in the experiments below we use a quadratic cost $\frac{1}{2}\gamma||\breve{\mathbf{u}}||^2$, resulting in the performance criterion:

$$\beta(\boldsymbol{\varphi}) = \ell\big(\mathbf{x}(\boldsymbol{\varphi}), \mathbf{u}(\boldsymbol{\varphi})\big) + \frac{1}{2}\gamma||\breve{\mathbf{u}}(\boldsymbol{\varphi})||^2. \tag{6.4}$$

where $\mathbf{x}(\boldsymbol{\varphi})$ is shorthand for truncating the next velocity from $\boldsymbol{\varphi}$, leaving only the current state $\mathbf{x} = [\mathbf{q}^\mathsf{T}\ \dot{\mathbf{q}}^\mathsf{T}]^\mathsf{T}$ (section 6.3.2).

6.3.4 Invariant coordinates

In order to optimize a limit cycle, $\mathbf{Q}$ must represent a closed trajectory. However, the case of locomotion presents a challenge — when all generalized coordinates are considered, locomotion is not a limit cycle but rather a coil-shaped trajectory, because the global positions increase with every period.

In the previous chapters of this dissertation, this complication is avoided by simply excluding those coordinates from the state space, keeping only the corresponding velocity. This poses no problems, because the dynamics of the system is *invariant* in these dimensions, and so we may arbitrarily set the values of these coordinates to 0 during forward dynamics.

However, here we cannot do that, because the velocities are implicitly represented by the positions $\mathbf{q}$. We solve this by modifying $\mathbf{Q}$ to represent the velocities of these coordinates. The size of $\mathbf{Q}$ remains the same, but the transformation matrix Υ is modified to copy the velocity explicitly from $\mathbf{Q}$ to $\boldsymbol{\varphi}$, and insert 0 at the entries of $\boldsymbol{\varphi}$ which correspond to the coordinates of global position. In a planar simulation, the only invariant coordinate is the horizontal position; in 3D, both planar positions are invariant.

6.3.5 Inverse dynamics with contacts

The SLCP contact model presented in section 2.4.2 cannot be used for inverse dynamics, because the contact forces are computed through a non-convex optimization procedure, which is not easily invertible. In order to overcome this difficulty, we use a different algorithm for computing reaction forces, which is based on convex optimization [Todorov, 2011b], and can be transformed to compute the inverse equation.

The SLCP reaction force is found by minimizing a residual which is a smooth function of the state. The model presented here follows a similar structure, but replaces the complementarity condition of SLCP with a more relaxed formulation.

Given the current position $\mathbf{q}$, the geometric computations in MuJoCogenerate J, the Jacobian between state coordinates and contact coordinates. We use this Jacobian to introduce the reaction force $\mathbf{p}$ (specified in contact coordinates) to equation 6.1:

$$\dot{\mathbf{q}}' = \dot{\mathbf{q}} + hM^{-1}(\mathbf{r} + \hat{\mathbf{u}}) + M^{-1}J^{\mathsf{T}}\mathbf{p}.$$

Note that the last term is *impulsive*, in that it is not multiplied by the time step h. Using J again to project this equation to contact space, we get:

$$J\dot{\mathbf{q}}' = \mathbf{c} + A\mathbf{p},$$

where $\mathbf{c} = J\big(\dot{\mathbf{q}} + hM^{-1}(\mathbf{r} + B\mathbf{u})\big)$ is the velocity in contact space when $\mathbf{p} = 0$, and $A = JM^{-1}J^{\mathsf{T}}$ is the inverted mass matrix in contact coordinates. A *forward* contact model maps from $(\mathbf{c}, A)$ to $\mathbf{p}$ (and therefore $\dot{\mathbf{q}}'$); an *inverse* contact model maps from $(\dot{\mathbf{q}}', A)$ to $\mathbf{p}$ (and therefore $\hat{\mathbf{u}}$).

The contact model of Todorov [2011b] solves for $\mathbf{p}$ by seeking the reaction force that minimizes kinetic energy in contact space. Following the derivation presented there, this condition is equivalent to minimizing a residual of the form $r(\mathbf{p}) = \frac{1}{2}\mathbf{p}^{\mathsf{T}}(A{+}R)\mathbf{p}{+}$ $\mathbf{p}^{\mathsf{T}}\mathbf{c}$, where $R = R(J\mathbf{q})$ is a regularization matrix which serves for ensuring (relaxed) complementarity: R is a diagonal matrix whose elements are a function of the current distance in contact coordinates. When the distance along one contact dimension is big, the corresponding terms in R are very big, and therefore the corresponding terms in $\mathbf{p}$ reduce to 0; when the contact distance is small (or negative, indicating penetration), the corresponding terms in R go to zero, and the contact model is free to eliminate all kinetic energy in that dimension. The friction cone constraints are translated to log-barrier functions $d(\mathbf{p})$, and the problem is solved with a modified interior-point method.

As Todorov [2011b] shows, this formulation can be inverted: since $\mathbf{p}$ minimizes $r + d$, it must satisfy $r_{\mathbf{p}} + d_{\mathbf{p}} = 0$. Through some further analytic manipulation, we can construct a residual expression for $\mathbf{p}$ that depends only on A and $J\dot{\mathbf{q}}'$. Unsurprisingly, this residual [Todorov, 2011b, equation 23] includes the term $\frac{1}{2}\mathbf{p}^\mathsf{T} R\mathbf{p}$, allowing R to manifest a relaxed complementarity condition as before. The parametric form of R is not dictated by the theory; in our experiments, we got the most realistic-looking results with R being exponentially-dependent on the contact distance.

6.4 Inverse dynamics optimization

We seek to minimize the total cost $\boldsymbol{\beta} = \sum_k \beta^k(\boldsymbol{\varphi}^k)$ by computing $\boldsymbol{\beta}_{\mathbf{Q}}$ and $\boldsymbol{\beta}_{\mathbf{QQ}}$, its derivatives with respect to our minimal representation of the trajectory.

Using the expanded representation $\boldsymbol{\varphi}$ (section 6.3.2), we define the stack of expanded representation vectors $\boldsymbol{\Phi} \triangleq \text{stack}\{\boldsymbol{\varphi}^k\} \in \mathbb{R}^{3qN}$, which is a linear function of the minimal representation $\mathbf{Q}$:

$$\boldsymbol{\Phi} = \Lambda \mathbf{Q}, \tag{6.5}$$

where the matrix $\Lambda \in \mathbb{R}^{qN \times 3qN}$ is made of adjacent copies of Υ (6.2).

We can compute the gradient and Hessian of β with respect to $\boldsymbol{\varphi}$ by finite-differencing the inverse-dynamics function $\mathbf{u}(\boldsymbol{\varphi})$ and applying the chain rule. This provides a quadratic model for β for every time step k along the trajectory:

$$\beta(\boldsymbol{\varphi}^k + \delta\boldsymbol{\varphi}) \approx \beta_0^k + \delta\boldsymbol{\varphi}^\mathsf{T} \beta_{\boldsymbol{\varphi}}^k + \frac{1}{2}\delta\boldsymbol{\varphi}^\mathsf{T} \beta_{\boldsymbol{\varphi}\boldsymbol{\varphi}}^k \delta\boldsymbol{\varphi}.$$

We stack the gradients $\boldsymbol{\beta}_{\boldsymbol{\Phi}} = \text{stack}\{\beta^k_{\boldsymbol{\varphi}}\}$ and arrange the quadratic terms in the block-diagonal matrix $\boldsymbol{\beta}_{\boldsymbol{\Phi\Phi}} = \text{diag}\{\beta^k_{\boldsymbol{\varphi\varphi}}\}$ to obtain a quadratic model for $\boldsymbol{\beta}$ over the expanded representation:

$$\boldsymbol{\beta}(\boldsymbol{\Phi} + \delta\boldsymbol{\Phi}) \approx \boldsymbol{\beta}_0 + \delta\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\beta}_{\boldsymbol{\Phi}} + \frac{1}{2}\delta\boldsymbol{\Phi}^{\mathsf{T}}\boldsymbol{\beta}_{\boldsymbol{\Phi\Phi}}\delta\boldsymbol{\Phi}$$

We can now compress the derivatives of $\boldsymbol{\beta}$ with respect to the expanded representation to their equivalents in the minimal representation using the matrix Λ (6.5):

$$\boldsymbol{\beta}_{\mathbf{Q}} = \Lambda^{\mathsf{T}}\boldsymbol{\beta}_{\varphi} \tag{6.6a}$$

$$\boldsymbol{\beta}_{\mathbf{QQ}} = \Lambda^{\mathsf{T}}\boldsymbol{\beta}_{\varphi\varphi}\Lambda. \tag{6.6b}$$

6.4.1 Compressed representation

Equations 6.6 use the sparse matrix Λ to linearly transform a quadratic model in the expanded representation $\boldsymbol{\Phi}$ to the corresponding quadratic model in the minimal representation $\mathbf{Q}$. This reduction in dimensionality is information-preserving, because the expanded representation is only a notational convenience. However, this manipulation suggests that a similar manipulation can be used to allow optimization with compressed representation that is smaller than $\mathbf{Q}$, as long as it is linearly-expandable to $\mathbf{Q}$. This criterion is met by many low-dimensional representations of closed trajectories, such as Fourier transform and splines, and this type of dimensionality reduction is common in simultaneous optimization [e.g., Bessonnet et al., 2010, Anderson and Pandy, 2001].

In the experiments described below we use a Fourier series representation for compression. In one dimension, we can linearly expand a vector $\mathbf{t} \in \mathbb{R}^s$ (representing the first s coefficients in a Fourier series) to the N-vector $F\mathbf{t}$ representing the full periodic trajectory (where the Fourier-transform matrix $F \in \mathbb{R}^{N\times s}$ can be obtained, for example, by taking the first s rows of an FFT of the identity matrix). Given $\mathbf{S} \in \mathbb{R}^{sq}$, the Fourier series of all state dimensions, we write: $\mathbf{Q} = \mathcal{F}\mathbf{S}$, where $\mathcal{F} \in \mathbb{R}^{sq\times Nq}$ is the full-trajectory Fourier-transform matrix, built from stacked copies of the matrix

F. This martix allows us to write the derivatives of $\boldsymbol{\beta}$ with respect to the Fourier representation:

$$\boldsymbol{\beta}_{\mathbf{S}} = \mathcal{F}^{\mathsf{T}}\boldsymbol{\beta}_{\mathbf{Q}} \tag{6.7a}$$

$$\boldsymbol{\beta}_{\mathbf{SS}} = \mathcal{F}^{\mathsf{T}}\boldsymbol{\beta}_{\mathbf{QQ}}\mathcal{F}. \tag{6.7b}$$

Although this is a lower-dimensional representation, the computational cost of every iteration remains almost the same, because the derivatives of the inverse dynamics still require finite-differencing of the expanded representation $\mathbf{\Phi}$. Note that this method can only find the locally-optimal trajectory within the limited subspace of all trajectories; the optimal unconstrained trajectory $\mathbf{Q}^*$ still has a lower cost than $\mathbf{S}^*$.

6.4.2 Symmetry

In domains of bipedal locomotion, we expect the optimal limit cycle to be symmetric — the starting position of the swing leg should match the ending position of the stance leg, and vice versa. This constraint allows us to optimize for only one step, instead of a full stride. We enforce this constraint by applying a linear transformation (swapping the sides of the body) to the last state $\mathbf{x}^N$ whenever it is considered in conjunction with the first state $\mathbf{x}^1$.

6.5 Results

We demonstrate the power of inverse optimization by optimizing a running gait for a simulated 3D bipedal robot, illustrated in figure 6.1. The model has 31 degrees of freedom: the ball joint between the torso and the pelvis has 3 DOF (the head is welded to the torso), each leg has 3 DOF at the hip, 1 at the knee and 3 at the ankle, and each arm has 3 DOF at the shoulder and 1 at the elbow. The physical properties of the body parts are specified in table 6.1.

Every foot has three contact points with the ground: one at the back, and two at the front. In this experiment we use $N = 21$ timesteps of 15msec, which correspond

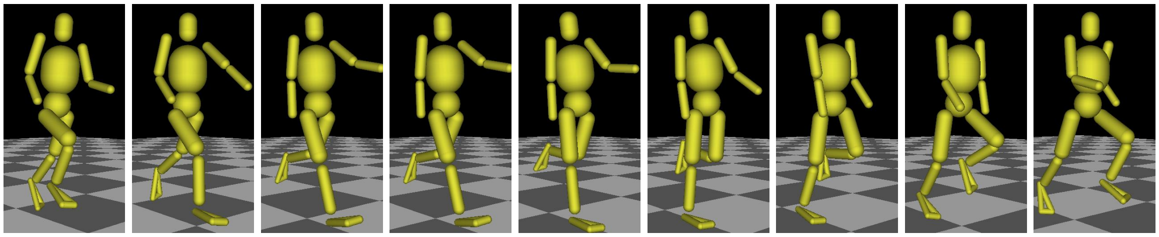


Figure 6.1: A sequence of frames depicting the optimal running gait; this gait can be seen in a movie available at http://www.youtube.com/watch?v=CL11JTnH6P8. Note the extension of the arm in frame 2, coinciding with the flight phase. This arm motion emerged in the optimal solution as a way to eliminate excessive angular momentum.

to $\sim$ 190 steps per minute, in accordance with standard human running pace. The Fourier representation uses $s = 11$, in accordance with Ren et al. [2007].

The cost function of the running task compares the horizontal velocity of the runner's center of mass $\dot{\mathbf{q}}_y$ to some desired velocity, while maintaining all body parts at a forward orientation:

$$\ell(\mathbf{x}, \mathbf{u}) = 100(\dot{\mathbf{q}}_y - v_d)^2 + ||\mathbf{q}_{xz}|| + 0.01||\mathbf{u}||^2$$

where $\mathbf{q}_{xz}$ is a vector of all joint angles in the $x-$ and $z-$axes, and the last term is a quadratic cost on applied joint torques. The penalty term on $\mathbf{q}_{xz}$ is required because we are using direct torque control, and in the absence of such penalty, the runner tries to distribute the actuation across all spatial angles equally, resulting in a hideously-unrealistic gait. In our simulation we use a set velocity of $v_d = 6$m/s, and the resulting optimal running gait reaches an average velocity of 5.1m/s.

The limit cycle is initialized by setting the first state to a physically-unrealistic open-leg pose elevated above the ground. Due to the symmetrical formulation and the Fourier representation, the initial limit cycle is an interpolation between this pose and its mirror image, which looks like a prototypical leg swing motion. The optimization process converges after about 2500 Newton-step iterations, which take about 10 minutes on a desktop computer with two 6-core processors. The most computationally-intensive step is finite-differencing the inverse dynamics. However, this computation can be executed in parallel, saving a lot of computation time.

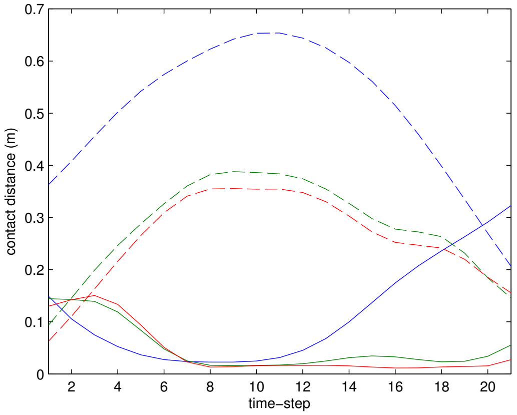


Figure 6.2: Distance in contact space between feet and ground.

The resulting gait can be seen in a movie[10]. Figure 6.2 shows the distances, in contact space, of the feet's contact points during one step. The heel strikes at time-step 6, and the toes soon follow at step 8, which makes this a heel-first running gait. With the heel leaving the ground at step 12, and toe-off at time 21, the flight phase extends between time-steps 1 and 5, making it a veritable running gait.

The most striking feature of the resulting running gait is the emergent coordination between the legs and the opposite arms. The left arm thrusts forward in synchrony with the right leg during the flight phase (best seen in frame 2 of figure 6.1), eliminating the angular momentum of the pelvis and torso that would otherwise result in an increased cost. This feature came about without any form of explicit modeling, and attests to the power of the optimization process. The arm also exhibits a minor jerk in the anti-phase of the forward thrust; we are not sure what causes that motion.

6.6 Discussion

This chapter uses an invertible smooth contact model to find an optimal trajectory in a high-dimensional domain. The formulation we use is completely general, and requires no hybrid representation. While we demonstrate this algorithm in a domain of humanoid bipedal running, the algorithm we present has general applicability, and

[10] Available at http://www.youtube.com/watch?v=CL11JTnH6P8

body	length (cm)	radius (cm)	mass (kg)
trunk	10	18	28.5
head	10	8.5	4.2
pelvis	13	n/a	9.2
thigh	35	7.4	7.3
shin	31	5	2.8
foot	27.4	3	2.3
upper arm	30	5	2.7
lower arm	26	4	1.5

Table 6.1: Morphological specification of the 3D runner. All bodies but pelvis are capsules. Total height: 1.9m (∼6'3"); total weight: 75.25kg (∼166 lbs.).

the very same optimization can be applied to a diverse set of tasks, from multi-legged locomotion to dexterous hand manipulation.

The resulting optimal motion shares some features of human bipedal running. However, biomechanical realism is not the primary goal of this project. We are certain that further tweaking of the cost function and the body anthropometry can lead to running gaits that more faithfully imitate human running. Yet, such biomimicry is contingent on the development of algorithms capable of optimizing behavior in biomechanically-realistic domains (which are often high-dimensional) in the presence of contacts. The algorithm presented in this paper meets these requirements.

We are not explicitly representing the stochasticity of the domain. However, following the theory of Stochastic LCP (section 2.4.2) the smooth contact dynamics can be construed as an approximation for the expected reaction forces under stochastic dynamics.

In order to control a real-world system (or a stochastic simulation), the Hessian around the optimal trajectory can be used to derive a locally-linear feedback controller [Tassa, Erez, and Todorov, 2011a]. By invoking the *separation* principle [Stengel, 1994], the feedback controller optimized for the deterministic domain can serve to control its stochastic counterpart, as long as the noise profile is not dependent on the state.

We take a model-based approach to optimizing behavior, which allows for efficient optimization. In contrast, model-free methods construct an approximation of the dynamics (either in forward or inverse formulation) from interaction with the environment. This model-free approach cannot be directly applied to tasks which require powerful articulated motion, and result in inherently unstable behavior. On the other hand, it is impossible to build a perfectly accurate model of a multi-joint robot. Therefore, in order to apply such model-based methods to control real robots, some mechanism for model adaptation and learning must be incorporated. One example for such a method is presented by Abbeel et al. [2007]. In addition, some tolerance for modeling errors must be incorporated into the feedback controller; this can be achieved, for example, using model-predictive control, as in the previous chapter.

Finally, the algorithm presented here requires that we fix the period of the limit-cycle in advance; this could be relaxed by allowing the time step length h to be optimized as well, adding one dense row to the Hessian.

Chapter 7

Conclusion and Future Work

This dissertation presents a set of algorithms for the computation of locally-optimal trajectories in nonlinear, underactuated dynamical systems. The dynamics is treated as a black box, without applying analytic manipulation to the time-stepping function **f**, nor to its inverse dynamics counterpart. The use of general-form cost functions and black-box dynamics guarantees the generality and extensibility of these methods.

The use of smooth contact models allows us to apply standard optimization tools to domains with multiple contacts without having to explicitly represent or prescribe the contact times. This means that complex motion patterns with unpredictable number of simultaneous contacts can be solved just as easily.

The significance of this work lies at its outlining of a new kind of tradeoff in nonlinear motor control design. Traditionally, research in control theory focuses on finding provably stable policies, which are indispensable for mission-critical tasks such as missile control. Meanwhile, the study of optimal control, adaptive dynamic programming, and reinforcement learning, all focus on methods which can produce globally optimal behavior, but require the computational consideration of the entire volume of state space.

In contrast, if biological realism is an accepted design goal, then the expectations we have from our controllers can be more relaxed than those of the control engineer. The unavoidable eventual failure of any specific natural behavior (we all fall sometime) makes the design of truly robust controllers a futile endeavor. By forfeiting global guarantees, we can apply optimization to otherwise prohibitively-hard domains, and generate creative and robust motor behavior using standard computational tools.

This rich set of behaviors comes about without the need to explicitly pre-define the details of every motion pattern; instead, these emerge from first principles through numerical optimization.

Beyond the immediate implications for robotic research, I see autonomous dynamic simulation as a tool that may be applied to the investigation of a wide variety of natural phenomena. Motor behavior comes about as the result of an interplay between the actuated body, the physical environment, and the underlying control mechanism. By developing algorithms that optimize autonomous motor behavior, we may address a variety of what-if questions that cannot be studied in real life, since in our simulated domain we can independently alter the physics, morphology and control. Going forward, my goal is to develop general tools that can support investigation across a range of biomechanical phenomena.

In *computational biomechanics*, such optimization tools can allow better prediction of post-treatment outcome in patient-specific models, providing surgeons and therapists with a virtual platform for the investigation of various scenarios. In *neuroscience*, studies of motor control often attempt to answer the question "why we move the way we do" by building a normative model, explaining an observed behavior in terms of some first principles (e.g., minimum torque). Dynamic simulations offer a concrete method to test hypotheses regarding potential first principles, allowing for the analysis and explanation of experimental data in a concrete fashion. This tool may also be applied to investigations in *evolutionary biology*, tracing the role of various adaptations by investigating counter-factual models of alternative morphologies.

Efficient optimization algorithms can also be used for meta-optimization in the space of morphological configurations. This is an open-ended field of research, with countless possible applications, from prosthesis design and sport medicine to theoretical biology.

One computational feature of biological neural processing that is not addressed in this dissertation is dimensionality reduction. While the algorithms presented here treat equally every dimension of state space, there is good reason to expect a more hierarchical organization in neural systems that control an overactuated system of many overlapping muscles, and perhaps distinguish between dimensions with immediate relevance to the task, and dimensions who only play a supportive role.

In my mind, the biggest open question left out of this dissertation is the problem of inverse optimal control. A long-standing assumption of biology is that the current state of nature is in part the result of an optimization process, and this dissertation provides tools for recreating this optimization given some cost function. However, there is no scientific consensus about what are the relevant first principles underlying the various categories of motor behavior. The application of broad principles to detailed biomechanical models must be mediated by some concrete hypotheses; for example, the principle of minimizing metabolic effort can be modeled by minimizating of muscle mechanical work, or joint torques, or some operational definition of fatigue. These questions are often the locus of much scientific debate, and the variety of proposed optimality criteria is a challenge that must be dealt with when applying optimal control in these domains.

Understanding the optimization criteria that may underlie observed natural phenomena can provide important insights into our objects of study, as well as the biophysics which both constrains and affords their behavior. As such, I believe that these tools hold great promise to the study of natural motion.

References

P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, page 1, 2007.

M. Ackermann and A. J. V. den Bogert. Optimality principles for model-based prediction of human gait. *Journal of biomechanics*, 2010.

E. L. Allgower and K. Georg. *Introduction to numerical continuation methods*. SIAM, 2003. ISBN 9780898715446.

F. A. Almeida. Waypoint navigation using constrained infinite horizon model predictive control. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008.

F. C. Anderson and M. G. Pandy. Dynamic optimization of human walking. *Journal of Biomechanical Engineering*, 123(5):381–390, Oct. 2001. doi: 10.1115/1.1392310. URL http://link.aip.org/link/?JBY/123/381/1.

H. Attias. Planning by probabilistic inference. In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, 2003.

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15(4):319–350, 2001.

D. P. Bertsekas. Dynamic programming and suboptimal control: A survey from ADP to MPC. *European Journal of Control*, 11(4-5):310–334, 2005. ISSN 0947-3580.

G. Bessonnet, J. Marot, P. Seguin, and P. Sardain. Parametric-Based dynamic synthesis of 3D-Gait. *Robotica*, 28(04):563–581, 2010. doi: 10.1017/S0263574709990257. URL http://journals.cambridge.org/action/displayAbstract?fromPage=online&aid=7792574.

H. G. Bock and K. J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *Proceedings of the 9th IFAC world congress*, 1984.

A. E. Bryson and Y. C. Ho. *Applied optimal control*. Wiley New York, 1975. ISBN 0470114819.

H. Chen and F. Allgower. A Quasi-Infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica*, 34(10):1205–1217, 1998. doi: 10.1016/S0005-1098(98)00073-9. URL http://www.sciencedirect.com/science/article/B6V21-3VCT7RM-5/2/e304a7046ed4bc72e4b4ff0f03ceabde.

C. Chevallereau, G. Abba, Y. Aoustin, F. Plestan, E. Westervelt, C. Canudas-de-Wit, and J. Grizzle. RABBIT: a testbed for advanced control theory. *IEEE Control Systems Magazine*, 23(5):57–79, 2003.

R. Cottle. *The Linear Complementarity Problem*. Academic Press, Boston, 1992. ISBN 0121923509.

R. Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002. URL citeseer.ist.psu.edu/coulom02reinforcement.html.

M. Diehl, H. G. Bock, J. P. Schloder, R. Findeisen, Z. Nagy, and F. Allgower. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577, 2002.

M. Diehl, H. Ferreau, and N. Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In *Nonlinear Model Predictive Control*, page 391. 2009.

K. Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

J. C. Dunn and D. P. Bertsekas. Efficient dynamic programming implementations of Newton's method for unconstrained optimal control problems. *Journal of Optimization Theory and Applications*, 63(1):23–38, 1989. ISSN 0022-3239.

T. Erez and W. D. Smart. What does shaping mean for computational reinforcement learning? In *7th IEEE International Conference on Development and Learning (ICDL)*, pages 215–219, 2008.

T. Erez and W. D. Smart. Coupling perception and action using minimax optimal control. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 58–65, Nashville, TN, USA, 2009. doi: 10.1109/ADPRL.2009.4927526. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4927526.

T. Erez and W. D. Smart. A scalable method for solving high-dimensional continuous pomdps using local approximation. In *Proceedings of Uncertainty in AI*, 2010. URL http://event.cwi.nl/uai2010/papers/UAI2010_0092.pdf.

T. Erez and E. Todorov. Inverse dynamics optimal control for domains with contacts (under review for AAAI), 2011.

T. Erez, Y. Tassa, and E. Todorov. Infinite-horizon model predictive control for periodic tasks with contacts (under review for R:SS), 2011.

A. Gosavi. A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis. *Machine Learning*, 55(1):5–29, 2004. ISSN 0885-6125.

D. R. Hofstadter. *Godel, Escher, Bach: An Eternal Golden Braid*. Basic Books, 1979. ISBN 0140179976.

D. R. Hofstadter. *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, Inc., New York, NY, USA, 1996. ISBN 0465024750.

B. Hu and A. Linnemann. Toward infinite-horizon optimality in nonlinear model predictive control. *IEEE Transactions on Automatic Control*, 47(4):679–682, 2002. doi: 10.1109/9.995049. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=995049.

D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Programming*. Elsevier, 1970.

M. I. Jordan. *Learning in graphical models*. Kluwer Academic Publishers, 1998.

B. Kappen, V. Gomez, and M. Opper. Optimal control as a graphical model inference problem. *arXiv*, 2009. URL http://arxiv.org/pdf/0901.0633.

H. J. Kappen. Path integrals and symmetry breaking for optimal control theory. *Journal of statistical mechanics: theory and experiment*, 2005.

H. J. Kim, Q. Wang, S. Rahmatalla, C. C. Swan, J. S. Arora, K. Abdel-Malek, and J. G. Assouline. Dynamic motion planning of 3D human locomotion using gradient-based optimization. *Journal of Biomechanical Engineering*, 130(3):031002, June 2008. ISSN 0148-0731. doi: 10.1115/1.2898730. URL http://www.ncbi.nlm.nih.gov/pubmed/18532851. PMID: 18532851.

G. Lantoine and R. P. Russell. A hybrid differential dynamic programming algorithm for robust low-thrust optimization. In *AAS/AIAA Astrodynamics Specialist Conference and Exhibit*, 2008.

L.-Z. Liao and C. A. Shoemaker. Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems. Technical Report 92-097, Cornell Theory Center, 1992.

S. Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22:159–196, 1996.

D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 2000. ISSN 0005-1098.

S. K. Mitter and N. J. Newton. A variational approach to nonlinear estimation. *SIAM Journal on Control and Optimization*, 42(5):1813–1833, 2004.

E. Mizutani and S. Dreyfus. Stagewise Newton, differential dynamic programming, and neighboring optimum control for neural-network learning. In *Proceedings of the 2005 American Control Conference*, pages 1331–1336, Portland, OR, USA, 2005. doi: 10.1109/ACC.2005.1470149.

K. Mombaur, A. Truong, and J.-P. Laumond. From human to humanoid locomotionan inverse optimal control approach. *Autonomous Robots*, 28:369–383, 2010. URL http://dx.doi.org/10.1007/s10514-009-9170-7.

M. Morari and J. H. Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4-5):667–682, 1999. ISSN 0098-1354.

U. Muico, Y. Lee, J. Popovic, and Z. Popovic. Contact-aware nonlinear control of dynamic characters. *ACM Transactions on Graphics*, 28(3):1, 2009. ISSN 07300301. doi: 10.1145/1531326.1531387. URL http://portal.acm.org/citation.cfm?doid=1531326.1531387.

R. Munos and A. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, pages 1348–1355, Stockholm, Sweden, 1999. Morgan Kaufmann Publishers Inc. URL http://portal.acm.org.offcampus.lib.washington.edu/citation.cfm?id=1624410.

K. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of Uncertainty in AI*, page 467–475, 1999.

G. Pannocchia, S. J. Wright, and J. B. Rawlings. Existence and computation of infinite horizon model predictive control with active steady-state input constraints. *IEEE Transactions on Automatic Control*, 48(6):1002–1006, 2003. doi: 10.1109/TAC.2003.812783. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1205193.

L. Ren, R. K. Jones, and D. Howard. Predictive modelling of human walking over a complete gait cycle. *Journal of Biomechanics*, 40(7): 1567–1574, 2007. ISSN 0021-9290. doi: 10.1016/j.jbiomech.2006.07.017. URL http://www.sciencedirect.com/science/article/B6T82-4M6SG7S-1/2/78e348be9b5a8db36f5808f07e3aeeaa.

M. Riedmiller, J. Peters, and S. Schaal. Evaluation of policy gradient methods and variants on the Cart-Pole benchmark. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 254–261, Honolulu, HI, USA, 2007. doi: 10.1109/ADPRL.2007.368196. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4220841.

A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, volume 298, page 305, 1993.

S. P. Singh. Reinforcement learning algorithms for average-payoff Markovian decision processes. In *Proceedings of the twelfth national conference on Artificial intelligence*, pages 700–705, 1994. ISBN 0262611023.

R. F. Stengel. *Optimal Control and Estimation*. Dover Publications, Sept. 1994. ISBN 0486682005.

M. Stolle and C. Atkeson. Policies based on trajectory libraries. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Y. Tassa and T. Erez. Least squares solutions of the HJB equation with neural network Value-Function approximators. *IEEE Transactions on Neural Networks*, 18 (4):1031–1041, 2007. ISSN 1045-9227. doi: 10.1109/TNN.2007.899249. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4267720.

Y. Tassa and E. Todorov. Stochastic complementarity for local control of discontinuous dynamics. In *Proceedings of Robotics: Science and Systems (RSS)*, 2010.

Y. Tassa, T. Erez, and W. Smart. Receding horizon differential dynamic programming. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, page 1465. MIT Press, Cambridge, MA, 2008.

Y. Tassa, T. Erez, and E. Todorov. Optimal limit-cycle control recast as bayesian inference (to appear in IFAC), 2011a.

Y. Tassa, T. Erez, and E. Todorov. Fast model predictive control for reactive robotic swimming (under review for R:SS), 2011b.

R. Tedrake, T. W. Zhang, and H. S. Seung. Stochastic policy gradient reinforcement learning on a simple 3D biped. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2849–2854, 2004. ISBN 0780384636.

E. Theodorou, J. Buchli, and S. Schaal. reinforcement learning of motor skills in high dimensions: a path integral approach. In *international conference of robotics and automation (icra 2010)*, 2010. URL http://www-clmc.usc.edu/publications/T/theodorou-ICRA2010.pdf.

E. Todorov. Linearly-solvable markov decision problems. *Advances in neural information processing systems*, 19:1369, 2007.

E. Todorov. General duality between optimal control and estimation. In *proceedings of the 47th IEEE conference on decision and control*, 2008.

E. Todorov. Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences*, 106(28):11478, 2009.

E. Todorov. MuJoCo: A physics engine for model-based control (under review), 2011a. URL http://www.cs.washington.edu/homes/todorov/papers/MuJoCo.pdf.

E. Todorov. A convex, smooth and invertible contact model for trajectory optimization (accepted to ICRA), 2011b. URL http://www.cs.washington.edu/homes/todorov/papers/contactConvex.pdf.

E. Todorov and W. Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the American Control Conference.*, pages 300–306, Portland, OR, USA, 2005. doi: 10.1109/ACC.2005.1469949.

M. Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

K. Wampler and Z. Popovic. Optimal gait and form for animal locomotion. *ACM Transactions on Graphics (TOG)*, 28(3):1–8, 2009.

J. M. Wang, D. J. Fleet, and A. Hertzmann. Optimizing walking controllers. In *ACM Transactions on Graphics (TOG)*, SIGGRAPH Asia '09, page 168:1168:8, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-858-2. doi: 10.1145/1661412.1618514. ACM ID: 1618514.

C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge,England, 1989.

Y. Weiss and W. T. Freeman. Correctness of belief propagation in gaussian graphical models of arbitrary topology. *Neural computation*, 13(10):2173–2200, 2001.

A. Witkin and M. Kass. Spacetime constraints. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques - SIGGRAPH '88*, pages 159–168, 1988. doi: 10.1145/54852.378507. URL http://portal.acm.org/citation.cfm?doid=54852.378507.

J. Wu and Z. Popović. Terrain-adaptive bipedal locomotion control. *ACM Transactions on Graphics*, 29:72:1–72:10, July 2010. ISSN 0730-0301.

K. Yin, K. Loken, and M. van de Panne. SIMBICON: simple biped locomotion control. *ACM Transactions on Graphics*, 26(3):105, 2007. ISSN 07300301. doi: 10.1145/1276377.1276509. URL http://portal.acm.org/citation.cfm?doid=1276377.1276509.