# Efficient Learning and Inference for Highdimensional Lagrangian Systems 

Paul N. Vernaza<br>University of Pennsylvania, paul.vernaza@gmail.com

Follow this and additional works at: http://repository.upenn.edu/edissertations
Part of the Artificial Intelligence and Robotics Commons

## Recommended Citation

Vernaza, Paul N., "Efficient Learning and Inference for High-dimensional Lagrangian Systems" (2011). Publicly Accessible Penn Dissertations. 301.
http://repository.upenn.edu/edissertations/301

# Efficient Learning and Inference for High-dimensional Lagrangian Systems 


#### Abstract

Learning the nature of a physical system is a problem that presents many challenges and opportunities owing to the unique structure associated with such systems. Many physical systems of practical interest in engineering are high-dimensional, which prohibits the application of standard learning methods to such problems. This first part of this work proposes therefore to solve learning problems associated with physical systems by identifying their low-dimensional Lagrangian structure. Algorithms are given to learn this structure in the case that it is obscured by a change of coordinates. The associated inference problem corresponds to solving a high-dimensional minimum-cost path problem, which can be solved by exploiting the symmetry of the problem. These techniques are demonstrated via an application to learning from highdimensional human motion capture data. The second part of this work is concerned with the application of these methods to high-dimensional motion planning. Algorithms are given to learn and exploit the struc- ture of holonomic motion planning problems effectively via spectral analysis and iterative dynamic programming, admitting solutions to problems of unprecedented dimension com- pared to known methods for optimal motion planning. The quality of solutions found is also demonstrated to be much superior in practice to those obtained via sampling-based planning and smoothing, in both simulated problems and experiments with a robot arm. This work therefore provides strong validation of the idea that learning low-dimensional structure is the key to future advances in this field.


## Degree Type

Dissertation

## Degree Name

Doctor of Philosophy (PhD)

## Graduate Group

Electrical \& Systems Engineering

## First Advisor

Daniel D. Lee

## Keywords

machine learning, robotics, motion planning, learning dynamical systems

## Subject Categories

Artificial Intelligence and Robotics

# EFFICIENT LEARNING AND INFERENCE FOR HIGH-DIMENSIONAL LAGRANGIAN SYSTEMS 

Paul N. Vernaza

A DISSERTATION

## in

## Electrical and Systems Engineering

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2011

[^0]Roch Guerin, Professor<br>Electrical and Systems Engineering Graduate Group Chairperson

Dissertation committee

Daniel E. Koditschek, Professor<br>Electrical and Systems Engineering

J. Andrew Bagnell, Associate Professor

Robotics Institute Carnegie Mellon University

Daniel D. Lee, Associate Professor
Electrical and Systems Engineering

Maxim Likhachev, Asst. Research Prof. Robotics Institute Carnegie Mellon University

# EFFICIENT LEARNING AND INFERENCE FOR HIGH-DIMENSIONAL LAGRANGIAN SYSTEMS 

## COPYRIGHT

2011

Paul N. Vernaza

To my parents,
Jorge and Aura

## Acknowledgments

I am immeasurably grateful towards my many colleagues and friends, far too numerous to name here, to whom much of the credit is owed for making the GRASP laboratory an interesting and intellectually stimulating place to work. I would also like to express my gratitude towards the GRASP faculty, who have been supportive in every way of my efforts. I credit my dissertation committee for providing indispensable guidance concerning not only my dissertation, but also my career. I will forever be in the debt of my advisor, Daniel D. Lee, for taking a risk hiring me as a student, for being a brilliant mentor, and for pushing me to stay focused on my goals even when it was most difficult to do so.

I dedicate this work to my parents, Jorge and Aura, for making every sacrifice to ensure I was always afforded the opportunities that were never afforded them.

ABSTRACT<br>EFFICIENT LEARNING AND INFERENCE FOR HIGH-DIMENSIONAL LAGRANGIAN SYSTEMS<br>Paul N. Vernaza<br>Daniel D. Lee

Learning the nature of a physical system is a problem that presents many challenges and opportunities owing to the unique structure associated with such systems. Many physical systems of practical interest in engineering are high-dimensional, which prohibits the application of standard learning methods to such problems. This first part of this work proposes therefore to solve learning problems associated with physical systems by identifying their low-dimensional Lagrangian structure. Algorithms are given to learn this structure in the case that it is obscured by a change of coordinates. The associated inference problem corresponds to solving a high-dimensional minimum-cost path problem, which can be solved by exploiting the symmetry of the problem. These techniques are demonstrated via an application to learning from high-dimensional human motion capture data.

The second part of this work is concerned with the application of these methods to high-dimensional motion planning. Algorithms are given to learn and exploit the structure of holonomic motion planning problems effectively via spectral analysis and iterative dynamic programming, admitting solutions to problems of unprecedented dimension compared to known methods for optimal motion planning. The quality of solutions found is also demonstrated to be much superior in practice to those obtained via sampling-based planning and smoothing, in both simulated problems and experiments with a robot arm. This work therefore provides strong validation of the idea that learning low-dimensional structure is the key to future advances in this field.

## Contents

Acknowledgments ..... iv
1 Introduction ..... 1
1.1 The curse of dimensionality ..... 1
1.2 Learning dynamical systems ..... 2
1.3 Motion planning in high-dimensional spaces ..... 5
1.4 Overview ..... 7
1.5 Relation to published work ..... 7
I Learning structured Lagrangian models ..... 8
2 Learning dynamical systems ..... 9
2.1 Dynamical systems ..... 9
2.2 System identification ..... 11
2.3 Learning latent state ..... 12
2.3.1 Hidden Markov Models ..... 13
2.3.2 Gaussian Process Dynamical Models ..... 15
2.4 Learning in optimal control systems ..... 18
2.4.1 Reinforcement learning ..... 18
2.4.2 Inverse optimal control ..... 21
3 Learning structured Lagrangians ..... 23
3.1 Lagrangian mechanics ..... 24
3.1.1 Conservation laws and Noether's theorem ..... 25
3.2 Structured Lagrangians ..... 26
3.2.1 Conservation of energy ..... 27
3.2.2 Kinetic Lagrangians ..... 28
3.2.3 Geometric interpretation ..... 31
3.2.4 Fermat's principle of least time ..... 32
3.3 Fermat Components Analysis ..... 33
3.3.1 Spectral solution ..... 33
3.4 Estimation of the Lagrangian ..... 35
4 Inference for structured Lagrangians ..... 38
4.1 Inference by optimization ..... 38
4.2 Dynamic programming ..... 39
4.3 Symmetry of the value function ..... 41
4.4 Low-dimensional structure of optimal paths ..... 43
5 Discovering structure in physical data ..... 46
5.1 Problem statement ..... 46
5.2 Experiments with human motion capture data ..... 47
6 Future directions: non-kinetic Lagrangians ..... 55
II Planning with structured Lagrangians ..... 60
7 Motion planning in high-dimensional spaces ..... 61
7.1 Problem definition ..... 61
7.2 General methods ..... 63
7.2.1 Local optimization ..... 63
7.2.2 Potential fields ..... 63
7.2.3 Sampling-based planning ..... 64
7.2.4 Deterministic planning ..... 64
7.3 Discovering and exploiting structure ..... 65
7.3.1 Distance and heuristic functions ..... 65
7.3.2 MDP homomorphisms ..... 67
7.3.3 Symmetric optimal control systems ..... 67
8 Planning with structured cost functions ..... 69
8.1 Heuristics for deterministic search ..... 69
8.1.1 Properties of the Fermat heuristic ..... 70
8.1.2 The multi-agent towing problem ..... 73
8.1.3 Results ..... 74
8.2 Randomized planning ..... 75
8.2.1 Planning for a robot arm ..... 77
8.2.2 Results ..... 79
9 Learning cost structure ..... 82
9.1 Motivation ..... 82
9.2 Related work ..... 84
9.3 Spectral learning of cost structure ..... 85
9.3.1 Estimating the cost basis ..... 86
9.3.2 Compressing the cost function ..... 87
9.3.3 Planning a path ..... 88
9.3.4 A generalized shortcut heuristic ..... 88
9.3.5 Computational considerations ..... 89
9.4 Results ..... 89
9.4.1 Planning for a robot arm ..... 90
9.4.2 Planning for a deformable robot ..... 92
10 Learning dimensional descent ..... 98
10.1 Motivation ..... 98
10.2 Method ..... 100
10.2.1 Technical details ..... 101
10.3 Simulation results ..... 102
10.3.1 Deformable robot planning ..... 102
10.3.2 Arm planning ..... 103
10.4 Experimental results ..... 105
11 Assumptions, guarantees, and limitations ..... 112
11.1 SLASHDP ..... 112
11.2 LDD ..... 114
11.3 Fermat heuristic ..... 115
12 Future directions: structured planning ..... 118
12.1 Structured motion planning ..... 118
12.2 Structured discrete planning ..... 119
13 Conclusions ..... 121

## List of Figures

1.1 CMU motion capture data illustration ..... 3
1.2 Overview of learning from physical sequence data ..... 4
1.3 Trade-off between prior assumptions and dimensionality ..... 5
1.4 The PR2 robot ..... 6
2.1 A graphical model representation of a hidden Markov model ..... 14
2.2 A simple MDP ..... 19
3.1 Visualization of a kinetic Lagrangian with low-dimensional structure ..... 30
3.2 Illustration of Snell's law ..... 32
3.3 Interpretation of factorization provided by FCA and kinetic Lagrangians as a graphical model ..... 34
3.4 FCA applied to an inverse optics problem ..... 37
4.1 Illustration of Theorem 4.4.1 ..... 44
5.1 Graphical model associated with the GPDM ..... 47
5.2 Overview of method for sequence data modeling ..... 48
5.3 Reconstruction of novel trajectories from pairs of key frames ..... 49
5.4 Quantitative results of FCA reconstruction experiment ..... 51
5.5 A few views of an experiment reconstructing a novel action sequence ..... 52
5.6 Robot gait reconstruction experiment ..... 53
5.7 Quantitative results from gait reconstruction experiment ..... 54
6.1 Visualized result of Bayesian estimation of non-conservative force field with predicted trajectories ..... 58
6.2 Visualized result of Bayesian estimation of conservative force field with pre- dicted trajectories ..... 59
8.1 Illustration of Fermat heuristic computation ..... 71
8.2 Results of multi-robot planning experiments with deterministic planner. ..... 76
8.3 Illustration of low-dimensional structure in a simple arm planning problem ..... 77
8.4 Fermat heuristic: arm planning results ..... 80
8.5 Scaling performance of Fermat heuristic in arm planning ..... 81
9.1 Learned basis vectors for an arm planning problem ..... 83
9.2 Subjective comparison of different methods applied to a 36-dimensional arm planning problem ..... 91
9.3 Quantitative evaluation of SLASHDP on arm planning problem ..... 93
9.4 Visualization of learned basis for deformable robot planning problem ..... 95
9.5 Qualitative results of deformable robot planning with SLASHDP ..... 96
9.6 Quantitative results of deformable robot planning with SLASHDP ..... 97
10.1 Geometric interpretation of LDD ..... 101
10.2 LDD shape planning results ..... 104
10.3 LDD qualitative results ..... 106
10.4 Quantitative evaluation of LDD and various methods ..... 107
10.5 Setup of windows experiment ..... 108
10.6 Video stills showing inefficiencies of sampling-based planning ..... 109
10.7 Video stills comparing LDD and sampling-based planner on arm planning experiment ..... 110
10.8 Quantitative results from arm planning experiment on the PR2 robot ..... 111
11.1 Illustration of LDD success, failure ..... 116

## Chapter 1

## Introduction

This work is concerned primarily with two distinct subjects: learning the nature of a physical system, and controlling a controllable physical system. In order to make the work interesting and applicable to real-world problems, it is assumed that these systems inhabit high-dimensional state spaces.

At the broadest scope, the entirety of this work is devoted to surpassing the obstacles that arise as a result of this assumption of high dimensionality. As such, it will be helpful to begin with a brief introduction to the concept of the curse of dimensionality.

### 1.1 The curse of dimensionality

The phrase curse of dimensionality appears to have been coined by R. E. Bellman, the esteemed inventor of dynamic programming. In regard to the problem of the maximization of a function, he wrote [11]:

In the first place, the effective analytic solution of a large number of even simple equations as, for example, linear equations, is a difficult affair. Lowering our sights, even a computational solution usually has a number of difficulties of both gross and subtle nature. Consequently, the determination of this maximum is quite definitely not routine when the number of variables is large.

All this may be subsumed under the heading "the curse of dimensionality." Since this is a curse which has hung over the head of the physicist and astronomer for many a year, there is no need to feel discouraged about the possibility of obtaining significant results despite it.

Bellman touches upon most of the salient features that we still associate with the curse of dimensionality; most notably, its ubiquity and inevitability. Curiously, however, Bellman remains optimistic in the face of these problems, noting that "significant results" might be
obtained regardless. Over half a century later, we can safely say that Bellman's optimism was warranted. Although the curse of dimensionality visits new fields as inevitably and rapidly as ever, researchers are just as rapidly developing new tools to combat and/or avoid it.

As previously mentioned, the curse of dimensionality provides a common obstacle to the solution of the seemingly disparate problems addressed in this work. The next sections give a brief overview of these problems and how each is fundamentally affected by this curse.

### 1.2 Learning dynamical systems

The first problem motivating this work might be descibed broadly as learning the dynamics of a physical system: given some empirical observations of the state of a physical system at sampled times, we would like to learn a model of the system's behavior so that we might predict its behavior in novel circumstances.

An example of this is shown in Fig. 1.2. We are given training data consisting of snapshots of a high-dimensional physical system evolving in some predictable ways-in this case, the physical system is a human being performing a variety of exercises. Observations consist of the tracked three-dimensional positions of reflective markers placed at various positions on the subject's body, as depicted in Fig. 1.1. The concatenation of the Cartesian coordinates of all these markers comprises the state space of our system (for now, we will ignore the distinction between state and observation spaces).

We would now like to devise a learning algorithm that produces some sort of model of the dynamics exhibited by the observations as they traverse the high-dimensional state space. The model should be such that we can subsequently perform different types of queries about the learned dynamics by evaluating an inference algorithm that answers queries about the learned model. The query depicted in Fig. 1.2 requests the most likely observation sequence interpolating two specific, previously-unobserved observations, corresponding to start and end poses of a "jumping jack" motion. Evaluating the inference algorithm on this query yields a predicted sequence of high-dimensional observations, which should hopefully be similar to a jumping jack motion previously observed in the training set, while also being consistent with the boundary conditions we specified in the query.

As will be discussed in some detail later, this type of problem is most commonly analyzed from either a machine-learning or control-theoretic point of view. From the control-theoretic perspective, this might be treated as a problem in system identification, whereas someone from a machine learning background might refer to this problem as learning a dynamical system. Typical applications of interest in these fields, in addition


Figure 1.1: Subject wearing motion capture suit with attached retroreflective markers. (Image courtesy of the CMU Graphics Lab Motion Capture Database project, http://mocap.cs.cmu.edu)
to the one just described, include speech recognition, stock market prediction, helicopter control, human activity recognition, and natural language processing, just to name a few.

For many non-trivial applications, the basic issue is the same: as we increase the dimensionality of the problem (i.e., the number of markers in the example described), the curse of dimensionality makes it exponentially more difficult to build an accurate model and query it efficiently. From a statistical/learning-theoretic perspective, this is to say that the sample complexity of the problem grows very rapidly with the dimension; i.e., unless we make some assumptions about the model, we would need an infeasible number of examples to learn it well. Moreover, from a computational efficiency perspective, many of the algorithms we would use to perform inference would also become intractable in high dimensions.

As alluded to earlier, addressing the curse of dimensionality is therefore of utmost importance. The obvious solution to this problem is to leverage an assumption of some kind of low-dimensional structure, and indeed, many classical methods are based on this assumption. The key to the present work is also leveraging low-dimensional structure; however, the present work is distinguised from previous work in that it additionally exploits the assumption of physical origin of the analyzed data, but in a way that is sufficiently general that a detailed physical model is not required. As such, a reasonable balance is struck that avoids the extreme problems of a crippling inability to leverage structure (and thus susceptibility to the curse of dimensionality) on the one hand, and too-strong assumptions that hamper general applicability of the method on the other. This is illustrated in Fig. 1.3,


Figure 1.2: Overview of learning problem. Given training data (sequences of human motion poses), we would like to learn a model such that given a test query, we can apply an inference algorithm to the model to make a prediciton.


Figure 1.3: Illustration of how different learning methods trade-off prior assumptions against being able to cope with high-dimensional systems.
where the method of structured Lagrangians proposed here is compared to a variety of other common methods for learning in dynamical systems.

### 1.3 Motion planning in high-dimensional spaces

The second problem motivating this work is that of how to generate motion plans for systems possessing high-dimensional configuration spaces. That is to say, we desire to find a continuous path through the system's configuration space that connects two specified configurations subject to some constraints, such as obstacle avoidance.

As a particular example of this, consider the robot depicted in Fig. 1.4, which is designed primarily for mobile manipulation tasks in a household environment. Many envisioned tasks for this robot require the generation of smooth, efficient paths connecting the robot's current configuration to a desired configuration. For instance, the robot might be grasping a glass plate, and we might want it to place the plate in the dishwasher while preventing collision of the plate with the environment. We would also like it to do so as efficiently as possible.

Doing so requires the generation of an seven-dimensional path in the configuration space of the robot's arm. Although this is not overtly beyond the reach of current motion planning technology, caveats apply. First, the problem is difficult enough, thanks to the curse of dimensionality again, that most methods settle for finding a feasible solution as


Figure 1.4: The PR2 robot
opposed to an optimal (i.e., efficient) one. Second, the same methods, although they work well in relatively open environments, are likely to fail when faced with very cluttered environments.

These specific drawbacks are properties of sampling-based planning algorithms, which are currently quite possibly the most popular methods for generating high-dimensional motion plans. There are many reasons for the success of these methods, but perhaps one of the most significant is simply that very few other methods are competitive when it comes to reliably producing plans in high-dimensional spaces.

This is somewhat distressing when one considers the simplicity of these algorithms, which basically consist of sampling random configurations, and attempting to connect them with simple paths. The random nature of sampling-based methods seems to unabashedly ignore the important underlying structure of the problem; it would therefore be surprising and unfortunate if there were no more clever method.

Obviously, this is a vast simplification that also ignores other relevant methods-proper discussion of this issue is postponed until later. However, it touches on the principal motivating factor for this work, which is the desire to find and exploit as much structure in the problem as possible. In fact, this might be thought of as a different paradigm for developing motion planning algorithms-one that is structure-centric as opposed to sampling-centric.

An advantage of an algorithm that does not leverage problem-specific structure, on the other hand, is that it need not be modified to adapt to different domains. It is in the desire to retain this feature of adaptability in a structure-exploiting algorithm, that
machine learning becomes relevant. Instead of using a-priori knowledge to modify the algorithm for each particular problem, it would be desirable to have the algorithm itself learn the structure of the problem and exploit it automatically. The second part of this work is devoted precisely to developing such methods.

### 1.4 Overview

This work is divided into two parts: the first concerned with the issue of learning from physical sequence data, and the second focusing on the problem of motion planning in highdimensional spaces. The first part begins with a review of the literature on learning in dynamical systems in its many forms, giving a basis in which to interpret the contributions made here. We then move on to a study of how the physical nature of the systems under consideration might lead to interesting structure useful for learning and inference, proposing concrete algorithms along the way. This part concludes with an experimental study and thoughts regarding potential extensions.

After reviewing relevant literature on motion planning in high-dimensional spaces, the second part draws directly on the results generated in the first part to propose a series of novel approaches for motion planning in high-dimensional spaces. First, methods are proposed to exploit structure in such problems, assuming it is known a-priori. A method is then given to automatically learn the structure from problem data. Finally, two closelyrelated algorithms are given to exploit this structure - one motivated from a view of generating a compressed value function for feedback motion planning, and the other based on an iterative optimization perspective. Experimental results validating these methods are given along the way.

### 1.5 Relation to published work

Much of the work presented here is based on work that has either been published previously by the author or is currently under submission. Parts of the first part of this work are based on material published in [100]. The SLASHDP algorithm presented mainly in Chapter 9 is published in [98]. The LDD algorithm that is the subject of Chapter 10 will appear in [99]. A separate submission focusing on the experimental results obtained with the PR2 robot, is under review. Referenced, but not discussed at length, is work on planning for legged locomotion, published in [101]. The current manuscript, however, presents much of this material in novel and concise ways, with additional interpretations, and in a unified setting.

## Part I

## Learning structured Lagrangian models

## Chapter 2

## Learning dynamical systems

Perhaps the best way to understand the present work in terms of current research is in terms of the general theme of learning in dynamical systems. Although there is a great body of literature falling under this general heading, much of it is sadly scattered across miscellaneous sub-domains of control theory, machine learning, and even operations research-very much to the detriment of all. This chapter makes some attempt to bring together at least a few of the major, notable, and/or relevant concepts and approaches from the different domains.

### 2.1 Dynamical systems

An immediate prerequisite for further discussion is a basic understanding of what a dynamical system is and what it is about a dynamical system that need be learned. In abstract terms, one might define a dynamical system as an entity whose state changes in a prescribed way with the evolution of (some abstract notion of) time. An intuitive example of a dynamical system (and the primary concern of this work) is any physical system we encounter in our everyday experience. At some level of abstraction, the state of such a system could be considered equivalent to its geometric configuration and velocities, and its dynamics as those specified by Newton's laws.

Although this physical context is the one in which we will study dynamical systems currently, a slightly broader understanding of dynamical systems will aid in understanding how this work stands in relation to current research. That said, since complete taxonomy of different categories of dynamical systems and corresponding research issues would take us too far afield, just a brief summary of a few relevant categories is now given. An example will first serve to elucidate some of these distinctions.

Physical systems subject to physical laws are ultimately composed of collections of
particles subject to forces. The state of the system can have many representations, but for now, we represent it as the concatenated Cartesian coordinates $x$ of all the particles in the system. If we consider the universe to consist only of these particles, Newton's second law then stipulates that there exists a function $F(x, \dot{x})$ such that $[7]$

$$
\begin{equation*}
\ddot{x}=F(x, \dot{x}), \tag{2.1.1}
\end{equation*}
$$

where the dot notation is used to express time derivatives.

## Continuous vs. discrete

This simple example already serves to illustrate a number of concepts fundamental to dynamical systems. First, the state of this physical system and the time variable upon which it depends are both assumed to be continuous rather than discrete, since positions and time are naturally continuous entities. More abstract physical models, by contrast, might model the state as a discrete entity.

## Deterministic vs. stochastic

Newton's Enlightenment-era model of mechanics is also prominently deterministic, whereas quantum mechanics (for instance) is inherently stochastic. That is, (2.1.1) contains no element of randomness and hence associates to each initial condition of the system, a unique trajectory; whereas a more detailed model would take into account our inherent uncertainty in our ability to predict the future state of the system.

## Autonomous vs. controlled

In a similar philosophical vein, the physical system described is autonomous in that its future state depends only on its current state, and not on any external influence. A system affected by some external influence, considered to be the effect of our own free will and not stochasticity, might be called a control system.

## Linear vs. nonlinear

The last distinction made here is that between linearity and nonlinearity. The forces in Newton's second law may be linear (i.e., springs) or nonlinear (e.g., nuclear forces) functions of the state. It is notable that this distinction only applies to systems whose state space is a vector space.

Obviously, many other distinctions have been omitted from this list.

### 2.2 System identification

Different approaches to learning of dynamical systems may be categorized principally by the types of dynamical systems to which they apply. It should therefore come as no surprise that some of the earliest work of this nature was concerned solely with the simple linear case previously described. Although the linear assumption will prove too restrictive for the needs of the current work, it will nonetheless be useful to give an idea of the basic principles of these classical methods in order to connect the current work to well-studied issues in control theory. Moreover, the linear case given as intuition as to how to proceed in the nonlinear case, as we will see shortly.

The simplest kind of linear control system is linear in the mapping of input signals (denoted $u(t)$ ) to output signals (denoted $x(t)$ ), where we consider these signals to form vector spaces in the obvious way. These systems are therefore fully characterized by their behavior on a basis of input signals. This behavior takes a particularly simple form when the system is time-invariant. In this case, we can take the input basis to be a collection of time-shifted unit delta (or impulse) functions, and assume that the response to these inputs are time-shifted versions of one another. The behavior of the system is thus fully specified by the impulse response (denoted here by $h(t)$ ); i.e., the output signal generated by any one of the basis input signals.

For a discrete-time, scalar, causal system with a finite impulse response of length $N$, the preceding discussion is summarized by noting that the state sequence can be written as the convolution of the input and impulse response:

$$
\begin{equation*}
x(t)=\sum_{k=0}^{N-1} h(k) u(t-k) \tag{2.2.1}
\end{equation*}
$$

The goal of learning or system identification in this case would be to determine the impulse response $h(t)$ that completely characterizes this system, from observations of state sequences $x(t)$ and input sequences $u(t)$.

The previous equation implies that for each $t$, we observe a single linear constraint on the impulse response. If we stack these linear constraints in matrix form, we see that the state sequence is the multiplication of a Hankel matrix of inputs with the impulse response. By observing sufficient outputs, assuming that the inputs are varied enough, we will eventually obtain a Hankel matrix of row rank at least $N$. At this point, we can solve for the impulse response by simple linear regression. Denoting by $U$ the Hankel matrix of inputs, we obtain a solution via the pseudoinverse:

$$
\begin{equation*}
h=\left(U^{T} U\right)^{-1} U^{T} x \tag{2.2.2}
\end{equation*}
$$

At this point we note that we have formulated and solved a very basic machine learning problem as well. We defined output (or dependent) variables $x(t)$, and have modeled these as linear functions of input (or independent, or regressor) variables $u(t)$. Learning consists of solving for the model parameters $h$ using linear regression, and inference would correspond to applying the linear model to predict future state sequences given novel inputs.

The simple model described so far can be extended in various ways. In an entirely symmetric argument, we can replace "control input" with "previous states" to obtain an autoregressive (AR) model of an autonomous system. We can then add these two models to obtain

$$
\begin{equation*}
x(t)=\sum_{k=0}^{N-1} h_{u}(k) u(t-k)+\sum_{k=0}^{N-1} h_{x}(k) x(t-k) \tag{2.2.3}
\end{equation*}
$$

which gives the familiar input-output representation of a linear system as the sum of a zero-state response and a zero-input response.

We can once again view the problem of identifying the impulse responses as a linear regression problem and proceed as before. From a machine learning perspective, we have now doubled our parameter set and included a new set of regressors to model the richer behavior of this more complex system.

This perspective motivates a natural extension to nonlinear systems [85]: simply replace the regressors with nonlinear functions (or features) of the original regressors. Similarly, we need not be restricted to linear regression to estimate the output from the regressors-any conceivable regression technique could be used.

Going any further down this path brings us solidly into the realm of machine learning. That said, system identification from a control theoretic perspective is a deep subject to which this very brief treatment has not done any justice. The reader is referred to texts such as [60] for a more detailed review.

### 2.3 Learning latent state

We have thus far neglected the effect of the dimensionality of the state space on our ability to learn a dynamical system, since the examples of the previous section assumed a scalar state space. To gain some intuition in this area, we consider the problem of learning a linear, autonomous system in state space representation:

$$
\begin{equation*}
x_{t+1}=A x_{t} \tag{2.3.1}
\end{equation*}
$$

Here $x$ is now assumed to reside in a vector space of dimension $N$, and $A$ is an $N \times N$ matrix to be learned. Recalling a problem mentioned in the introduction, this is generally a difficult problem for large $N$, due to a number of factors. Assuming the optimal case, where we were to attempt to use this model to describe a system with truly linear dynamics, our ability to determine $A$ robustly would still depend heavily on whether we were able to observe trajectories of a sufficiently varied nature.

In the more likely case that the modeled system were not truly linear, bias would limit the accuracy of such a model. We also observe the number of parameters in $A$ grows as the square of the dimension. Overfitting due to the large number of parameters would therefore be a concern if we were to attempt to fit such a model in practice.

### 2.3.1 Hidden Markov Models

Out of these concerns and others grows the desire to leverage low-dimensional structure in dynamical systems. This basic desire forms the impetus behind what is probably the most celebrated and widespread model in machine learning for the analysis of dynamical systems: the Hidden Markov Model (HMM).

A representation of an HMM as a Bayesian network is given in Fig. 2.1. The HMM differs from dynamical systems seen so far in a number of ways-the most salient in the current context being that the HMM models the state (or observation) $x$ as being a function of a (typically) lower-dimensional latent state. It is this latent state alone that possesses dynamics in the HMM model. The observed state is, in this view, a marionette of sorts controlled by the latent dynamics.

In contrast to the control theoretic perspective, the HMM is usually formulated in terms of discrete latent and observation spaces, though this is not strictly necessary - a standard Kalman filter model can also be viewed as a latent space model, though in continuous space.

The HMM is also most often formulated as a stochastic model defining a joint (discrete) probability distribution over all latent states and observations. Crucially, it does so in such a way as to prevent exponential growth of the number of parameters necessary to determine the joint distribution. With no imposed structure, an arbitrary joint distribution for a length $T$ sequence with $N$ latent states and $K$ observation states, would entail a number of parameters on the order of $(N K)^{T}$. The HMM avoids this exponential growth rate by making two conditional independence assumptions:

- All future states are independent of all past states, conditioned on the present state (Markov assumption)
- An observation $z_{t}$ is conditionally independent of all other variables, given the latent


Figure 2.1: A graphical model representation of a hidden Markov model. Shaded areas represent unobserved states.
state $x_{t}$
A (typically made) additional assumption of time-invariance of the dynamics avoids even linear growth of the parameter set in the sequence length. In this case, the HMM is fully determined by the $N^{2}+N K$ parameters of the conditional probability tables $P\left(z_{t} \mid z_{t-1}\right)$ and $P\left(x_{t} \mid z_{t}\right)$.

## HMM inference

As previously noted, in the continuous case with linear dynamics and observation models, the HMM reduces to a linear dynamical system. Inference algorithms include the Kalman filter for forming the online state estimate $P\left(x_{t} \mid z_{1}, \ldots, z_{t}\right)$, and the Rauch-TungStreibel [76] smoother for computing the offline estimate $P\left(x_{t} \mid z_{1}, \ldots, z_{T}\right)$.

In the discrete-state setting more often encountered in machine learning, the online estimate is obtained via the forward algorithm, and the smoothed estimate by the aptlynamed forward-backward algorithm. The latter applies dynamic programming to efficiently compute probabilities in two passes of the data [72].

## HMM learning

The problem of learning an HMM consists of determining the conditional probabilities $P\left(x_{t} \mid x_{t-1}\right)$ and $P\left(z_{t} \mid x_{t}\right)$ given observation sequences $x_{1}, \ldots, x_{T}$. As is so often the case, the most popular algorithm for doing so exploits the efficient HMM inference algorithm to solve the learning problem iteratively. This algorithm, known as the Baum-Welch algorithm [10], can be considered an application of the popular Expectation Maximization (EM) algorithm. As such, it alternates between estimating a distribution over the latent states given parameter estimates and observations; and maximizing the expectation of the log-likelihood of the latent state and observations under this distribution, viewed as a function of the parameters. The EM algorithm guarantees convergence to a local maximum of the likelihood function, but no assurances are given as to global optimality.

Fascinating recent developments $[23,62,44]$ have shown that it is possible to efficiently learn HMMs in Probably Approximately Correct (PAC) settings, given reasonable restrictions; that is, with high probability, an arbitrarily-close approximation to an unknown HMM can be learned in polynomial time. In particular, [44] gives an efficient spectral algorithm for deducing the HMM from observation statistics. The key to this result is finding linear structure in this discrete problem and subsequently applying ideas from the subspace system identification literature $[61,68,96]$ to obtain an analogous algorithm. This provides validation of the principle that discovering low-dimensional structure is often the key to solving high-dimensional learning problems.

## Extensions

Although HMMs in the machine learning literature are typically of the discrete-state variety, some work has been done to extend the HMM learning framework to continuous state spaces. As previously mentioned, the linear case is well-understood in the control systems community; hence, novel extensions typically consider the case of nonlinear dynamics and nonlinear observation functions.

A notable example [37] proceeds in a straightforward way by replacing the Kalman smoother used in a linear dynamical system with an extended Kalman smoother (EKS), a commonly-used nonlinear variant of the Kalman smoother that simply linearizes the dynamics to obtain approximate Gaussian state distributions using the otherwise unmodified Kalman smoother. EM is then used, as in the traditional HMM, to learn the model parameters. Though the inference step is not particularly problematic due to the use of the EKS, the maximization step requires the solution of a nonlinear maximization that is generally difficult. Cleverly, however, the nonlinear dynamics and observation functions are modeled by Gaussian radial basis functions. In this case, since the EKS provides Gaussian state distributions, it happens that the maximization step has an analytic solution obtained by solving a set of linear equations.

### 2.3.2 Gaussian Process Dynamical Models

Another way of extending ideas from the HMM to continuous settings comes by way of Gaussian Process Dynamical Models (GPDMs [102, 103]), which have recently been applied to the analysis of human motion capture data [103], people tracking [94], probabilistic estimation [50], and doubtless other problems. The GPDM can be derived from a perspective similar to the one adopted for nonlinear system identification described in Section 2.2.

Consider the following discrete-time, state-space nonlinear dynamical system model, where $z_{t}$ is the latent state at time $t, x_{t}$ is the corresponding observation, and both $\eta_{t}$ and
$\nu_{t}$ are draws of Gaussian noise:

$$
\begin{align*}
& z_{t}=f\left(z_{t-1}\right)+\eta_{t}  \tag{2.3.2}\\
& x_{t}=g\left(z_{t}\right)+\nu_{t} .
\end{align*}
$$

A straightforward approach to system identification for this model would be to model $f$ and $g$ as weighted combinations of nonlinear basis functions $\phi$ and $\psi$ :

$$
\begin{align*}
f(z) & =\sum_{i} a_{i} \phi_{i}(z)  \tag{2.3.3}\\
g(z) & =\sum_{j} b_{j} \psi_{j}(z) . \tag{2.3.4}
\end{align*}
$$

$\phi$ and $\psi$ could be chosen to represent appropriate features or regressors from which to regress the functions $f$ and $g$ by linear regression; i.e., finding weight vectors $a$ and $b$ such that the resulting model minimizes a squared error loss on some training data. We could increase the complexity of the model simply by adding more basis functions; however, adding too many basis functions would lead to overfitting.

## Gaussian processes

Alternatively, we can take a statistical approach: instead of fitting a deterministic set of parameters, a prior distribution over parameters can be defined, implying a corresponding prior distribution over functions $f(z)$ and $g(z)$. We would then like to use this prior in Bayes' theorem to produce posterior distributions over these functions given observed training data. For example, given some training observations, $z_{1}, \ldots, z_{T}$, we would want to compute the posterior density

$$
\begin{equation*}
p\left(g(z) \mid g\left(z_{1}\right), \ldots, g\left(z_{T}\right)\right) \tag{2.3.5}
\end{equation*}
$$

For such an expression to make sense, we must assume the existence of an underlying stochastic process; that is, for each $z$, there should exist a well-defined random variable $g(z)$ so that we can sensibly make such inferences. Fortunately, the construction of such a process constitutes a classical result in machine learning attributed to Neal [64]. By making appropriate assumptions on the prior parameter distribution, then a central limit argument easily shows that $g(z)$ converges in distribution to a Gaussian in the limit as we take the number of basis functions to infinity. The underlying stochastic process is therefore a Gaussian process (GP).

In this limit, the basis responses form an infinite-dimensional vector $\psi$ (i.e., a function)
from which we can derive all of the second-order moments of the GP. Given an isotropic, zero-mean, unit-variance prior on the parameters, a straightforward computation yields, for arbitrary $z$ and $z^{\prime}$,

$$
\begin{equation*}
\mathbb{E} g(z) g\left(z^{\prime}\right)=\left\langle\psi(z), \psi\left(z^{\prime}\right)\right\rangle . \tag{2.3.6}
\end{equation*}
$$

The zero-mean assumption likewise implies zero mean of the GP. Since a GP is fully determined by its first- and second-order moments, (2.3.6) gives us all the information we need to compute the inference (2.3.5). In a final twist, the basis responses are typically not even specified directly; instead, the inner product in (2.3.6) is instead defined in terms of a positive-definite kernel function

$$
\begin{equation*}
\left\langle\psi(z), \psi\left(z^{\prime}\right)\right\rangle:=K\left(z, z^{\prime}\right), \tag{2.3.7}
\end{equation*}
$$

enlisting a classic machine learning trick known as the kernel trick [5].

## GPDM learning and inference

With a basic understanding of Gaussian process regression, as described in the previous section, the GPDM is fairly simple to explain. The basic concept of the method is to use GPs to model the functions $f(z)$ and $g(z)$ in (2.3.3) and (2.3.4). These GPs fully determine the joint distribution of $x$ and $z$, from which any inference regarding latent states and observations can be performed. Unfortunately, the joint distribution does not constitute a GP; therefore, Markov Chain Monte Carlo (MCMC) methods are used to perform some of these inferences.

The fact that the joint distribution is not a GP also implies that the standard GP learning technique - adapting the kernel parameters to maximize the likelihood of the data - is not strictly possible. However, this deficiency is addressed in a way borrowed from Gaussian Process Latent Variable Models [55]; i.e., the latent states are treated as deterministic parameters that are optimized to maximize the joint likelihood along with the kernel hyperparameters.

In practice, this optimization constitutes a nonlinear optimization problem with a very large number of parameters, since each dimension of each latent state constitutes a distinct parameter. Initialization of the latent states using PCA of the observations [103] helps ease this problem somewhat. Alternatively, N4SID [96], a technique borrowed from the subspace system identification literature, has also been used successfully in this capacity [50].

### 2.4 Learning in optimal control systems

The latent state models most common in the machine learning community, such as those discussed in Section 2.3, are typically concerned with autonomous systems with no notion of control. Adding a notion of control raises a plethora of new issues that have been addressed from both machine learning and control theoretic perspectives.

Perhaps the most fundamental issue arises as a result of considering some control schemes to be superior to others. A natural question to ask in this case is what constitutes the optimal control scheme. The answer to this question is the core of what is known optimal control theory in the control theory community. In the domain of machine learning, such questions are typically treated in the context of reinforcement learning. A brief summary of the latter follows first, for the sake of comparison with the learning methods just described.

### 2.4.1 Reinforcement learning

Reinforcement learning provides a basis in which to study problems concerning the optimal behavior of agents acting in uncertain environments. In the classical (and probably still most common) setting, the environment consists of a discrete state space, and time evolves in discrete steps.

We therefore usually envision the problem as taking place on a finite graph, such as is illustrated in Fig. 2.2. This figure illustrates a very simple Markov Decision Process (MDP). The dynamics of the problem are implied by the arrows, which push the agent towards a new state at each time step. Where two or more arrows emerge from a state, the agent-whose current state is illustrated by the stick-figure - is able to influence, by exerting some control action, the probability that he will transition to some particular next state over another. Upon entering each state, the agent receives a reward, illustrated as a number of coins for each state. The agent's goal is to find an optimal policy-i.e., a mapping of states to control actions - to maximize his cumulative reward over time.

The state is assumed to be known in an MDP. Relaxing this assumption by adding observations and latent states yields a construct known as a Partially Observable Markov Decision Process (POMDP). The HMM might be considered a special case of a POMDP where controls are absent and the state transition graph has a certain linear structure.

Reinforcement learning typically concerns itself with two major problems: on the one hand, the planning problem of finding an optimal policy; and on the other, the learning problem of learning the dynamics of the environment. As one might expect, these problems are closely intertwined.


Figure 2.2: A simple MDP

## Planning

Planning for MDPs may be approached in many ways, but the most common relies on dynamic programming (DP). Central to the concept of dynamic programming is the notion of a value function $V(\cdot)$ that gives for each state, the maximum cumulative reward attainable starting in that state and following an optimal policy. The value function has a simple recursive definition known as Bellman's equation [12]. Denoting by $P\left(x^{\prime} \mid x^{\prime}, a\right)$ the probability of transitioning from state $x$ to state $x^{\prime}$ after taking action $a$, and denoting by $R(x)$ the per-state reward, Bellman's equation is given by

$$
\begin{equation*}
V(x):=\max _{a}\left\{\sum_{x^{\prime}} P\left(x^{\prime} \mid x, a\right)\left(R\left(x^{\prime}\right)+\gamma V\left(x^{\prime}\right)\right)\right\}, \tag{2.4.1}
\end{equation*}
$$

where $\gamma \in[0,1)$ is a given scalar discount factor.
The optimal policy $\pi(x)$ evaluated for a given state $x$ is simply that action which minimizes the right-hand-side of (2.4.1). Knowing the value function is therefore equivalent to knowing the optimal policy.

A classic algorithm for finding the value function consists of turning (2.4.1) into a fixed-point iteration that is guaranteed to converge to the true value function [12] and is appropriately referred to as value iteration.

Bellman himself was one of the first to recognize a critical deficiency of this approach: in large and/or high-dimensional state spaces, the curse of dimensionality makes it impossible to even store the value function as a simple table of values, much less perform the required iteration repeatedly over all states. He proposed as a potential solution to instead represent the value function approximately by a finite, weighted sum of smooth basis
functions, attempting, in his words, "to trade additional computing time, which is expensive, for additional memory capacity, which does not exist." [13]. This approach is usually known today as approximate dynamic programming or value function approximation, and it remains a very active area of research today [30, 31, 32].

Planning in MDPs is an extremely rich and active field that, regrettably, would take us too far afield if we were to discuss it thoroughly at this point. We therefore move on to a brief summary of the learning problem.

## Learning

When we speak of learning in an MDP, it usually refers to the task of planning in an MDP with unknown dynamics. Conceptually, this could be performed by first learning the dynamics (i.e., the state-transition probabilities) and subsequently using one of the planning methods described in the previous section to solve the planning problem; this is the so-called model-based approach. Many methods, however, are based on the observation that the actual state-transition probabilities need not be computed explicitly if all we are concerned with is that the agent act optimally in the world; these methods are referred to as model-free methods.

One well-known example of such a method is Q-learning [104], which employs a value-iteration-like fixed point algorithm to estimate a function $Q(x, a)$ defined as the optimal value conditioned on first taking a step with action $a$. Given a (potentially variable) learning rate $\alpha_{t}$, this results in an iteration without the expectation over actions that would require knowledge of the state-transition distribution:

$$
\begin{equation*}
Q(x, a) \leftarrow Q(x, a)+\alpha_{t}\left[R(x)+\gamma \max _{a^{\prime}} Q\left(x_{t+1}, a^{\prime}\right)-Q(x, a)\right] . \tag{2.4.2}
\end{equation*}
$$

$Q$-learning can be considered a special case of the more general class of temporal difference learning methods, which perform incremental updates based on some sort of error signal such as that found in the right-hand-side of (2.4.2) [97]. As with approximate dynamic programming in the known-model case, the standard approach to making such methods work in high-dimensional spaces is to use function approximation to represent value or $Q$ functions [38, 91]. Perhaps the most well-known success story achieved by such methods is the case of TD-Gammon, a backgammon-playing program based on temporal difference with a neural network function approximator that eventually learned to play at a world-class level by competing in countless trials against itself [92], demonstrating the power of such methods.

### 2.4.2 Inverse optimal control

Many of the ideas of reinforcement learning have equivalents in control theory under the general heading of optimal control. Whereas reinforcement learning is again usually formulated in terms of a discrete setting, optimal control is most often described in terms of continuous space and time. However shallow this difference may appear, it leads to fairly different solution techniques that are worth discussing briefly.

## Optimal control

If there is a classical optimal control problem-just as the small-state-space MDP is the classical reinforcement learning problem-it is most probably the linear quadratic regulator (LQR). The LQR is the optimal solution to the problem of controlling a linear system with a cost functional that is quadratic in the state and control signals. The LQR solution is an analytic function of the parameter matrices.

In general, the solutions to problems in continuous time and space are given by value functions that satisfy the Hamilton-Jacobi-Bellman (HJB) equation, which constitutes the continuous equivalent of the Bellman equation described previously. Unfortunately, the HJB equation is a partial differential equation that does not admit analytic solution in most cases. Discretization is therefore usually necessary to solve these problems.

## Inverse optimal control

The principal philosophical difference between optimal control and reinforcement learning is that optimal control is usually concerned only with the issue of finding optimal control laws and not with any notion of learning. However, there does arise from optimal control a learning problem that is completely distinct from that of classical reinforcement learning; namely, inverse optimal control (IOC). While reinforcement learning assumes that the cost (or reward) functional is known but the dynamics are not, inverse optimal control makes the opposite assumption; i.e., the dynamics are known, but the cost functional is unknown.
R. E. Kalman appears to have been the first to postulate and solve such a question, for the single-input LQR problem [46]. The general LQR inverse problem was ultimately solved by Boyd et. al. [15] three decades later as a simple application of linear matrix inequalities. Aside from that, it does not appear that IOC was historically a pressing issue for the control community.

In the field of operations research, aspects of what is known as inverse optimization may readily be recognized as essentially works in inverse optimal control, though without the trappings of control theory or machine learning [3]. In particular, the inverse shortest path problem [21], which seeks a cost function that renders observed paths, shortest paths;
can be considered the solution to IOC in a deterministic MDP setting. The inverse shortest path problem is also a special case of the inverse network flow problem. Work by Ahuja and Orlin establishes dualities between different types of network flow problems and their inverse problems [4]. It should also be noted that all such problems are ill-posed as stated; the solution to this issue is usually to optimize a criterion such as minimum distance from an initial solution.

The last 10 years have marked the beginning of earnest interest in the IOC problem by the machine learning community, largely sparked by the work by Ng et. al., who referred to the problem as inverse reinforcement learning [66, 2], and notably applied their methods to autonomous, aerobatic helicopter flight, among other problems [65].

Closely related to IOC are methods based on a structured learning concept. Exemplary among these is Max Margin Planning (MMP [75]), which models observed trajectories as optimal trajectories with respect to an MDP planning problem with unknown cost function. MMP learning consists of solving a sequence of planning problems, updating the current cost estimate iteratively in perceptron-like fashion. Of course, a deficiency of this approach is that for high-dimensional problems, it may prove intractable to solve the planning aspect. The current work might be viewed as one way to address this intractability.

## Chapter 3

## Learning structured Lagrangians

The main objective of this work is to introduce a new category of dynamical-systemlearning problems along with learning and inference algorithms to solve (special cases of) it efficiently. As mentioned in the introduction, this first part of the current work describes these algorithms from a machine learning perspective. The applications of these methods to contexts of pure planning and control will be described later in the second part of this work.

What differentiates the current work from the types of approaches studied in Chapter 2 is the desire not only to learn systems of a physical nature, but also to exploit whatever structure we may gain from the assumption of physicality. This philosophy stems partially from a tacit admission that the possibly once hoped-for goal of a universal reasoning and learning machine, is at the very least, not within the grasp of any technology now available or even on the horizon. In artificial intelligence, the term Good Old Fashioned AI (GOFAI) is sometimes used to refer to what is now regarded as this sort of outmoded mentality [41].

The success of HMMs and other latent-state representations, in addition to the recent emergence of structured learning techniques, plainly demonstrate the advantages of exploiting structure where ever possible. On the other hand, there is certainly a trade-off to be made, since the very raison d'être for machine learning is arguably to avoid having to manually build every slight detail about the world into our machines. Managing this trade-off is therefore more of an art than a science - and, though one can always hope to the contrary, it is seems unlikely that this will change in the foreseeable future.

It is with this philosophy in mind that the current proposal of leveraging the structure of physical systems is submitted. As referred to previously in Fig. 1.3, certain methods might benefit from leveraging more detailed physical models ( $[18,59,82]$ ) at the expense of generality, while others (such as those described in Chapter 2) might make the opposite trade-off. The current work can be seen as filling a gap between these extremes.

Moving on from these generalities, the immediate concern will be to elucidate the $L a$ grangian structure whose exploitation is hereby proposed. Before this can be accomplished, an extremely brief introduction to Lagrangian dynamics is necessary for the sake of being reasonably self-contained. The ultimate goal of this chapter will be to give a concrete, efficient algorithm for discovering said structure and showing how it can be exploited to learn Lagrangian systems. The next chapter will then focus on exploiting Lagrangian structure in the inference problem.

### 3.1 Lagrangian mechanics

We examine in this section the behavior of a physical system consisting of a collection of particles obeying Newton's laws. We eschew this view, however, in favor of the equivalent formulation of Lagrangian mechanics. The Lagrangian view may be derived by expressing Newton's second law $(F=m a)$ in a peculiar way. Specifically, we posit the existence of a function $L(x, \dot{x}, t)\left(\right.$ where $\left.x \in \mathbb{R}^{N}\right)$ such that

$$
\begin{equation*}
F=\frac{\partial L}{\partial x} \tag{3.1.1}
\end{equation*}
$$

and

$$
\begin{equation*}
m a=\frac{d}{d t} \frac{\partial L}{\partial \dot{x}} \tag{3.1.2}
\end{equation*}
$$

Given such a function, Newton's second law can obviously be written as

$$
\begin{equation*}
\frac{d}{d t} \frac{\partial L}{\partial \dot{x}}-\frac{\partial L}{\partial x}=0 \tag{3.1.3}
\end{equation*}
$$

Curiously, it can be shown that a path $x(t)$ satisfying (3.1.3) (i.e., Newton's second law) is also a locally extremal path of the cost functional (or action)

$$
\begin{equation*}
J\{x\}=\int_{t_{0}}^{t_{1}} L(x, \dot{x}, t) d t \tag{3.1.4}
\end{equation*}
$$

subject to $x\left(t_{0}\right)=x_{0}$ and $x\left(t_{1}\right)=x_{1}$, for constants $t_{0}, t_{1}, x_{0}, x_{1}$. This is to say that local perturbations (or variations) of $x(t)$ yield no change in the cost functional- $x(t)$ is in this sense a local minimum or maximum of the cost functional. This is the key trick of the Lagrangian formalism: to view physical paths as those that locally optimize the action.

The importance of this view lies in the subtle but crucial observation that, as with any optimization problem, we can choose any coordinates we like in order to find a path that optimizes the action. From this we deduce that any path satisfying Newton's laws in Cartesian coordinates, must satisfy - when written in some arbitrary coordinates $q$-the

Euler-Lagrange equations:

$$
\begin{equation*}
\frac{\partial L}{\partial q}-\frac{d}{d t} \frac{\partial L}{\partial \dot{q}}=0 \tag{3.1.5}
\end{equation*}
$$

Therefore, in order to express the dynamics in terms of $q$ coordinates, we need only to find an expression for $L$ in $q$ coordinates, and apply (3.1.5). This can be achieved by identifying $L$ as the coordinate-invariant quantity

$$
\begin{equation*}
L=T-V \tag{3.1.6}
\end{equation*}
$$

where $T$ is the potential energy of the system and $T$ is its kinetic energy.
To summarize some key points, the Lagrangian perspective gives us an optimizationbased view of physical trajectories. This view allows us easily to determine the ODEs governing these trajectories in arbitrary coordinates simply by writing down the Lagrangian and solving the associated Euler-Lagrange equations.

### 3.1.1 Conservation laws and Noether's theorem

A further benefit of the Lagrangian perspective is that it leads to a deeper understanding of how conservation laws arise in nature. A conservation law is simply a property of a physical system that is invariant with respect to time. For example, if the total energy of a system does not change in time, then that system satisfies conservation of energy. Similarly, a physical body whose momentum remains constant in time is said to obey the law of conservation of momentum.

A classic and celebrated result due to Emmy Noether identifies precisely how such conservation laws arise via the Lagrangian formalism. Informally, Noether's theorem states that every differentiable symmetry of the Lagrangian leads to a corresponding conservation law [7]. In fact, Noether's theorem goes as far as to give a precise expression for the conservation law in terms of the symmetry. However, as the symmetries considered in this work are simple enough that a general formula is not necessary, we will not discuss this formula in further detail.

To give a concrete example of Noether's theorem, consider a Lagrangian $L(x, \dot{x})$ for $x \in \mathbb{R}^{N}$ such that $\partial L / \partial x_{i}=0$, for some $i$. Writing the Euler-Lagrange equations for this coordinate yields

$$
\frac{\partial L}{\partial x_{i}}-\frac{d}{d t} \frac{\partial L}{\partial \dot{x}_{i}}=-\frac{d}{d t} \frac{\partial L}{\partial \dot{x}_{i}}=0 .
$$

Since the time derivative of $\partial L / \partial \dot{x}_{i}=0$, we therefore have that

$$
\frac{\partial L}{\partial \dot{x}_{i}}=k
$$

for some trajectory-dependent constant $k$, for all time.
Making this problem even more concrete, suppose that $x$ is the position of a particle of mass $m$ near the Earth's surface, and $x_{3}$ is the height of the particle above an arbitrary reference point. Further suppose that the particle is moving freely through space, affected by no other force than gravity. The Lagrangian for this particle is given by

$$
\begin{equation*}
L(x, \dot{x})=\frac{m}{2}\|\dot{x}\|^{2}-g x_{3} . \tag{3.1.7}
\end{equation*}
$$

Note that $\partial L / \partial x_{1}=\partial L / \partial x_{2}=0$. Therefore, by the above argument, we can deduce that conservation laws hold for the $x_{1}$ and $x_{2}$ coordinates:

$$
\begin{align*}
& \frac{\partial L}{\partial \dot{x}_{1}}=m \dot{x}_{1}=k_{1}  \tag{3.1.8}\\
& \frac{\partial L}{\partial \dot{x}_{2}}=m \dot{x}_{2}=k_{2} \tag{3.1.9}
\end{align*}
$$

These conservation laws constitute the familiar conservation of momentum laws in the directions $x_{1}$ and $x_{2}$ parallel to the Earth's surface.

As any student of high-school physics is aware, these conservation laws greatly simplify the analysis of simple ballistic trajectories - solving any such problem reduces to the solving the one-dimensional ODE $\ddot{x}_{3}=-g$, since the other velocities remain constant. Fortunately, this phenomenon is general: the presence of conservation laws always simplifies the analysis of physical systems. Since Noether's theorem essentially tells us that we can obtain a conservation law for free with every symmetry that we can find in our system, finding these symmetries is of utmost importance when we are faced with the analysis of any physical system. The importance of this statement cannot be stressed enough, as it may very well be considered the foundation of this entire work.

### 3.2 Structured Lagrangians

The method proposed here, at a high level, is simply to propose that observed trajectory data obeys a Lagrangian model and to subsequently fit the Lagrangian model from the observations. The Lagrangian model will then give a complete description of the physical dynamics underlying the trajectories, which we can consult, in principle, to answer any query regarding the behavior of the system in novel situations.

Unfortunately, we cannot hope to accomplish this in the general case, due to problems that arise in both learning and inference. For a system of $N$ particles in a three-dimensional space, the Lagrangian (written in Cartesian coordinates) is a map $L(x, \dot{x}, t): \mathbb{R}^{3 N} \times \mathbb{R}^{3 N} \times$ $\mathbb{R} \rightarrow \mathbb{R}$ that takes as input the state of the system, its velocities, and time; and produces
the difference of kinetic and potential energy for the system in that state. Learning such a map in a supervised sense would therefore require training examples pairing states and differences of potential and kinetic energies - data that would most likely not be readily available in most cases.

Many physical systems of interest are also high-dimensional. Suppose we would like again to study a system of $N$ particles, where $N$ is large. Assuming we can obtain training data to fit the Lagrangian, we are still affected by the curse of dimensionality: a look-uptable representation of $L$ would contain a number of elements scaling exponentially in $N$, thus necessitating strong prior assumptions on $L$ in order to be able to fit it. Using the wrong prior assumptions would lead to a biased model of $L$, thus limiting the ability of such a model to fit the data well.

The problem of inference for a Lagrangian system is at least well-defined: the physical trajectories associated with a Lagrangian system are exactly those that locally optimize the action. In principle, given some representation of a Lagrangian, we could therefore find paths by solving optimization problems. However, in a high-dimensional space, finding such local optima may prove very difficult. Each dimension of the sought-after path, $x_{i}(t)$ is a function, or alternatively, a vector in an infinite-dimensional vector space. Thus, we are faced with the problem of optimization in a "high-dimensional, infinite-dimensional" space. Such a problem might be approached using a functional gradient descent technique, but this entails discretization with concomitant numerical issues. Even assuming that we are able to overcome this problem, it is unclear as to whether the found optimum, of which there may be many, would be representative in any way-possibly necessitating the use of sampling to find other optima, which may prove intractable.

All of these obstacles point decidedly towards the conclusion that fitting a general Lagrangian model is neither a tractable or well-defined problem. Our aptly-named solution to this issue is the introduction of the notion of a structured Lagrangian - a name intended perhaps to evoke connections to structured learning. This section gradually introduces a particular kind of structured Lagrangian, though it is not the only conceivable kind. Still, arguments will be made as to why this is a sensible model for a variety of cases.

### 3.2.1 Conservation of energy

Although this section has so far painted a grim picture of the feasibility of the Lagrangian learning model, Noether's theorem provides an optimistic counterbalance by guaranteeing us that every symmetry we can find in the Lagrangian "decreases by one" the complexity of the problem. Our search for structure in Lagrangians will therefore be guided by the desire to find and exploit these symmetries.

A simple first simplification is therefore to remove the explicit dependence of the Lagrangian on time. It can be shown by Noether's theorem that this assumption, remarkably, implies conservation of energy. On the one hand, energy conservation may seem to constitute a rather onerous assumption, since it would preclude modeling desirable effects such as friction and inelastic collisions. On the other hand, allowing arbitrary dissipative and external forces into our model would be unwise - a system obeying Newton's laws subject to unrestricted external forces can simulate the behavior of any kind of second-order dynamical system, which is exactly contrary to our objective of exploiting some kind of additional structure present in physical systems. In this light, ignoring dissipative effects would seem advantageous.

### 3.2.2 Kinetic Lagrangians

The other major assumption to be made will address most of the problems previously mentioned with the Lagrangian learning concept in one fell swoop. Namely, we will assume the following concrete form of the Lagrangian in terms of some coordinates $x$ :

$$
\begin{equation*}
L(x, \dot{x})=\frac{1}{2} m(x)\|\dot{x}\|^{2} \tag{3.2.1}
\end{equation*}
$$

Such Lagrangians will be referred to as kinetic Lagrangians in this work, as they represent Lagrangians that possess only a kinetic energy term, and no potential energy. This kinetic energy is exactly the kinetic energy of a system of particles in Cartesian space with a common mass dependent on the position of all the particles. Though this may appear to be an odd model at the outset, kinetic Lagrangians possess a number of interesting properties that make them a suitable model for learning applications.

## Supervised training

First, the conservation of energy resulting from the time-independence of this Lagrangian implies that the kinetic energy of the system within trajectories is constant, since the potential energy is zero. Writing this energy as $E$, this implies that

$$
\begin{equation*}
m(x)=\frac{2 E}{\|\dot{x}\|^{2}} \tag{3.2.2}
\end{equation*}
$$

Note that the Lagrangian is completely determined by the function $m(x)$, implying that training examples of pairs $(x, m(x))$ are all that we need in order to fit the Lagrangian. $m(x)$ is in turn determined by the inverse speed of the trajectory and a per-trajectory constant energy. Assuming for now that we can obtain samples of trajectories in these coordinates, obtaining samples of $x$ and $\dot{x}$ is therefore possible. Although the per-trajectory
energy $E$ may not be known, it represents only one unknown parameter per trajectory, which might be estimated by a method such as Expectation Maximization. If we assume we are observing only one continuous trajectory, $E$ is irrelevant in the sense that it only scales our estimate of the Lagrangian by a constant factor. This constant factor becomes irrelevant when we perform inference by optimizing the Lagrangian.

## Conservation laws

The other problems anticipated with the learning Lagrangian approach arose principally in the case of high-dimensional systems, the main challenge being to identify the correct prior assumption to make as to the structure of the Lagrangian in order to avoid complications owing to the curse of dimensionality. Inspired by Noether's theorem, our approach will be to assume the presence of symmetries of the Lagrangian, which will generate conservation laws that should simplify the task of learning the Lagrangian.

Armed with a concrete form of the Lagrangian (i.e., the kinetic Lagrangian), this is straightforward. From the machine-learning perspective, a natural symmetry to propose is a simple translational symmetry equivalent to the presence of linear low-dimensional structure. That is, we can propose the symmetry

$$
\begin{equation*}
L(x, \dot{x})=L(x+s v, \dot{x}), \forall s \tag{3.2.3}
\end{equation*}
$$

for a particular vector $v$. Owing to the form of the kinetic Lagrangian, this is equivalent to stating that

$$
\begin{equation*}
m(x)=m(x+s v), \forall s \tag{3.2.4}
\end{equation*}
$$

This is illustrated in Fig. 3.1. We say informally that $m$ has low-dimensional structure in the sense that $m$ does not vary across the subspace spanned by $v$. Therefore, if we wish to estimate $m$ in the sense of supervised learning, every such symmetry reduces the dimensionality of our estimation problem by one.

That said, it would be undesirable and physically unjustified to declare such symmetries by fiat merely because doing so simplifies the estimation problem. Instead, these presence of these symmetries will be detected by identification of the conservation laws that necessarily follow.

Accordingly, the form of these conservation laws is now derived. Suppose $v$ above is such that

$$
\frac{\partial m}{\partial x_{i}}(x)=0, \forall x
$$

that is, $m$ does not depend on the coordinate $x_{i}$. Such a coordinate will be referred to as a cyclic coordinate, to borrow a term from physics. Then the Euler-Lagrange equations


Figure 3.1: Visualization of a kinetic Lagrangian with low-dimensional structure
imply

$$
\begin{align*}
\frac{\partial L}{\partial x_{i}}-\frac{d}{d t} \frac{\partial L}{\partial \dot{x}_{i}} & =0 \\
\frac{1}{2}\|\dot{x}\|^{2} \frac{\partial m}{\partial x}-\frac{d}{d t}\left[m(x) \dot{x}_{i}\right] & =0 \\
m(x) \dot{x}_{i} & =k^{\prime} \tag{3.2.5}
\end{align*}
$$

where $k^{\prime}$ is a trajectory-dependent constant. We can combine this new conservation law with conservation of energy (3.2.2) to obtain

$$
\begin{equation*}
\frac{\dot{x}_{i}}{\|\dot{x}\|^{2}}=\frac{k^{\prime}}{2 E} \tag{3.2.6}
\end{equation*}
$$

Finally, observing that both $k^{\prime}$ and $E$ are trajectory-dependent constants, we can combine them into a new constant $k$, yielding

$$
\begin{equation*}
\frac{\dot{x}_{i}}{\|\dot{x}\|^{2}}=k \tag{3.2.7}
\end{equation*}
$$

It is the presence of this conservation law that we will use to detect symmetries of the form (3.2.4). A description of the concrete algorithm used to accomplish this is postponed briefly in order to give more insight into the nature of kinetic Lagrangians.

### 3.2.3 Geometric interpretation

Kinetic Lagrangians have an important geometric interpretation that warrants further attention here. This geometric connection may be established using a well-known correspondence that provides useful insight in the current context.

First, we introduce the notion of the length $L$ of a curve $x(t)$ on a manifold with Riemannian metric $g$, which is given by

$$
\begin{equation*}
L(x)=\int_{t_{0}}^{t_{1}}\left(\sum_{i j} g_{i j}(x) \dot{x}_{i} \dot{x}_{j}\right)^{1 / 2} d t \tag{3.2.8}
\end{equation*}
$$

By introducing a time-reparameterized path $z(t)=x(s(t))$, with $s\left(t_{0}\right)=t_{0}, s\left(t_{1}\right)=t_{1}$, and $d s / d t \neq 0$; and performing a change of variables in the expression above to $s$, it is easily shown that $z$ and $x$ have the same length; that is, $L$ is invariant under timereparameterization of curves. This coincides well with our intuition about the length of a curve, which is a geometric property independent of however we happened to parameterize the curve. Taking $g_{i j}(x)=\delta_{i j} m(x)$ yields the familiar-looking expression

$$
\begin{equation*}
\int_{t_{0}}^{t_{1}} \sqrt{m(x)}\|\dot{x}\| d t \tag{3.2.9}
\end{equation*}
$$

This can be related to the kinetic Lagrangian by applying the Cauchy-Schwarz inequality to the inner product of the functions $\sqrt{m(x)}\|\dot{x}\|$ and 1 , yielding

$$
\begin{equation*}
\int_{t_{0}}^{t_{1}} m(x)\|\dot{x}\|^{2} d t \int_{t_{0}}^{t_{1}} d t \geq\left(\int_{t_{0}}^{t_{1}} \sqrt{m(x)}\|\dot{x}\| d t\right)^{2} \tag{3.2.10}
\end{equation*}
$$

with equality iff. $\sqrt{m(x)}\|\dot{x}\| \propto 1$-which incidentally recovers conservation of energy for minimizers of the Lagrangian. From this we can deduce that minimal-length curves are also minimizers of the energy functional (i.e., the kinetic Lagrangian), up to a timereparameterization.

Thus, the trajectories associated with kinetic Lagrangians have a geometric interpretation as geodesics of a Riemannian manifold with an isotropic metric $m(x) \delta_{i j}$.

These geodesics furthermore coincide with the intuitive notion of shortest or minimumcost paths, if we think of $\sqrt{m(x)}$ as being a cost function that assigns penalties to curves passing through regions of space. We can therefore find physical trajectories associated with kinetic Lagrangians by solving minimum-cost path problems. This view is significant algorithmically due to the great number of efficient algorithms available to solve such problems.


Figure 3.2: Illustration of Snell's law

### 3.2.4 Fermat's principle of least time

A further important interpretation of kinetic Lagrangian systems comes from optics. Fermat's principle of least time states that the path taken by a ray of light is that which minimizes the total time needed for the ray to travel between endpoints. Taking the aforementioned view of kinetic Lagrangian trajectories as being minimum-cost paths, we can identify cost with inverse speed or index of refraction to conclude that these trajectories are also minimum-time trajectories. Therefore, we can think of these trajectories as those that rays of light would travel-possibly in a space of dimension greater than three.

Perhaps the most famous formula of optics is Snell's law, illustrated in Fig. 3.2. When a ray of light passes from a material with index of refraction $n_{1}$ to another with index of refraction $n_{2}$, the angle of the ray made with the normal to the interface changes according to $n_{1} \sin \theta_{1}=n_{2} \sin \theta_{2}$. This can be derived from the conservation law (3.2.7) via the identifications

$$
\begin{align*}
n & =\frac{1}{\|\dot{x}\|}  \tag{3.2.11}\\
\dot{x}_{2} & =\frac{\sin \theta}{n} \tag{3.2.12}
\end{align*}
$$

assuming that the interface is orthogonal to the $x_{2}$ coordinate direction. This reveals that $n \sin \theta=k$, for some constant $k$, for all time; which implies that $n_{1} \sin \theta_{1}=n_{2} \sin \theta_{2}$. Eq. (3.2.7) can therefore be considered a generalization of Snell's law that applies to higherdimensional spaces.

### 3.3 Fermat Components Analysis

The algorithmic implementation of the program set out previously-i.e., to deduce symmetries of the form (3.2.2) from the presence of conservation laws of the form (3.2.7) -is now related. Even though it is possible to implement this idea directly, it is usually more efficient to perform the converse operation, learning a basis for the subspace in which the function $m(x)$ does vary. The reason for this is that, in the analysis of a very highdimensional system, we expect to find more dimensions in which conservation laws do hold than not, making it more efficacious to focus on the latter than the former.

To phrase the issue in terms of a latent-state model, we posit that there exists a lowdimensional acyclic subspace in which $m(x)$ does vary, but we do not directly observe trajectories projected onto this subspace; instead, we can only observe high-dimensional trajectories that are generally not constrained to lie within the acyclic subspace. The task is then to identify this subspace.

This latent-state model stands notably in contrast to methods such as HMMs and GPDMs, which model all of the dynamics as occurring in the latent space, the observations being merely high-dimensional images of latent states. By contrast, in the kinetic Lagrangian model, there exists a factorization of the state space into a latent space that exhibits complex dynamics (the acyclic space), and a residual space that exhibits simple dynamics conditioned on the acyclic state. This is illustrated by Bayesian network illustrated in Fig. 3.3. The current state of each variable pertaining to the low-dimensional acyclic state can in general depend on the value of any acyclic variable in the previous state, but the each variable in the cyclic state is conditionally independent of other variables given the acyclic state.

### 3.3.1 Spectral solution

The latent subspace may be identified in the following way. We denote by $z \in \mathbb{R}^{N}$ the coordinates of the observed trajectories, and by $x \in \mathbb{R}^{d}$ the coordinates of the latent trajectories, such that these are related by the expression $x=W^{T} z$, for some to-bedetermined matrix $W \in \mathbb{R}^{N \times d}$. Eq. (3.2.7) therefore implies that

$$
\begin{equation*}
\frac{W^{T} \dot{z}}{\|\dot{z}\|^{2}} \neq k \tag{3.3.1}
\end{equation*}
$$

for any $k \in \mathbb{R}^{d}$. It is therefore proposed to identify $W$ by choosing it to "maximize" the inequality over the best-fit value of $k$, in expectation:

$$
\begin{equation*}
W=\underset{W^{T} W=I}{\arg \max } \min _{k} \mathbb{E}\left(W^{T} \dot{z}-k\|\dot{z}\|^{2}\right)^{T}\left(W^{T} \dot{z}-k\|\dot{z}\|^{2}\right) \tag{3.3.2}
\end{equation*}
$$

Low-dim., complex dynamics

High-dim., factored dynamics


Figure 3.3: Interpretation of factorization provided by FCA and kinetic Lagrangians as a graphical model

Concretely, this can be achieved as follows. Suppose that we obtain sampled trajectory data of the form $z[t]$ and $\dot{z}[t]$, where $t$ indexes discrete times; and that these sampled data points pertain to $M$ different trajectories, with $p_{t} \in\{1, \ldots, M\}$ denoting the trajectory to which the $t$ th sampled datum belongs. Representing by $w_{i}$ the $i$ th column of $W$, the following characterization of $w_{1}$ is proposed:

$$
\begin{equation*}
w_{1}=\underset{\|w\|=1}{\arg \max } \min _{k_{1}, \ldots, k_{M}} \sum_{t}\left(w_{1}^{T} \dot{z}[t]-k_{p_{t}}\|\dot{z}[t]\|^{2}\right)^{2} . \tag{3.3.3}
\end{equation*}
$$

This implements the idea of (3.3.2), albeit with additional bookkeeping to keep track of different $k$ values for different trajectories. The solution is readily obtained using a standard analysis, reminiscent of the way PCA may be derived. First, we note that the inner minimization over $k$ may be solved analytically in terms of $w$ simply by setting the gradient with respect to $k$ equal to zero, from which we obtain

$$
\begin{align*}
& k_{i}=w^{T}\left(\frac{\sum_{\left\{t \mid p_{t}=i\right\}} \dot{z}[t]\|\dot{z}[t]\|^{2}}{\sum_{\left\{t \mid p_{t}=i\right\}}\|\dot{z}[t]\|^{4}}\right)  \tag{3.3.4}\\
& k_{i}:=w^{T} d_{i} \tag{3.3.5}
\end{align*}
$$

defining $d_{i}$ in the last step. We then substitute this solution into the objective:

$$
\begin{equation*}
w_{1}=\underset{\|w\|=1}{\arg \max } \sum_{t}\left(w^{T} \dot{z}[t]-w^{T} d_{p_{t}}\|\dot{z}[t]\|^{2}\right)^{2} \tag{3.3.6}
\end{equation*}
$$

Defining

$$
\begin{equation*}
Q:=\sum_{t}\left(\dot{z}[t]-d_{p_{t}}\|\dot{z}[t]\|^{2}\right)^{T}\left(\dot{z}[t]-d_{p_{t}}\|\dot{z}[t]\|^{2}\right) \tag{3.3.7}
\end{equation*}
$$

this can finally be expressed as

$$
\begin{equation*}
w_{1}=\underset{\|w\|=1}{\arg \max } w^{T} Q w \tag{3.3.8}
\end{equation*}
$$

the well-known solution to which is the eigenvector corresponding to the maximum eigenvalue of $Q$. Subsequent columns of $W$ are therefore obtained as the next $d-1$ eigenvectors of $Q$, sorted in order of descending eigenvalue.

We refer to this method as Fermat Components Analysis, in reference to the aforementioned optical interpretation of kinetic Lagrangian systems.

### 3.4 Estimation of the Lagrangian

Having deduced a basis for the acyclic subspace, estimation of the Lagrangian is mostly a routine task. Recalling Section 3.2.2, making the approximation that all trajectories are equally energetic, we can simply regress the Lagrangian from the acyclic subspace components in a purely supervised way using any standard method. In practice, a NadarayaWatson kernel estimator was found to produce good results in this capacity.

Fig. 3.4 shows the results of FCA applied to what might be called an "inverse optics" problem in two dimensions; i.e., given (simulated) observed trajectories of light rays passing through a medium of varying but unknown refractive index, the task is to deduce the varying refractive index of the medium. Brighter areas in the figure have higher refractive index (equivalently, lower speed). Light rays, illustrated as colorful lines, turn towards high-cost regions upon entering them to minimize travel time, in accordance with Fermat's principle of least time.

The solid arrow shows the first Fermat component of the data, while the dashed arrow shows the first PCA component of the data. Fig. 3.4b shows the underlying onedimensional cost function learned by performing regression of the cost function from the Fermat component, and Fig. 3.4c shows the result of regression from the PCA component.

The Fermat component approximates very well the actual direction in which the refractive index actually varies, while the PCA component is almost orthogonal to it. Consequently, regression of the refractive index from the FCA component recovers the structure of the medium almost perfectly, while regression from the PCA component yields very poor results.


Figure 3.4: Simple demonstration of Fermat components applied to a kinetic Lagrangian system: light rays (dashed lines in 3.4a) traveling through a medium of varying refractive index.

## Chapter 4

## Inference for structured Lagrangians

Having settled the issue of learning the Lagrangian in the previous chapter, we now turn to the issue of inference. Recall that inference in the Lagrangian model consists of solving an optimization problem, which may be generally difficult to solve in high-dimensional spaces. In this chapter, we will see how the low-dimensional structure associated with kinetic Lagrangians leads to surprisingly efficient inference algorithms, paving the way for a complete solution for learning from high-dimensional physical sequence data.

### 4.1 Inference by optimization

The physical paths associated with a given Lagrangian, as reviewed in Chapter 3, are those that locally minimize or maximize the action, which for a kinetic Lagrangian consists of the cost functional

$$
\begin{equation*}
J\{x\}=\int_{t_{0}}^{t_{1}} \frac{1}{2} m(x)\|\dot{x}\|^{2} . \tag{4.1.1}
\end{equation*}
$$

The inference algorithm proposed here consists of finding the global minimizer of this action. This approach, as opposed to finding an arbitrary local minimum or maximum by local optimization techniques, might be justified by making a common approximation in a probabilistic inference framework.

Probabilistic inference for this system entails defining a probability distribution over paths. A natural choice for such a distribution would assign high probabilities to paths of small action and large probabilities to paths of large action. This may be implemented in
practice by defining the following Boltzmann distribution over paths:

$$
\begin{equation*}
P(x)=\frac{e^{-J\{x\} / T}}{Z(T)} \tag{4.1.2}
\end{equation*}
$$

where $T$ is a temperature parameter and the partition function $Z(T)$ is whatever constant is necessary to ensure that the distribution integrates to one over all paths. In quantum mechanics, a similar approach results in what is known as the path integral formulation of quantum mechanics.

Unfortunately, calculating the probability of even a single path in this setting is computationally intractable, as it requires the computation of the partition function, which can only be determined by calculating and summing the actions of all the paths. Approximations are therefore necessary to proceed.

Various approximations and sampling techniques are possible. The one used here is the zero-temperature approximation: in the zero-temperature limit, the Boltzmann distribution (4.1.2) converges to a delta distribution on the mode-i.e., the path of maximum probability, or minimum action. This approximation hence consists of finding just the path that globally minimizes the action. Although this is generally a very difficult problem, the next sections will show that globally optimal solutions can be found efficiently for kinetic Lagrangians with low-dimensional structure.

### 4.2 Dynamic programming

As the optimization method proposed here is ultimately based on dynamic programming (DP), a brief review of some DP concepts is now undertaken. In general terms, DP can refer to a variety of methods used to solve countless types of problems. Underlying all of these is a notion of optimal substructure.

Dijkstra's algorithm is perhaps the most classic of all DP algorithms. Dijkstra solves what is commonly known as the shortest path problem on a graph, defined as follows: given a (possibly directed) graph with a cost assigned to each arc, find a contiguous sequence of arcs (a path) emanating from a given node, and terminating in another given node, such that the summed cost of these arcs is minimized.

As is typically the case in DP, Dijkstra's algorithm employs a value function, defined as the minimum cost of a path starting at a given node and ending at the goal node. The key insight of Dijkstra is that it is possible to compute the values in ascending order, starting at the goal and computing one value per iteration, each of which takes time scaling linearly in the number of nodes (for a naive implementation), yielding a net quadratic complexity in the size of the graph.

Dijkstra's algorithm constitutes just one way of solving Bellman's equation, which is a recurrence relation in the value function encapsulating the optimal substructure of the shortest path problem. As mentioned earlier, the analogous equation for optimal control systems in continuous spaces is the Hamilton-Jacobi-Bellman (HJB) equation. For kinetic Lagrangians, the HJB equation ultimately reduces to the well-known Eikonal equation of optics, which-denoting by $V(x)$ the value function at a state $x$ and defining $m(x)$ as before - is a partial differential equation (PDE) with the simple form

$$
\begin{equation*}
\|\nabla V(x)\|=\sqrt{m(x)} \tag{4.2.1}
\end{equation*}
$$

The corresponding optimal control law is then simply

$$
\begin{equation*}
\dot{x}=-\nabla V(x) \tag{4.2.2}
\end{equation*}
$$

the integration of which yields optimal paths.
The numerical solution of PDEs such as the Eikonal equation usually entails sampling the domain on a regular lattice and computing an approximate solution at the lattice points. The regular lattice structure allows one to replace the partial derivatives with finitedifference approximations, which gives rise to systems of (generally) non-linear equations that may be solved to obtain approximate solutions numerically at the lattice points. Performing this procedure for the Eikonal equation leads to a system of equations quadratic in the values at the lattice points.

Fortunately, the special structure of the Eikonal equation admits a solution more efficient than solving a quadratic equation in a massive number of variables. The solution, commonly known as the Fast Marching Method [83] (FMM) (developed also independently by Tsitsiklis [93]), is exactly analogous to Dijkstra's algorithm, except that each iteration solves a one-dimensional quadratic equation for the next value.

Such algorithms are efficient in the sense that they require only a single pass through the discretized state space to compute all values. However, this proves much too burdensome for high-dimensional state spaces, whose discretized size grows exponentially with dimension.

By making the cost function (or $\sqrt{m(x)}$ ) complex enough, one could certainly devise pathological examples where finding an optimal path would require visiting every lattice point, no matter how efficient the search algorithm; it therefore seems that the algorithms are not so much to blame as is our insistence on representing the cost function in a form that allows it to be of arbitrary complexity. One would hope that there exist algorithms that, given a compressed representation of the cost, would produce a similarly compressed representation of the value function.

Fortunately, this is the case. We will see that compressed cost functions-specifically, those that can be compressed by storage in look-up tables omitting certain dimensions lead to compressed value functions that can be computed, stored, and queried efficiently. Afterwards, we will examine an alternate view that explains the same result in terms of the low-dimensional structure of the optimal paths themselves.

### 4.3 Symmetry of the value function

This section gives a simple proof of the fact that low-dimensional cost functions are associated with symmetric value functions. However simple, it may be considered the key result that enables all of the methods presented in this work.

We consider in this section the problem of finding a minimizer $x^{*}(t)$ of the functional

$$
\begin{align*}
J\{x\}= & \int_{0}^{1}\|\dot{x}(t)\| C(x(t)) d t  \tag{4.3.1}\\
\text { subject to } \quad & x(0)=0 \\
& x(1)=y,
\end{align*}
$$

for a given cost function $C(x)$ and boundary condition $y$, which is equivalent to the kinetic Lagrangian in the sense described in Section 3.2.3. We then define the value function to be the minimum of this functional, viewed as a function of the free endpoint $y$ :

$$
\begin{equation*}
V(y)=\min _{x(t)} J\{x\} \tag{4.3.2}
\end{equation*}
$$

The following theorem states that the value function associated with a cost function that varies only in a low-dimensional subspace, is symmetric about rotations that preserve the subspace.

Theorem 4.3.1. Suppose that $C(x)$ is the cost associated with a variational problem of the form (4.3.1) and $V(y)$ is the associated value function. If $W$ is such that $C(x)=$ $C\left(W W^{T} x\right), \forall x$, and $R$ satisfies $R W=W, R^{T} R=I$, then $V(y)=V(R y), \forall y$.

Proof. Let $z(t)=R x(t)$. The proof follows by algebra:

$$
\begin{align*}
V(y) & =\min _{x(t)} \int_{0}^{1}\|\dot{x}\| C(x) d t, \text { s.t. } x(0)=0, x(1)=y \\
& =\min _{z(t)} \int_{0}^{1}\|\dot{z}\| C\left(R^{T} z\right) d t, \text { s.t. } z(0)=0, z(1)=R y \\
& =\min _{z(t)} \int_{0}^{1}\|\dot{z}\| C\left(W W^{T} R^{T} z\right) d t, \text { s.t. } z(0)=0, z(1)=R y \\
& =\min _{z(t)} \int_{0}^{1}\|\dot{z}\| C\left(W W^{T} z\right) d t \text {, s.t. } z(0)=0, z(1)=R y \\
& =\min _{z(t)} \int_{0}^{1}\|\dot{z}\| C(z) d t, \text { s.t. } z(0)=0, z(1)=R y \\
& =V(R y) . \tag{4.3.3}
\end{align*}
$$

Practically speaking, the significance of this theorem is that it enables us to implement the plan alluded to earlier, to directly compute a value function in a compressed form from which any desired value may be extracted efficiently. This fact is made explicit in the following corollary.

Corollary 4.3.2 (Value function compression). Suppose that $C(x)$ is the cost associated with a variational problem of the form (4.3.1) and $V(y)$ is the associated value function. If $W$ is such that $C(x)=C\left(W W^{T} x\right), \forall x$; and $\nu$ is any vector such that $W^{T} \nu=0$ and $\|\nu\|=1$, then

$$
\begin{equation*}
V(y)=V\left(W W^{T} y+\left\|\left(I-W W^{T}\right) y\right\| \nu\right), \forall y . \tag{4.3.4}
\end{equation*}
$$

Proof. We proceed by constructing a rotation satisfying the conditions of Theorem 4.3.1 that rotates $y$ onto the subspace spanned by $W$ and $\nu$. Defining

$$
\begin{equation*}
y_{\perp}=\left(I-W W^{T}\right) y, \tag{4.3.5}
\end{equation*}
$$

we choose $R$ to be any rotation satisfying

$$
\begin{align*}
R W & =W \\
R \frac{y_{\perp}}{\left\|y_{\perp}\right\|} & =\nu \tag{4.3.6}
\end{align*}
$$

Note that the orthonormality assumptions on $\nu$ render this a valid rotation. Then

$$
\begin{align*}
R y & =R\left(W W^{T} y+\left(I-W W^{T}\right) y\right)  \tag{4.3.7}\\
& =W W^{T} y+\left\|y_{\perp}\right\| \nu \tag{4.3.8}
\end{align*}
$$

Applying Theorem 4.3.1 to this expression yields the desired conclusion.

The compressed representation of the value function therefore consists of the restriction of the value function to a subspace spanned by $W$ and an arbitrary vector $\nu$ orthogonal to it. The value at an arbitrary point $y$ may be extracted by rotating $y$ onto this subspace and evaluating the restricted value function at this new point. Thus, if $W$ is an $N \times d$ matrix, we have reduced the problem of computing a $N$-dimensional value function to that of computing a $d+1$-dimensional value function.

### 4.4 Low-dimensional structure of optimal paths

An alternate version of this result focuses on the low-dimensional nature of the optimal paths themselves. It states that an optimal path lies in the smallest linear subspace containing both the endpoints and the subspace in which the cost function varies. This is illustrated in Fig. 4.4.1. Shown is an optimal path in three dimensions of a cost function that varies only in one dimension. The blue blocks represent high-cost regions that are assumed to extend ad infinitum in directions orthogonal to the cost variation direction; all other points in the space are assigned a uniform cost, which is assumed to be a distinct cost from the blue-shaded regions. The optimal path lies in the smallest two-dimensional subspace containing both the direction of cost variation and both of the endpoints.

This is made precise by the following theorem.
Theorem 4.4.1. Suppose that $C(x)$ is the cost associated with a variational problem of the form (4.3.1). If $W$ is such that $C(x)=C\left(W W^{T} x\right), \forall x$, then there exist an optimal path $x^{*}(t)$ of this problem and a path $a(t)$ such that

$$
\begin{equation*}
x^{*}(t)=W a(t)+y s(t) . \tag{4.4.1}
\end{equation*}
$$

Proof. The proof is a straightforward application of the calculus of variations. First, assuming that the matrix of $W$ has $d$ columns, we can choose coordinates such that the cost function $C$ only depends on the first $d$ coordinates. We therefore assume that $\partial C / \partial x_{i}=0 \forall i>d, \forall x$. As before, we will refer to coordinates on which the cost function does not depend as cyclic coordinates.


Figure 4.1: Illustration of Theorem 4.4.1

We then define the Lagrangian

$$
L(x, \dot{x})=\|\dot{x}\| C(x)
$$

and apply the Euler-Lagrange equations, yielding

$$
\frac{d}{d t} \frac{\partial L}{\partial \dot{x}_{i}}-\frac{\partial L}{\partial x_{i}}=\frac{d}{d t}\left[\frac{\dot{x}_{i}}{\|\dot{x}\|} C(x)\right]-\|\dot{x}\| \frac{\partial C}{\partial x_{i}}=0 .
$$

For any cyclic coordinate, we can substitute $\partial C / \partial x_{i}=0$, implying that $\forall t$, and for some yet-unknown constants $k_{i}$,

$$
\begin{equation*}
\frac{\dot{x}_{i}}{\|\dot{x}\|} C(x)=k_{i} . \tag{4.4.2}
\end{equation*}
$$

As discussed in Section 3.2.3, the invariance of the cost functional to reparameterization of time allows us to assume that $\|\dot{x}\|=\alpha$, where $\alpha$ is some irrelevant constant. (4.4.2) then yields $N-d$ independent, separable ODEs for each cyclic coordinate. Integration of these produces

$$
\begin{align*}
x_{i}(t) & =\alpha k_{i} \int_{0}^{t} \frac{1}{C(x(t))} d t \\
& =\alpha k_{i} F(t) \tag{4.4.3}
\end{align*}
$$

defining $F(t)$ in the last step.

We then write the path as a linear combination of the standard basis vectors, $e_{i}$ :

$$
x(t)=\sum_{i} x_{i}(t) e_{i}
$$

and substitute (4.4.3) into this equation, which produces

$$
x(t)=\sum_{i=1}^{d} x_{i}(t) e_{i}+\sum_{i=d+1}^{N} \alpha k_{i} F(t) e_{i} .
$$

We now solve for the $k_{i}$ in this expression to express the last basis vector in terms of the problem data. The $k_{i}$ can be computed from the final conditions; i.e., $x_{i}(1)=\alpha k_{i} F(1)$. Substitution of this expression and simplification then yields

$$
x(t)=\sum_{i=1}^{d} x_{i}(t) e_{i}+\frac{F(t)}{F(1)} \sum_{i=d+1}^{N} x_{i}(1) e_{i} .
$$

This expression can rewritten by expressing the basis $e_{i}$ in terms of the basis $w_{i}$ to yield the desired expression, with $\left(x_{1}(t), \ldots, x_{d}(t)\right)$ becoming $a(t)$, and $F(t) / F(1)$ becoming $s(t)$.

Theorem (4.4.1) suggests another potential avenue for the algorithmic exploitation of low-dimensional structure to solve (4.3.1). Instead of computing the high-dimensional value function via a low-dimensional parameterization, and subsequently integrating the optimal control law in a high-dimensional space; we can equivalently find a path directly in the low-dimensional subspace in which it is known to be contained. This is usually the preferred solution method, though computing the high-dimensional value function is useful in some contexts as well.

## Chapter 5

## Discovering structure in physical data

The application of learning and inference with structured Lagrangians to the specific problem of the analysis of high-dimensional human motion capture data is now detailed, first reviewing the issues surrounding the problem to be solved.

### 5.1 Problem statement

The problem to be addressed in this section is that which was already described in some detail in Section 1.2 and Fig. 1.2. Summarizing that description, we suppose we are given training data consisting of high-dimensional vectors representing observations of a number of physical trajectories sampled at discrete times, where these observations specifically consist of the positions of a variety of markers located on the body of a human actor performing various motions. We would then like to learn a model of the motions from this data and subsequently use it to reconstruct novel trajectories from their endpoints alone.

The primary problem to overcome in dealing with this data is its high dimensionality: each observation is represented as a 990-dimensional vector of real numbers. Having a means to deal this issue is hence an immediate prerequisite of any learning procedure applied to this problem.

Previous work in this field may be divided roughly into two camps: those that leverage detailed physical models, and those that do not. Among the latter are GPDMs [102, 103], which were discussed at some length in Section 2.3.2. A fundamental difference between GPDMs and the kinetic Lagrangian model employed here lies in their respective representations as graphical models. The GPDM's graphical model is illustrated in Fig. 5.1, while the kinetic Lagrangian's graphical model is illustrated in Fig. 3.3. Extra conditional


Figure 5.1: Graphical model associated with the GPDM
dependencies in the GPDM compared to the HMM 2.1 are due to the fact that inference in the GPDM automatically affects the implied dynamics and observation models. Another interesting point of difference between the GPDM and the kinetic Lagrangian models, is that the conservation laws in the latter produce simplified dynamics for the greater part of the state, whereas in the GPDM, the high-dimensional observations are modeled as images of low-dimensional trajectories. This difference is ultimately due to the fact that the kinetic Lagrangian is able to obtain a more appropriate graphical model for systems to which its assumptions apply.

The other relevant line of work is that which uses more detailed models of human motion [18, 59, 17, 19], typically used in applications of tracking people in video sequences. These approaches use simplified human kinematics and dynamics models, sometimes obtained from the biomechanics literature. Although some parameters of these models are estimated from data, the extent to which they depend on specific prior knowledge about this domain limits their general applicability compared to the kinetic Lagrangian method presented here.

### 5.2 Experiments with human motion capture data

Given the learning method presented in Chapter 2 and the inference techniques of Chapter 3, applying kinetic Lagrangian modeling to physical sequence data is straightforward. An overview of the procedure is depicted in Fig. 5.2. We assume that we are given sampled observations of trajectories expressed in some arbitrary $z$ coordinates. FCA is then applied to learn a basis for the subspace in which the cost function $m(x)$, varies. These

observe trajectores

learn "features"
(acyclic basis)

regress cost from features

recover model that generated trajectories

Figure 5.2: Overview of method for sequence data modeling
basis vectors may be thought of as features from which the cost can be regressed via the Eq. (3.2.2), assuming that all trajectories are equally energetic, which allows us to observe $m(x)$ given the local velocities of the trajectories. Finally, novel trajectories interpolating arbitrary endpoints can be recovered by exploiting the low-dimensional structure of $m(x)$ via one of the approaches described in Chapter 4.

This methodology was consequently applied to a subset of the CMU human motion capture database. The subset selected consisted of 24 disjoint sequences depicting one of three actions: jumping jacks, side twists, and knee-elbow touches. Three of these were held out for evaluation, one for each action; and the remaining 21 sequences were used for training; i.e., the top three Fermat components of the data were found, then $m(x)$ was learned in the resulting three-dimensional space with Nadaraya-Watson kernel regression.

Two other simple methods were implemented for comparison-one that takes into account the boundary conditions, but not the training data; and one that takes into account the training data, but not the boundary conditions. The first is simply linear interpolation between the start and end poses. In the second method (henceforth referred to as the "integration method"), a mapping was learned from poses in the original high-dimensional space to velocities observed at those poses using Nadaraya-Watson kernel regression. Given an initial condition, these velocities were integrated to obtain a trajectory. Results are shown in Figures 5.3 and 5.4.

As these figures attest, only the knee-touch behavior is adequately described by linear interpolation. Linear interpolation on the jumping jack produces an unnatural motion that lifts the hands vertically up, staying close to the body. It also fails to reconstruct the small hop present in the motion, as can be seen by observing the trajectories of the feet. In the side twist, linear interpolation causes the arms to shrink together, meet at the center of the body, and expand out again. The integration method completely fails to


Figure 5.3: Visualization of experiments in producing novel trajectories from pairs of key frames. Leftmost image in each set is the true, held-out trajectory. Other images depict the output of different learning methods applied to reconstruct the trajectory from the key frames.
produce a plausible trajectory for the knee touch, getting stuck in an unlikely pose. Its performance is better in the other two cases, although it never attains the final pose, as seen clearly in Figure 5.4. The individual appendages also appear less coordinated in this method; in the jumping jack, the right arm initially falls slack by the side, and lags behind the left. During the side twist, the arms initially twist out of the desired plane of rotation before rotating about the desired axis. The integration method did seem to capture some initial small details of the knee-touch well, but completely failed to reconstruct the main knee-touching behavior.

The results obtained with FCA address most of these issues. In each case, the trajectories reached the end point with very little error, and the interpolating sequence captured the main features of the motions very well. The arcs traveled by the arms are well-defined, as well as the small jump evident in the feet, though it is not quite as pronounced as in the original sequence. The rigid relative positions of the arms are preserved well in the side twist, and the nearly-linear knee touch motion is also executed well, except for some incidental nonlinear motion of the arms.

The kinetic Lagrangian methodology was then used to reconstruct a novel action sequence not present in the original training set. The initial pose given was the initial pose of a jumping jack, and the final pose was the final pose of a side twist. The result is shown in Figure 5.5. As expected, the resulting sequence resembles half of a jumping jack, followed by a side twist. The sequence appears mostly natural, except that the right arm bends backward in an awkward way upon starting the side twist.

Finally, FCA was applied to data collected from a realistic physical simulation of a humanoid robot based on the Open Dynamics Engine. A hand-coded controller guided the robot through a normal walking gait while the Euclidean coordinates of major joints on the robot were logged. The data were then partitioned at each footfall, training on individual steps made in the course of the gait. One of these steps was held out from the training set and the the motion between the two footfalls was subsequently synthesized based on the learned kinetic Lagrangian.

Linear interpolation actually managed to account for much of the overall motion, which would seem to explain the minimal quantitative difference between linear interpolation and the kinetic Lagrangian approach (Fig. 5.7). However, visualizing the resulting synthesized trajectories (Fig. 5.6) shows that linear interpolation failed to reconstruct the parabolic trajectory of the flight leg, resulting in the foot dragging on the ground. The kinetic Lagrangian was able to successfully capture this important detail by finding a nonlinear trajectory interpolating the start and end poses. The integration method produced results that rapidly diverged from a feasible gait, as seen in Fig. 5.7.


Figure 5.4: Sum of squared errors between true pose and corresponding poses in the reconstructed sequence. Boxes mark the trajectory obtained with linear interpolation, triangles mark the trajectory obtained with the nonlinear regression + integration method, and circles mark the trajectory obtained via FCA.


Figure 5.5: A few views of an experiment reconstructing a novel action sequence. Sequence begins with a jumping jack and ends with a side twist.


Figure 5.6: Visualization of experiment reconstructing a portion of a robot gait sequence from starting and end poses. Note especially the trajectories of the left foot.


Figure 5.7: Quantitative results from the gait reconstruction experiment. Plot shows sum of squared errors between true pose and corresponding poses in the reconstructed sequences.

## Chapter 6

## Future directions: non-kinetic Lagrangians

Thus far, we have only discussed modeling with kinetic Lagrangians, which might be considered just one particular kind of structured Lagrangian. This particular form was motivated principally from a pragmatic viewpoint, in that kinetic Lagrangians admit efficient inference and learning algorithms. This is not to say, however, that kinetic Lagrangian modeling is not well-motivated physically. We already saw that the trajectories associated with kinetic Lagrangians are simply those that behave as rays of light in high-dimensional spaces, traveling through inhomogenous media. This assumption turns out to be far less restrictive than it appears, thanks to Hamilton's optical-mechanical analogy, which allows us to draw a precise correspondence between the trajectories of arbitrary physical systems and those taken by rays of light in (a higher-dimensional generalization of) optics [7].

Although this provides reassurance that it is safe to think of arbitrary physical trajectories as optical paths, it does not necessarily follow that a given physical model might be written in simplest terms as a kinetic Lagrangian. One could therefore imagine that using another form of structured Lagrangian might be more appropriate in certain circumstances. Whatever precise form of the Lagrangian might be used, the key concept would remain the same: to deduce conservation laws from symmetries of the Lagrangian and to subsequently exploit this structure to simplify the learning and inference problems. This chapter briefly demonstrates how such extensions might be derived by studying another type of structured Lagrangian.

Consider the following Lagrangian, for $x \in \mathbb{R}^{N}$ :

$$
\begin{equation*}
L(x, \dot{x})=\dot{x}^{T} \dot{x}-\Phi(x) . \tag{6.0.1}
\end{equation*}
$$

Supposing that this Lagrangian does not depend on some coordinate $x_{i}$, the corresponding
conservation law is simply conservation of momentum:

$$
\begin{equation*}
\dot{x}_{i}=v_{i} \tag{6.0.2}
\end{equation*}
$$

for some trajectory-specific constant $v_{i}$. Proceeding in a way analogous to the development of FCA, we might imagine a scenario where we were given data observed in a coordinate system a rotation away from one in which only a few coordinates would not obey conservation of momentum. It is not difficult to show that the analog of FCA in this case is simply PCA on the observed velocities.

With that issue settled, it remains to be seen how to go about estimating the Lagrangian, or equivalently, the potential function $\Phi$. Since this model implies simply that

$$
\begin{equation*}
\ddot{x}=-\nabla \Phi, \tag{6.0.3}
\end{equation*}
$$

the most straightforward learning model would entail assuming that we directly observe the accelerations of the trajectory data (albeit in some a-priori unknown basis) and somehow use this to infer $\Phi$.

Should we choose to implement such a scheme, some problems arise due to the extra derivatives that appear here with respect to the FCA case; in the latter, we only needed to observe the norm of the velocities of the trajectories, and this information could be used directly to estimate the Lagrangian. In this case, we must assume that we directly measure full accelerations-not just their norms-and subsequently, we can only use this to estimate the gradient of the potential function. The requirement that we directly measure accelerations is particularly troublesome, as estimating these from position data can prove very inaccurate.

A possible remedy to such woes is the use of Bayesian estimation via Gaussian processes (GPs), which enable us to put smooth priors directly onto function spaces. Furthermore, (differentiable) Gaussian processes are closed under differentiation, which allows us to perform inferences jointly involving these functions and their derivatives [86]. Therefore, GPs enable us to put a smoothness prior directly on the space of potential functions and perform inferences over the posterior distribution of potential functions given observations of its gradient at various locations. Similarly, we can address the noisy acceleration problem by modeling each trajectory as a smooth Gaussian process in time, subsequently inferring posterior distributions over accelerations given observations of positions.

To illustrate the feasibility of these ideas, this method was applied to a subset of the CMU motion capture database consisting of a single motion (jumping jacks). A lowdimensional subspace was inferred via PCA on velocities, as motivated above, and GP regression was used to infer novel accelerations in this subspace given training accelerations;
which in turn, were obtained as mean estimates from GPs in time, as described above. All GP hyperparameters were obtained by training to maximize likelihood of the observations. Inferred accelerations were then used to integrate novel trajectories starting with known locations and velocities. For comparison, a similar procedure was implemented whereby the subspace was learned via PCA on observed positions, the GP regression put a smooth prior directly on accelerations (as opposed to potentials), and the training accelerations were obtained via finite differences.

Figures 6.1 and 6.2 give a qualitative comparison of these methods. The figures use identical visual conventions, except that Fig. 6.1 shows contour lines of the mean potential function, obtained via GP inference given the observed accelerations. Since the other method did not model accelerations as arising from an underlying potential function, a potential function estimate could not be obtained in this case. Both learned force fields exhibit similar properties, though the conservative field (i.e., the one modeled as a potential function gradient) ostensibly appears more smooth, especially far from the training data. The novel trajectories induced by the conservative field also seemed to better follow the training trajectories, though this observation is moderated by the fact that the shown projections are merely two-dimensional projections of 990-dimensional trajectories.

Practically speaking, the kinetic Lagrangian model is probably better suited to the previously examined problem of reconstructing trajectories from endpoints, since the method described here does not admit a similar result as far as global optimization of the Lagrangian boundary value problem is concerned. However, the probabilistic nature of the conservative potential model may prove useful as a basis for future applications in Bayesian filtering; for instance, human pose tracking from video streams often leverages a probabilistic forward model of dynamics, which could be provided by a method such as that presented here.


Figure 6.1: Visualized result of Bayesian estimation of non-conservative force field with predicted trajectories. Solid orange lines show 2D projections (onto PCA subspace) of observed, high-dimensional training trajectories. Large arrows show estimated training accelerations. Broken lines show predicted trajectories.


Figure 6.2: Visualized result of Bayesian estimation of conservative force field with predicted trajectories. Solid orange lines show 2D projections (onto learned subspace) of observed, high-dimensional training trajectories. Large arrows show estimated training accelerations. Broken lines show predicted trajectories.

## Part II

## Planning with structured Lagrangians

## Chapter 7

## Motion planning in high-dimensional spaces

The second part of this work concerns the problem of motion planning in high-dimensional spaces. It will be seen that the analysis of structured Lagrangian systems presented in the first part of this work provides insight that opens fruitful avenues for research in this domain. As in the last part, it is submitted that the key to developing effective algorithms in this domain is learning and exploiting structure automatically and in a generic way.

### 7.1 Problem definition

The motion planning problem, in fact, consists not of a single problem, but a fairly diverse collection of problems falling under that general heading. As such, a precise definition of this problem, at least in the general sense, will not be given. Rather, it will suffice to say that motion planning typically refers to the issue of finding a continuous trajectory for some physical entity subject to boundary conditions and various other constraints, usually such that the entity avoids certain regions of space categorized as obstacles. The motion planning problem is distinguished from other planning problems by its physical character, which implies an underlying setting in a continuous domain [53].

The multitude of motion planning problems and algorithms in existence today are characterized by various distinctions, a few of which are described here.

## Completeness

A motion planning algorithm is said to be complete if it always either returns a feasible path or accurately reports that no such path exists. Very few motion planning algorithms
are of this variety, as the problem is NP-hard if formulated reasonably. Resolution completeness is a weaker notion in the sense that a resolution-complete algorithm is allowed to run indefinitely in the case that no solution exists, but it must still terminate in finite time if a solution does exist. Probabilistic completeness is weaker still in the sense that algorithms need only to terminate with probability approaching one as runtime approaches infinity [53]. In practice, probabilistic and resolution guarantees must be taken cum grano salis due to the failure of these notions to exclude unbounded computation.

## Representation

Different formulations of the motion planning problem may involve different representations of the problem data. Obstacles, for instance, might be specified geometrically in terms of semi-algebraic sets, or we may be given only an oracle that can be consulted to determine whether any specific point lies in the free space.

A subtle but critical point is that the representation of the problem data is intricately linked to the computational complexity of the problem, considering that the only meaningful way to express complexity is in terms of the input length. In this sense, one can reduce the computational complexity of a given algorithm simply by making its input representation less efficient [54].

Taking this factor into account, one might say that the only trustworthy complexity results are those that are specified in terms of reasonably efficient representations of the input data. This makes the use of compressed cost functions, as first described in Chapter 4, particularly appealing.

## Optimality

In the basic motion planning problem, it is only required that a feasible trajectory be returned; i.e., one that resides entirely within the free space. An alternate criterion requires that the trajectory be optimal in some sense; for example, we may penalize paths that cross obstacles and require that the algorithm return the path of minimum accumulated penalty.

## Holonomy

Typical constraints in motion planning problems may be characterized as being either holonomic or non-holonomic. Informally, we may think of holonomic constraints as those that only constrain the path to lie on some submanifold of the configuration space, whereas nonholonomic constraints are those that cannot be expressed in such a way. A standard example of a non-holonomic constraint is a minimum curvature constraint.

### 7.2 General methods

Various algorithms exist catering to different aspects of the motion planning problem. Some of the more notable and popular approaches are described here, especially with regard to the current context of high-dimensional motion planning.

### 7.2.1 Local optimization

One approach to overcoming the curse of dimensionality in an optimal motion planning context is to settle for obtaining a path that locally optimizes the cost functional in the variational sense. Such is the approach taken by the elastic band method [71] and the more recent method known as CHOMP [74]. Although the gradient-based nature of these algorithms permits them to be applied in high-dimensional spaces without much difficulty, the price paid for this advantage is primarily that the eventual solution remains homotopic (i.e., continuously deformable) to the initial solution. Though such methods can also be extended to apply to a setting of global optimization via Monte Carlo methods, doing so offers little in the way of computational complexity guarantees.

### 7.2.2 Potential fields

Another class of methods is based on the idea of having trajectories approximately follow the negative gradient of an artificial potential function. This approach is similar to that obtained via optimal control, in which the optimal control is that which aligns itself best with the negative gradient of the value function. Potential fields, on the other hand, need not admit any notion of optimality, thus simplifying the problem appreciably.

A significant deficiency of the potential field method in its original incarnation [49] is the possibility of the appearance of local minima in the potential. It is undesirable that the agent encounter one of these outside the goal region, as no progress is possible at these points. A partial remedy is simply to apply randomization at local minima in the hope that random walks will eventually take the agent away from the minimum's basin of attraction [9].

A more elegant approach, however, is to construct potential fields that possess only a single local minimum, located at the desired goal. Perhaps the most well-known of such methods (Rimon-Koditschek [79]) proceeds by explicitly generating such Morse functions in model sphere worlds, which can subsequently be deformed continuously into environments of a much more complex nature. Similar potential functions can be developed starting from the theory of harmonic functions [25].

An interesting property of the Rimon-Koditschek construction is that its computationally complexity is not ostensibly affected by the configuration-space dimension and can be
shown to be efficient in terms of the input obstacle descriptions. This aligns with the previously noted intuition that exponentially inefficient representations of the problem data are at least partially to blame for the inefficiency of algorithms based on them. Unfortunately, this efficiency comes at the expense of needing to generate geometric descriptions of obstacles in configuration space (as opposed to the workspace). For many high-dimensional problems-for example, planning for a robot arm - this is extraordinarily difficult to do, if not outright intractable [53]. For this reason, exact potential fields are not often used in these types of problems.

### 7.2.3 Sampling-based planning

Faring much better in this respect are the enormously popular class of sampling-based planning methods, such as Rapidly-exploring Random Trees (RRT) [52], Probabilistic Roadmaps (PRM) ([48, 6]), Expansive Space Trees [45], and SBL [81], just to name a few of the most well-known of these. The majority of these methods grow one or more trees in configuration space consisting of discrete samples connected by simple paths. The trees are grown with a Voronoi bias that encourages expansion in directions that have not been explored previously. This process is continued until the tree grows close enough to the goal region to terminate the search. Numerous variations on this theme have spawned almost countless works (many of which are summarized in [90]).

The rapid proliferation of these approaches is certainly not without merit, as they represent a clear advance in the practice of solving high-dimensional motion planning problems. In addition to their reliability, most sampling-based planners are very "data friendly" in that no direct obstacle description is required other than an oracle telling whether any given configuration is in collision. This black-box representation makes it very straightforward to apply sampling-based planners to new domains by leveraging existing implementations.

### 7.2.4 Deterministic planning

The final class of algorithms described here are loosely characterized as deterministic planners to differentiate them from the randomized planners described in Section 7.2.3. This class of planning algorithms consists of those ultimately derived from the $\mathrm{A}^{*}$ algorithm [40], which is similar to Dijkstra's algorithm, but explores states in a different order. Namely, at each iteration, the state explored is chosen to minimize the distance from the start state plus the value of a heuristic that lower-bounds the distance of the state to the goal. Crucially, $A^{*}$ is still guaranteed to find the optimum path to the goal, while hopefully expanding much fewer states than Dijkstra's algorithm would.

Modern descendants of A* exist to satisfy a variety of niches. The simplest, weighted $A^{*}$ simply multiplies the heuristic by some value greater than one, which encourages the search to "follow" the heuristic, trading off optimality for decreased execution time. D* [87, 88] generalizes $\mathrm{A}^{*}$ to allow incremental updates to the costs while preserving optimality, a feature particularly useful for robots exploring novel environments in real-time. In a similar vein, ARA* $[57,56]$ provides an anytime variant of A* by producing a series of solutions of using decreasing heuristic weighting factors, producing successively better solutions. $\mathrm{R}^{*}$ [58] combines aspects of $\mathrm{A}^{*}$ and randomized planning methods by performing a series of deterministic searches to random locations.

Such methods have been applied with success to relatively high-dimensional motion planning problems, such as planning for legged locomotion [101, 106] and manipulation planning [24, 22]. The key to such success is usually the existence of a strong heuristic function, which can be regarded either as an advantage or a disadvantage, in the following sense: being able to leverage prior knowledge in the form of heuristic function is a great advantage when a good heuristic is available; when it is not, then the reliance of these methods on a good heuristic becomes something of a liability.

Compared to sampling-based planning methods, this strong reliance of deterministic planning algorithms on a heuristic makes them generally more difficult to apply with success to high-dimensional problems. When a good heuristic is available, however, deterministic planning methods generally find much higher-quality solutions than do most sampling-based methods, which do not attempt any kind of optimization. This benefit, however, comes at the expense of increased planning times for high-dimensional problems.

### 7.3 Discovering and exploiting structure

The key to the methods described in this work is the discovery and exploitation of structure in some way, which is, at best, a nascent and under-appreciated idea in motion planning. We therefore briefly review known work on this subject.

### 7.3.1 Distance and heuristic functions

Of the methods described so far, only the deterministic planning algorithms have explicitly leveraged problem-specific (or more accurately, instance-specific) structure in some way, in the form of the heuristic function. The heuristic function allows the practitioner to inject problem-specific structure into the solver in a very convenient way, such that leveraging more knowledge (in the form of a tighter lower bound) leads to concrete gains in efficiency.

Similarly, in sampling-based planning, a distance function is usually employed to determine which node to expand next, much as in deterministic planning. Therefore, a practitioner can similarly use define the distance function as a way of injecting prior knowledge, though the ramifications of this are less explicit than in the deterministic case.

## Learning heuristics

An obvious question to ask at this point is whether such heuristics could be automatically generated or learned instead of designed by human experts. The answer constitutes a classic line of AI research with many intricacies that will not be discussed at length here, partially because the problems formulated in this context were always of a purely symbolic nature rather than a physical nature, making it difficult to connect this work in a meaningful way to the problem of motion planning.

That said, the classic work on learning heuristics did produce some interesting results of a general nature. A very early and basic idea out of which these results grew was that the solution of relaxed problem may serve as an admissible heuristic (i.e., one that lower-bounds the true value function) [34]. This sparked interest in the concept of possibly generating these relaxed problems in an automated way and subsequently solving these problems using $A^{*}$, perhaps even in a recursive way. This was shown by Valtorta to be a hopeless endeavor under certain conditions [95], implying in some sense that the relaxed problems should be far simpler to solve than the original in order for this approach to prove cost-effective. Notable work [63, 70] achieved this by applying various heuristic transformations to a problem specified in a $S T R I P S$-like formal representation in order to automatically generate admissible heuristics.

Pattern databases [27] constitute another interesting approach to the automatic generation of heuristics. This method identifies a number of target patterns or subgoals that must be achieved in order to find a solution. These subproblems are formulated so as to be relaxations of the original problem that admit efficient solutions. The subproblems can then be solved en masse for different initial configurations, storing the resulting optimal values in large look-up-tables. At search time, the solutions to these subproblems are looked up for the current configuration, and the maximum value of all these is used as the heuristic. Although this and related work produced many successes in solving discrete problems such as sliding puzzles and the Rubix cube, these successes never seem to have translated into benefits for continuous domains.

By contrast, recent work has explored the heuristic learning issue from a structured learning perspective to learn effective, albeit inadmissible heuristics for footstep planning for a legged robot [73]. This work employed structured learning to "teach" a lowdimensional planner to generate plans similar to those generated by a high-dimensional
planner. The result of the low-dimensional planner was then used as a heuristic for the high-dimensional planner, with positive results.

### 7.3.2 MDP homomorphisms

Research in reinforcement learning has also recently studied the issue of structure in planning problems and how it might be discovered and exploited, usually in the context of MDPs. One early and particularly insightful result in this area, due to Zinkevich [105], proved that symmetry of the MDP value function follows necessarily from symmetry of the MDP. The precise definition of symmetry in this case involves defining an equivalence relation on state-action pairs to express the intuitive idea that "being here and doing this" is equivalent to "being there and doing that."

In the deterministic case, this result implies the completely intuitive fact that shortestpath value functions on graphs symmetric about the goal state(s), are themselves symmetric. However trivial, this can be thought of as one possible discrete analog of the continuous result stated in Theorem 4.3.1.

More recent work in this field has generally focused on the study of MDP homomorphisms [77], which are essentially formal "analogies" between MDPs, possibly of different state spaces, that preserve dynamics under the analogies. Ultimately, a limitation of such approaches is that it is NP-hard to find such analogies in the general case, owing to the combinatorial nature of the problem induced by the discrete state space. It is primarily for this reason that current work on symmetry and homomorphisms in MDPs is not directly applicable to motion planning. This is in contrast to the methods presented here, which exploit the vector space structure of continuous spaces to efficiently exploit and learn symmetries in motion planning problems.

### 7.3.3 Symmetric optimal control systems

The final instance of symmetry exploitation in a planning context comes from optimal control, where Grizzle and Marcus [39] proved a very general result for optimal control systems possessing symmetries.

Grizzle's result can be thought of the continuous, optimal-control equivalent of Zinkevich's result [105] for discrete MDPs; i.e., the former shows that the optimal controls for systems possessing differentiable symmetries, also obey the same symmetries. This is accomplished by the use of a concept similar to the MDP homomorphism whereby the symmetry is shown to induce a lower-dimensional control system on the state-action space modulo the symmetry. It can then be shown that the action-parameterized value function (c.f., the MDP's Q function) is invariant under projection onto this quotient control
system, allowing one to compute a value at an arbitrary point by "looking up" the value of the point projected onto its equivalence class in the quotient control system, in a way exactly paralleling the results in Chapter 4.

This result can therefore be considered in some sense a generalization of those presented in Chapter 4 to arbitrary symmetries in arbitrary nonlinear optimal control systems. However, the result itself speaks nothing of the important issue of how to identify and exploit these symmetries in practice. By contrast, the specific kind of symmetries discussed in Chapter 4 (i.e., translational symmetries, or low-dimensional structure) are particularly well-suited to algorithmic exploitation - a fact that will be demonstrated by the methods given in the following chapters.

## Chapter 8

## Planning with structured cost functions

The ultimate goal of the remainder of this work will be to show how to automatically identify and exploit the low-dimensional structure of motion planning problems. However, in many cases, it may be that such structure is known a-priori to the practitioner and as such, need not be learned. This chapter accordingly explores different potential ways of exploiting structure when it is known.

### 8.1 Heuristics for deterministic search

The results derived in Chapter 4 may be applied immediately to obtain globally optimal motion plans for a very limited class of high-dimensional problems. Particularly, these are the holonomic motion planning problems that can be described as the solutions to the variational problem (4.3.1), such that the cost function $C(x)$ only varies in a small subspace of the ambient space.

Generally, however, it is more likely that the problem designer is aware of some lowdimensional structure that may not hold exactly; i.e., the cost function may approximately vary principally in a low-dimensional subspace, but it may still have variation in all dimensions. Furthermore, the problem may possess non-holonomic constraints.

This section describes a very simple way of handling such issues while still leveraging the known low-dimensional structure of the problem. Specifically, it is proposed to define as a heuristic (in the sense of deterministic planning) the optimal value of a relaxed version of the problem, much as described in Section 7.3.1. If the relaxed problem has low-dimensional structure, then its optimal value function may be computed efficiently via the results of Chapter 4. It is then hoped that doing so would produce a very strong heuristic that would
aid deterministic search in high-dimensional spaces. Such a heuristic will be referred to as the Fermat heuristic in this work.

### 8.1.1 Properties of the Fermat heuristic

This section discusses the details of this approach, simultaneously proving various nice properties of the Fermat heuristic. To do so, we will need to recall the following properties of A* search [67][35]:

- A* with an admissible heuristic is guaranteed to find the optimal solution, if it terminates
- A* with a consistent heuristic never "opens" the same node twice
- If A* uses an heuristic consisting of admissible heuristic weighted by some factor greater than one, then it is guaranteed to terminate with a solution within that factor of the optimum solution

We will first see that the Fermat heuristic is consistent and admissible, as long as we construct it carefully.

Definition 8.1.1 (Fermat heuristic). Consider a constrained path planning problem with cost function $C(x)$, and suppose we are given a cost function $B_{d}(x): \mathbb{R}^{N} \rightarrow \mathbb{R}^{+}$satisfying

$$
\begin{equation*}
B_{d}(x) \leq \min _{x_{d+1}, x_{d+2}, \ldots, x_{N}} C\left(x_{1}, \ldots, x_{d}, x_{d+1}, \ldots, x_{N}\right) \tag{8.1.1}
\end{equation*}
$$

and $\partial B_{d} / \partial x_{i}=0, \forall i>d, \forall x$. Then a $d$-th order Fermat heuristic $h_{d}\left(x ; B_{d}\right)$ for this problem is given by the optimal value of the unconstrained path planning problem with cost function $B_{d}(x)$.

Here $B_{d}(x)$ is a function chosen to lower-bound the original cost function by minimizing $C(x)$ over all but the first $d$ dimensions. Hence, $B_{d}(x)$ is a low-dimensional approximation of $C(x)$ whose optimal paths are hopefully similar to those of $C(x)$. Figure 8.1 gives a concrete example of this approximation.

Theorem 8.1.2. Any Fermat heuristic $h_{d}\left(x ; B_{d}\right)$ of a path planning problem is an admissible heuristic for that problem.

Proof. Denote by $V_{C}(x)$ the value function of the original constrained problem, and denote by $V_{U}(x)$ the value function of the original problem after relaxing its constraints. Since


Figure 8.1: Example showing a one-dimensional approximation $B_{1}(x)$ of a two-dimensional cost function $C(x)$. $B_{1}(x)$ is formed by minimizing $C(x)$ over $x_{2}$. Dashed lines show optimal paths with respect to corresponding cost functions.
$\forall x, B_{d}(x) \leq C(x)$, and since the value of the unconstrained problem must be less than that of the constrained problem, we have

$$
h_{d}\left(x ; B_{d}\right) \leq V_{U}(x) \leq V_{C}(x)
$$

Another nice property of the Fermat heuristic is that it is strictly better than the standard Euclidean distance heuristic, in the following sense:

Theorem 8.1.3. The greatest 0 -th order Fermat heuristic of a particular problem is exactly equal to the Euclidean heuristic for that problem, and the greatest d-th order Fermat heuristic is always greater than or equal to the Euclidean heuristic.

Proof. By "greatest $d$-th order heuristic" in the statement above, we mean the heuristic that uses the greatest lower bound $B_{d}$ possible. First note that in the case $d=0, B_{d}$ is a constant. The greatest 0 -th order Fermat heuristic therefore has

$$
B_{0}(x)=\min _{x} C(x)=B_{0}
$$

The value of the heuristic is consequently determined as the optimal value function of a 0 -dimensional cost function; i.e., constant cost. The path minimizing this cost is trivially a straight line between the endpoints $x$ and $y$, and its cost is equal to $\|x-y\| B_{0}$. The
greatest 0-th order Fermat heuristic is therefore given by

$$
h_{0}\left(x ; B_{0}\right)=\|x-y\| B_{0}
$$

which is exactly equal to the Euclidean heuristic for this problem. Note that the scaling by $B_{0}$ ensures that the Euclidean heuristic is as tight as possible.

The second part of the theorem follows easily from the fact that the value of the path planning problem with cost function $B_{d+1}$ is always greater than or equal to the value of the path planning problem with cost function $B_{d}$.

Finally, the Fermat heuristic is consistent, as shown by the following.
Theorem 8.1.4. Let the discrete cost function $C^{\prime}(x, y)$ be equal to the cost of some path passing through $x$ and $y$ under cost function $C$. Any Fermat heuristic $h_{d}\left(x ; B_{d}\right)$ of a problem with cost function $C$ is a consistent heuristic for the problem when $A^{*}$ is applied with discrete cost function $C^{\prime}$.

Proof. Note that in the statement of the theorem a distinction is made between the differential cost function used in the continuous problem and the discrete cost function used in the A* solution of the problem.

Denote by $h_{d}\left(x ; y, B_{d}\right)$ the value of the Fermat heuristic at $x$ with a different goal $y$. First, note that

$$
h_{d}\left(x ; B_{d}\right) \leq h_{d}\left(x ; y, B_{d}\right)+h_{d}\left(y ; B_{d}\right)
$$

since $h$ is a value function that consequently obeys the triangle inequality. Denoting by $V(x ; y)$ the optimal cost to travel from $x$ to $y$ in the original problem, we then have

$$
\begin{align*}
h_{d}\left(x ; B_{d}\right) & \leq V(x ; y)+h_{d}\left(y ; B_{d}\right) \\
& \leq C^{\prime}(x, y)+h_{d}\left(y ; B_{d}\right) \tag{8.1.2}
\end{align*}
$$

The consistency and admissibility of the Fermat heuristic when employed in the manner described above ensure that A* is guaranteed to find the optimal solution (up to sampling resolution) of our problem without "backtracking." In practice, however, we might find that we are willing to give up these guarantees of optimality in exchange for decreased computation time. The last property of $\mathrm{A}^{*}$ mentioned above is therefore particularly interesting. It provides a quantitative suboptimality guarantee in case we decide to inflate (scale up) the heuristic. Inflating the heuristic biases A* to explore the states with lower heuristic values, generally causing it to terminate faster, assuming that the heuristic does
decrease monotonically and continuously towards the goal. Note that were it not for the admissibility of the Fermat heuristic, we would not have this reassuring suboptimality guarantee.

### 8.1.2 The multi-agent towing problem

The following motion planning problem exemplifies a case where low-dimensional structure is known to be present a-priori. Suppose we have $N$ agents traveling in a two-dimensional space. The joint configuration of the agents at any given time is specified by a vector $x \in \mathbb{R}^{2 N}$ with the following format:

$$
x=\left(\begin{array}{llllllll}
y_{1} & y_{2} & \ldots & y_{N} & z_{1} & z_{2} & \ldots & z_{N} \tag{8.1.3}
\end{array}\right)
$$

where $y_{i}$ is the $y$ coordinate of the $i$ th robot, and $z_{i}$ is the $z$ coordinate of the $i$ th robot. The payload location $x_{P}$ is assumed to be equal to the robots' collective center of mass:

$$
x_{P}(x)=\left(\begin{array}{ll}
\frac{1}{N} \sum_{i=1}^{N} y_{i} & \frac{1}{N} \sum_{i=1}^{N} z_{i}
\end{array}\right)
$$

We assume we are given a sampled, spatially-varying function $p(x) \in(0,1)$ interpreted as the probability that the robots' payload will escape capture per time step spent at location $x$. If $x\left(t_{i}\right)$ is a sampled path of length $T$, then (assuming independence in time), the probability that the payload will evade capture over the entire path is given by

$$
\prod_{i=1}^{T} p\left(x\left(t_{i}\right)\right)
$$

If we wish to find a path maximizing the probability that the payload will evade capture, we can minimize the negative log of the previous expression over suitable paths. We can therefore define a strictly positive cost function $C(x)=-\log p(x)$, yielding the following objective function:

$$
J\{x\}=\sum_{i=1}^{T} C\left(x\left(t_{i}\right)\right)
$$

In order to create an informative Fermat heuristic, we need the dimension of $C$ to be low (or approximately low), in the sense that it should only depend on a small number of coordinates. Although this is not true for $C$ described in the original coordinates, it is true of $C$ expressed in a basis $B_{i}$ with

$$
\begin{equation*}
B_{1} \propto \sum_{i=1}^{N} e_{i}, \quad B_{2} \propto \sum_{i=N+1}^{2 N} e_{i} \tag{8.1.4}
\end{equation*}
$$

and the other basis vectors chosen arbitrarily to complete an orthonormal basis. Expressing our problem in these coordinates leads to a two-dimensional cost function. Finally, it is assumed that the velocities of the agents are coupled and bounded such that the aggregate norm of their velocities is equal to one (i.e., $\sum_{i=1}^{N} \dot{y}_{i}^{2}+\dot{z}_{i}^{2}=1$ ). This constraint can be thought of as a limit on the total energy expended by all of the agents at any point in time.

With just this set of constraints, we note that the problem can be solved exactly by the results of Chapter 4. However, to make the problem more realistic, we can add the constraint that no robot is allowed to come within a radius $\rho$ of any other robot. In this case, we use the high-dimensional value function only as a heuristic to guide $A^{*}$ search, as described previously.

### 8.1.3 Results

A* with the Fermat heuristic was implemented and applied to the constrained towing problem described in Section 8.1.2. For comparison, two other common heuristics were tested as well: the Euclidean distance heuristic, and what will be referred to as the naive projection heuristic.

The naive projection heuristic resembles heuristics commonly used in high-dimensional planning problems, and is based on the optimal cost required to move a subset of the coordinates to their final configurations, ignoring the required motion of the other coordinates. In our case, this heuristic was computed as the optimal cost of moving the payload to the goal, ignoring the coupling of the robots to the payload.

The Fermat heuristic is much more informative than the naive heuristic, as it does take into account the required motion of the robots relative to the center of mass in addition to the required motion of the center of mass, and is thus "informative in every dimension." The naive heuristic is informative only with respect to a two-dimensional subspace, and is completely uninformative with respect to the other $2(N-1)$ dimensions. For comparison, the Euclidean heuristic is sensitive to changes in every dimension, but it is not necessarily very informative.

The problem was first solved with $N=2$ for a simple cul-de-sac problem, using a discrete lattice with 390625 total states. This experiment was repeated five times for each heuristic, each time multiplying the heuristic by a different weighting factor $\epsilon \in[1,8]$. Results are shown in Figure 8.2a. For $\epsilon=1$, all three heuristics obtain the same optimal solution, as expected, but take comparatively many iterations to find it. Increasing $\epsilon$ causes both the Euclidean heuristic and the naive projection heuristic to find suboptimal solutions. The Fermat, heuristic, however, continues to find the optimal solution, but in progressively fewer steps. For $\epsilon=7$ and 8 , the Fermat heuristic finds the optimum in the
absolute minimum number of steps possible: the length of the path itself. The Fermat heuristic also terminated at least an order of magnitude faster than the other heuristics for the larger values of $\epsilon$.

The scalability of each method was then tested in terms of $N$, the number of robots in the problem. The projection heuristic fared the worst in this respect, in that it was unable to find a feasible solution for $N>4$ in less than 20,000 iterations, no matter what $\epsilon$ was tried. This is consistent with our previous observation that this heuristic is not informative with respect to the robots' configurations relative to their center of mass, which hampers its ability to find feasible solutions in large dimensions. Although the Euclidean heuristic was able to find feasible solutions for $N$ as great as 9 , all of these solutions were very close to linear interpolation between the start and goal configurations, essentially ignoring the cost function, and were therefore very costly. The Fermat heuristic, on the other hand, was able to find meaningful solutions for $N$ as great as 9 (see Figure 8.2b), where the number of lattice states exceeds $10^{25}$. For $N=8$ and 9 , the solutions found using the Fermat heuristic were over 30 times less costly than the solutions found with the Euclidean heuristic.

### 8.2 Randomized planning

Just as A* employs a heuristic distance function to guide the search, randomized planners usually benefit from leveraging an appropriate distance function, typically in order to implement sampling bias towards unexplored regions or towards the goal [26]. We can therefore take an approach similar to the one just presented for deterministic planning: define a cost function with low-dimensional structure, efficiently compute the value function, and use that as a distance function. The resulting distance will again be referred to as the Fermat heuristic.

The cost function in this context would typically penalize traveling through obstacles, yielding a distance that more accurately reflects the distance through the obstacle-free space. In the context of an RRT, for instance, it is hoped that using such a distance would lead to more efficient sampling compared to a naive distance function, such as Euclidean distance, since a naive distance might cause the tree to be extended in directions that would lead quickly to collision.

In the deterministic planning case, the heuristic was carefully constructed to yield provable performance bounds. Assuming this is not a concern in the randomized planning case (as is typically the case), we can be more flexible in how we obtain a low-dimensional approximate cost from our given cost function.

(a) Results of experiments with $N=2$ robots. Each point shows cost of a solution obtained versus the number of iterations necessary to obtain it using a particular heuristic and heuristic scaling factor.

(b) Scaling of performance of Fermat heuristic with problem dimension, using a very large heuristic weighting factor. Problem dimension is equal to $2 N$, where $N$ is the number of robots in the plan.

Figure 8.2: Results of multi-robot planning experiments with deterministic planner.


Figure 8.3: Illustration of how low-dimensional structure might arise in a simple arm planning problem. Locally, the cost (obstacle proximity) depends only on a one-dimensional subspace (spanned by $[a b]$ ) of the configuration space determined by the Jacobian (green arrows) and the cost gradient (red arrow).

### 8.2.1 Planning for a robot arm

To make this discussion concrete, we consider the problem of path planning for a highly articulated robot arm in the presence of obstacles. Our task is to find a collision-free path for the arm that moves the end-effector to a desired position in the workspace. Fortunately, this is not a problem that defies analysis by finding low-dimensional structure.

Fig. 8.3 shows how this structure might naturally arise in the case of a simple planar arm. just two revolute joints. We might define a cost function $C(x)$ for this problem as the inverse distance from the end-effector (located at position $x$ ) to the nearest obstacle. The local linearization of the cost function depends only on the gradient $\nabla C(x)$ of the costi.e., the direction that locally maximizes the rate at which the end-effector is brought into collision, illustrated as the red arrow protruding into the obstacle orthogonally at its surface. Assuming invertibility of the Jacobian $J=d x / d q$ at this configuration, we can express this principal cost component in configuration space by the vector

$$
\begin{equation*}
\binom{a}{b}:=J^{-1} \nabla C(x) \tag{8.2.1}
\end{equation*}
$$

This yields a decomposition of the configuration space into a subspace on which the cost depends principally (that spanned by $[a b]$ ) and an orthogonal subspace on which the cost does not depend, under this local approximation.

This argument extends to the general case, as shown by the following specific construction. First, we define the following obstacle-aware distance function (with parameters $\left.\alpha, d_{0}, \bar{d}\right)$ :

$$
\begin{align*}
h\left(q_{0}, q_{1}\right)= & \min _{q(t)} \int_{0}^{1} 1+\alpha e^{-\left(d_{\text {obs }}(q(t))-d_{0}\right) / \bar{d}} d t \\
& \text { subject to } q(0)=q_{0}, q(1)=q_{1} \tag{8.2.2}
\end{align*}
$$

This optimization problem defines the distance (i.e., heuristic function) $h$ to be the minimum cost accumulated by a path connecting the endpoints. The cost penalizes configurations exponentially according to their proximity $d_{o b s}$ to the nearest obstacle, which is defined as follows:

$$
\begin{equation*}
d_{o b s}(q)=\min _{x \in \operatorname{RobotBody}(q)} \min _{y \in \text { Obstacle }}\|x(q)-y\| \tag{8.2.3}
\end{equation*}
$$

Here $\operatorname{Robot} \operatorname{Body}(q)$ denotes the set of points inside the body of the robot when the arm is in configuration $q$, and Obstacle is the set of points inside obstacles.

In order to compute $h$ efficiently, we will see that if we make a local linear approximation to $d_{o b s}$, then $h$ is the optimal value of a path planning problem with a one-dimensional cost function. We can then exploit the low dimensionality of this cost function to compute the corresponding value function using the method described previously.

We first compute $d d_{o b s} / d q$ by defining $x^{*}(q)$ to be the $x$ that minimizes (8.2.3). Then applying the chain rule yields $d d_{o b s} / d q=n^{T} J$, where $n$ is the gradient of the distance-toobstacle function at $x^{*}(q)$ (efficiently computed offline by the FMM or Euclidean Distance Transform [EDT] [29], assuming a voxel-based obstacle representation), and $J$ is the Jacobian mapping joint velocities to the linear velocity of the point $x^{*}(q)$.

Linearizing $d_{o b s}(q)$ about a (differentiable) point $q_{0}$ then yields

$$
\begin{equation*}
d_{o b s}(q) \sim d_{o b s}\left(q_{0}\right)+n^{T} J\left(q-q_{0}\right) \tag{8.2.4}
\end{equation*}
$$

where $n^{T} J$ is evaluated at the point $q_{0}$. Note that $d_{o b s}\left(q_{0}\right)$ can also be computed efficiently via the EDT. Substitution into our original cost function yields the approximation

$$
\begin{equation*}
\alpha e^{-\left(d_{o b s}\left(q_{0}\right)+n^{T} J\left(q-q_{0}\right)-d_{0}\right) / \bar{d}} \tag{8.2.5}
\end{equation*}
$$

Remembering that our goal is to obtain a cost function that only depends on one
coordinate, we can perform an affine change of coordinates to $\phi$ coordinates such that

$$
\phi(q)=\left[\begin{array}{c}
n^{T} J  \tag{8.2.6}\\
b_{2} \\
\vdots \\
b_{N}
\end{array}\right]\left(q-q_{0}\right)
$$

where the other basis vectors $b_{2}, \ldots, b_{N}$ are chosen to complete an orthogonal basis also satisfying $\left\|b_{i}\right\|=\left\|n^{T} J\right\|$. This allows us to finally write

$$
\begin{align*}
h\left(q_{0}, q_{1}\right) \sim & \min _{\phi(t)} \int_{0}^{1} 1+\alpha e^{-\left(d_{o b s}\left(q_{0}\right)+\phi_{1}(q)-d_{0}\right) / \bar{d}} d t \\
& \text { subject to } \phi(0)=\phi\left(q_{0}\right), \phi(1)=\phi\left(q_{1}\right) \tag{8.2.7}
\end{align*}
$$

which is a path planning problem with a one-dimensional cost function that can be solved by the method discussed previously.

Since planning with this (or any) heuristic function will require it to be evaluated many times, a concern is whether this it can be computed efficiently enough to serve in this context. Fortunately, we can precompute solutions to the optimization problem above, leaving only some coordinate transformations and a table look-up to perform at run time. This is accomplished by discretizing a range of possible values for $d_{o b s}\left(q_{0}\right)$. Given each such value, we can then compute the two-dimensional value function associated with the one-dimensional cost function above. At run time, we can then look up the correct value function based on $d_{o b s}\left(q_{0}\right)$, change to $\phi$ coordinates, and look up the precomputed value.

### 8.2.2 Results

To demonstrate how the method can be employed in the context of randomized planning, an RRT was implemented to solve the arm planning problem using the Fermat heuristic. Aside from the distance function, the algorithm was basically identical to the single-tree, vanilla RRT described in [52].

The experiments simulated a planar arm with eight joints and no self-collisions. A good set of parameters was found by manual search, ultimately setting $\alpha=1.0, d_{0}=0.5, \bar{d}=$ 0.4. Three different kinds of statistics were collected and averaged over 40 trials: total number of samples drawn, total number of connection attempts to new samples made, and elapsed wall-clock time before finding a solution. Results from two different scenarios are shown in Figure 8.4.

It was observed that the Fermat heuristic consistently resulted in significantly fewer samples drawn than the naive heuristic. The total number of connection attempts tried


Figure 8.4: Results obtained applying Fermat heuristic to arm planning problem with randomized planner. Images on left show obstacles (in black), initial arm position (dotted line), and desired end-effector position (red cross). Bar graphs on right show corresponding statistics, averaged over 40 trials. Samples indicates total number of samples drawn by the RRT, connections shows total number of connection attempts, and elapsed shows wall-clock run time.
followed a very similar pattern, predictably. These observations seem to support the proposition that the Fermat heuristic results in more efficient sampling. Although it is very dependent on the fine details of the implementation, the elapsed wall-clock time is also shown to demonstrate that the Fermat heuristic can be implemented efficiently enough to compete with the extremely simple Euclidean distance function. These results were promising, as much more work could be done to optimize the implementation.

Figure 8.5 shows how the method scales as a function of the dimensionality of the problem (i.e., the number of joints in the arm). Surprisingly, when using the Fermat heuristic, little correlation was evident between the dimensionality and the total number of samples or connections needed to find a solution; 80-dimensional problems were solved easily. The performance of the Euclidean heuristic, on the other hand, scaled poorly with the dimension; problems of dimension greater than 16 could not be solved in the allotted time. Though a definitive explanation for this phenomenon remains elusive, it is speculated that the Fermat heuristic may have performed unexpectedly well because it depends on a linearization that becomes a better approximation for an arm with many joints.


Figure 8.5: Comparison of mean samples and connection attempts to first solution as a function of number of joints in arm (problem dimension) for scenario (A), as pictured in Figure 8.4.

## Chapter 9

## Learning cost structure

As seen in Chapter 8, it is possible to exploit low-dimensional structure to great effect in the solution of high-dimensional motion planning problems. However, manually finding and harnessing this structure can be prove to be a tedious process in general, with success dependent on the designer's ability to cleverly formulate a problem such that lowdimensional structure is evident. We therefore turn to the issue of automatically learning and exploiting low-dimensional structure via a method here referred to as Spectral Learning of Approximate Symmetries for High-dimensional Dynamic Programming (SLASHDP). SLASHDP can be understood as attempting to compress the cost function of the optimal control problem by predicting it from a small number of linear features that best explain the variation in the cost. Leveraging the results of Chapter 4, this compressed representation can be used to compute a compressed value function, from which a feedback plan over the entire space can be extracted efficiently.

### 9.1 Motivation

An informal example will serve to elucidate these ideas. Figure 9.1 depicts a simple arm planning problem in the presence of obstacles. Our goal will be to find a small set of basis motions that cause the most variation in the cost function, which in this case is a function that penalizes proximity of the arm to obstacles. We will then approximate the true cost function by a low-dimensional cost function written as function only of the basis coordinates. Fig. 9.1a shows the best two basis motions learned by SLASHDP for the particular obstacle field depicted. The first motion performs a curling motion that emphasizes motion of the joints near the base. As was desired, this motion seems to cause the greatest variation in the cost function. The second motion again rotates the joints near the base forward, but it simultaneously counter-rotates the joints further along the length


Figure 9.1: Visualization of learned basis vectors and cost as a function of basis coordinates for an arm planning problem. Basis vectors are illustrated in 9.1a. Each line shows a configuration $z_{i} u_{i}$, for $i \in\{1,2\}$ and varying values of $z_{i}$, superimposed on an obstacle map from which the cost function is derived. Sampled cost function in $\left(z_{1}, z_{2}\right)$ coordinates is displayed in 9.1b.
of the arm, causing the arm to fold in on itself.
Letting $z_{1}, z_{2}$ denote the projections of the state onto these basis vectors, we now discretize $z_{1}, z_{2}$ space and numerically estimate the cost as a function of just these coordinates. The result is shown in Figure 9.1b. We can think of this figure as a visualization of obstacle proximity in some generalized configuration space that is a rotation and projection away from our original configuration space. We can see from the figure that if we start at the origin and move forward along the first basis vector, the arm will intersect two obstacles, visible as tall shapes stretched out in the up-down direction. However, these obstacles have limited extent in the $z_{2}$ direction. It is easy to see how we could possibly navigate around them by moving up or down in the $z_{2}$ direction as we move forward in $z_{1}$. This is precisely the kind of behavior that SLASHDP will exhibit when we generate a high-dimensional feedback plan for this low-dimensional cost function.

### 9.2 Related work

A plethora of methods exist to address the general problem of planning in high-dimensional spaces, some of which are described in Chapter 7. However, far fewer have focused specifically on the exploitation of low-dimensional structure to simplify the task. That said, a relatively intuitive decomposition-based approach could be seen as attempting to exploit low-dimensional structure in some way; for instance, the work of [16] generates a plan in a high-dimensional configuration space given a plan in a low-dimensional workspace. Instead of performing a strict decomposition, some recent randomized planning methods have employed a hybrid approach, using low-dimensional auxiliary spaces to aid in creating a sort of sampling bias towards the most important degrees of freedom [84, 33, 14, 89]. Exemplary among these are BiSpace Planning [33] and Task-space RRT [84], which both grow trees in a low-dimensional, user-defined workspace or task space.

Very little work has been devoted to automatically identifying these interesting lowdimensional subspaces, which is the key idea of SLASHDP. One notable exception is [28], which uses PCA to determine directions in which to expand an RRT. SLASHDP, however, applies to the more general optimal motion planning problem.

SLASHDP is comparable in this sense to RRT* [47], which is asymptotically optimal. Unlike RRT*, SLASHDP has a concrete computational complexity bound-i.e., exponential in the sampling resolution and the dimension of the cost function (a term defined loosely for now). If this dimension is low, (e.g., less than 5) and the sampling resolution reasonable, SLASHDP can be used in practice to find globally optimal solutions, independent of the dimension of the configuration space. If the dimension of the cost function is only approximately low, it may still be used as a powerful approximate algorithm, as we will show experimentally.

SLASHDP may be viewed as a way to identify approximate symmetries in the motion planning problem that are exploited to find approximate solutions. Symmetry exploitation in discrete problems has been previously employed in the method of pattern databases (previously described in Sec. 7.3.1). In this case, a pattern database stores precomputed solutions to a set of target patterns representing necessary subgoals. The cost to obtain each target pattern from a given state is a lower bound on the optimal value; therefore, the heuristic can chosen to be the maximum optimal cost over all target patterns. Given some pattern database, the optimal costs associated with additional target patterns can sometimes be computed by exploiting known symmetries - hence obtaining a tighter lower bound. In previous work, these symmetries were both of a discrete nature and known a-priori. SLASHDP, by contrast, finds continuous symmetries automatically, and exploits these outside the context of a pattern database.

Although SLASHDP might be motivated from a variety of perspectives, the one principally adopted here views SLASHDP as compressing the cost function to efficiently obtain a compressed value function. In this sense, it is reminiscent of research in the artificial intelligence and machine learning domains that pursues a similar objective in the context of POMDPs. Belief compression [80] and value-directed compression [69] are among the most well-known of these techniques. Both methods obtain (discrete-state) compressed POMDPs in an automatic way and subsequently plan efficiently in these smaller representations, similar to the way SLASHDP obtains a compressed cost function and derives from it a compressed value function. SLASHDP, however, exploits the additional continuous structure of motion planning problems pre-discretization to obtain stronger results and more efficient algorithms where it is applicable.

### 9.3 Spectral learning of cost structure

SLASHDP applies to the holonomic optimal motion planning problem described in Section 4.3. Recalling that discussion, dynamic programming for this problem involves finding a value function $V(y)$ mapping states to the minimum cost of a path starting at $y$ and ending at the goal. The value function is defined as

$$
\begin{align*}
V(y):=\min _{x(t)} J\{x\}= & \int_{0}^{1}\|\dot{x}\| C(x) d t  \tag{9.3.1}\\
\text { subject to } \quad & x(0)=0 \\
& x(1)=y,
\end{align*}
$$

where $C(x)$ is a cost function assumed to be given as part of the problem specification. Due to the holonomic assumption, the optimal control law is expressed simply as

$$
\begin{equation*}
\dot{x}=-\nabla V(x) . \tag{9.3.2}
\end{equation*}
$$

As discussed in Chapter 4, the infeasibility of direct computation of the value function motivates a solution based on finding a compressed representation of the cost function, from which a corresponding compressed representation of the value function can be obtained efficiently. This can be achieved in practice by applying Corollary 4.3.2, reprinted here for convenience:

Corollary 9.3.1 (Value function compression). Suppose that $C(x)$ is the cost associated with a variational problem of the form (9.3.1) and $V(y)$ is the associated value function. If $W$ is such that $C(x)=C\left(W W^{T} x\right), \forall x$; and $\nu$ is any vector such that $W^{T} \nu=0$ and
$\|\nu\|=1$, then

$$
\begin{equation*}
V(y)=V\left(W W^{T} y+\left\|\left(I-W W^{T}\right) y\right\| \nu\right), \forall y \tag{9.3.3}
\end{equation*}
$$

This result means that if a suitable matrix $W \in \mathbb{R}^{N \times d}$ (with $d<N$ ) can be found, we can precompute a $d+1$-dimensional value function using the Fast Marching Method (FMM) and subsequently compute any value in the $N$-dimensional space by a simple lookup operation.

### 9.3.1 Estimating the cost basis

For such a scheme to prove useful in practice, we must have that $d \ll N$, since computing the value function will otherwise be infeasible. In the common case that no such $W$ exists, we can still apply this general methodology; however, instead of exactly compressing $C$ via the relation $C(x)=C\left(W W^{T} x\right)$, we will apply a lossy compression scheme for which this relation will only be approximately true.

Assuming that $C$ is at least once-differentiable, then $C(x)=C\left(W W^{T} x\right)$ implies that

$$
\begin{align*}
\nabla C(x) & =W W^{T} \nabla C\left(W W^{T} x\right) \\
\left(I-W W^{T}\right) \nabla C(x) & =\left(I-W W^{T}\right) W W^{T} \nabla C\left(W W^{T} x\right) \\
\left(I-W W^{T}\right) \nabla C(x) & =0 . \tag{9.3.4}
\end{align*}
$$

In the general case that the relation $C(x)=C\left(W W^{T}\right)$ does not hold exactly, a natural idea is to write

$$
\begin{equation*}
\nabla C(x)=W W^{T} \nabla C(x)+\left(I-W W^{T}\right) \nabla C(x) \tag{9.3.5}
\end{equation*}
$$

and choose $W$ such that, in expectation, $\left\|\left(I-W W^{T}\right) \nabla C(x)\right\|^{2}$ is as small as possible; or equivalently, so that $\left\|W W^{T} \nabla C(x)\right\|^{2}$ is as large as possible, while limiting the number of columns $d$ of $W$ :

$$
\begin{align*}
W:= & \underset{ }{W^{T} W=I}  \tag{9.3.6}\\
& \operatorname{rank} W=d
\end{align*}
$$

This may be approached using standard optimization techniques. Applying a number of
transformations, and defining $w_{i}$ to be the $i$ th column of $W$, we obtain

$$
\begin{align*}
\mathbb{E}_{x}\left\|W W^{T} \nabla C(x)\right\|^{2} & =\mathbb{E}_{x}\left\|\nabla C(x)^{T} W W^{T} \nabla C(x)\right\|^{2} \\
& =\sum_{i=1}^{d} \mathbb{E}_{x} \nabla C(x)^{T} w_{i} w_{i}^{T} \nabla C(x)  \tag{9.3.7}\\
& =\sum_{i=1}^{d} w_{i}^{T}\left(\mathbb{E}_{x} \nabla C(x) \nabla C(x)^{T}\right) w_{i} . \tag{9.3.8}
\end{align*}
$$

Adding Lagrange multipliers $\lambda_{i}$ to enforce the constraints $\left\|w_{i}\right\|^{2}=1$ and differentiating with respect to each $w_{i}$ yields

$$
\begin{equation*}
\mathbb{E}_{x} \nabla C(x) \nabla C(x)^{T} w_{i}=\lambda_{i} w_{i}, \forall i \in\{1, \ldots, d\} \tag{9.3.9}
\end{equation*}
$$

Back-substitution of this expression into the objective shows that in order to maximize the objective, we should choose the columns of $W$ to be the eigenvectors associated with the largest $d$ eigenvalues of $E_{x} \nabla C(x) \nabla C(x)^{T}$.

We refer to $M:=\mathbb{E}_{x} \nabla C(x) \nabla C(x)^{T}$ as the matrix of second moments of the cost gradient. Of course, since $M$ is not known a-priori, it must be estimated by sampling. Although sampling in high-dimensions is generally difficult, intuition suggests that if the spectrum of $M$ is sufficiently tapered, it should not be difficult to estimate the eigenvectors corresponding to its largest eigenvalues. As an extreme example of this, we might consider the case where $M$ has only one nonzero eigenvalue: drawing any single sample of the cost gradient will perfectly recover the eigenvector corresponding to the nonzero eigenvalue, but we will never draw a sample corresponding to any other eigenvalue. It is therefore logical to expect that eigenvectors corresponding to large eigenvalues are easy to estimate, while those corresponding to small eigenvalues are difficult to estimate; fortunately, we have no need to estimate the latter.

### 9.3.2 Compressing the cost function

Given a basis $W$, we now wish to estimate the compressed cost function. That is, writing $z=W^{T} x$, we wish to find $C_{z}(z) \approx C(W z)$, where $C_{z}: \mathbb{R}^{d} \rightarrow \mathbb{R}$ is the compressed cost function. We then proceed by discretizing $z$ coordinates on a regular grid, which will eventually enable us to compute the value function using a fast, grid-based version of the FMM. For each of these points $z[i]$, we must obtain an estimate of $C_{z}(z[i])$. This is very simple in the ideal case; we simply take $C_{z}(z)=C(W z[i])$.

In the likely event that $\exists x \mid C(x) \neq C\left(W W^{T} x\right)$, different states that project to the same $z$ coordinates can have different cost values-although we intentionally chose our
basis to minimize the expected difference, in some sense. The obvious solution to this problem is to sample the costs of states projecting to the same coordinates $z$, and to set $C_{z}(z)$ to the mean, max, or min of all such values. Alternatively, supposing for now that we are only interested in finding the optimal path to the goal for a particular query state, we know by Theorem 4.4.1 that the optimal path with respect to the compressed cost function will be contained in the $(d+1)$-dimensional subspace spanned by the columns of $W$ and the query state. Therefore, we can just as well restrict our sampling efforts to this subspace, thus obtaining a far simpler sampling problem. In the event that we subsequently use the value function thus obtained to find paths for query states not in this subspace, this strategy may not be optimal, as the resulting paths will not lie in this subspace; however, this was not found to be a significant issue experimentally, provided that the query configurations were not too far from that assumed in the computation of the compressed cost.

### 9.3.3 Planning a path

After learning a $d$-dimensional basis in which the cost is approximately low-dimensional and discretely sampling the cost in these coordinates, we can use the FMM to compute a $d+1$ dimensional value function from which we can derive a high-dimensional value function by invoking Theorem 4.3.1. This value function is a symmetric approximation of the problem's true value function, which cannot be computed tractably. Since we have learned a basis such that most of the variation of the cost is captured in this basis, we anticipate that the resulting symmetric approximate value function will be a good approximation to the true one.

Given the high-dimensional, symmetric, approximate value function obtained in this way, we can efficiently compute the corresponding optimal paths by integrating (9.3.2). In practice, however, these paths are computed by the equivalent method of integrating (9.3.2) with respect to the $d+1$-dimensional value function and lifting these paths to the original high-dimensional space by means of Theorem 4.4.1.

### 9.3.4 A generalized shortcut heuristic

The method described so far is useful for problems where the original coordinates are a rotation away from a new set of coordinates in which the cost is approximately lowdimensional. However, for some problems, this may only be true locally. For such problems, the following approach is suggested: given an initial path, randomly choose two points on this path, and find a new path between them via SLASHDP. If the cost of the found path is lower than the original segment, replace the old segment with the new one; otherwise,
keep the original segment. Repeat as many times as desired.
For each such iteration, we can focus our learning efforts on a smaller search volume surrounding the path endpoints. It is our hope that the cost function in this smaller volume can be adequately represented as a function of a small number of basis projections. If this does hold, we expect to be able to find a nearly optimal path interpolating these endpoints.

This method can be seen as a generalization of the straight-line shortcut heuristic [36] widely used to post-process feasible motion plans generated by sampling-based methods. The straight-line shortcut heuristic might be described in this way: connect two points on a path with a straight line; if the cost of this path is lower than the original segment, replace it with a straight line. If our cost function is constant in the free space, then a straight line is the optimal path. The straight-line shortcut method can then be seen as a special case that arises when a zero-dimensional (constant) approximation of the cost function is used. SLASHDP, however, can employ higher-dimensional cost approximations, resulting in higher-dimensional, nonlinear shortcuts-using a d-dimensional cost approximation, we can search over a space of paths spanning a $d+1$-dimensional space.

### 9.3.5 Computational considerations

We note that there are three major components to the computational cost of SLASHDP: sampling cost gradients, sampling the cost function in the reduced space, and performing dynamic programming. For a $d$-dimensional compressed cost using $k$ samples per dimension, sampling the cost function takes time $O\left(k^{d}\right)$, and dynamic programming takes time $O\left(k^{d+1}(d+1) \log k\right)$. If we make the reasonable assumption that we will not need to sample more gradients than there are samples in our grid, then sampling the gradients also takes time $O\left(k^{d}\right)$. In practice, any one of these factors may dominate the computation time, depending on the relative expense of computing the cost function and cost gradients vs. dynamic programming. However, we note that the sampling steps are embarrassingly parallel; therefore, given sufficient processors, we expect the bottleneck to be in the low-dimensional dynamic programming step. As it is possible to implement DP very efficiently, SLASHDP would probably benefit dramatically from extreme parallelization of the sampling component (say, for instance, using a GPU).

### 9.4 Results

For the purpose of evaluation, SLASHDP was applied two problems: planning for a robot arm with many degrees of freedom and planning for a deformable robot. Both are challenging high-dimensional motion planning problems.

### 9.4.1 Planning for a robot arm

For this experiment, simulated planar arms with varying numbers of joints were simulated. SLASHDP was applied to optimize a maximum-clearance-type objective that exponentially penalizes proximity to obstacles. Specifically, paths $q(t)$ in joint angle space were sought to optimize the cost functional

$$
J[q]=\int_{0}^{1}\left(1+e^{-\left(d_{o b s}(q(t))-d_{0}\right) / \bar{d}}\right)\|\dot{q}\| d t
$$

where $d_{\text {obs }}(q)$ is the nearest distance to an obstacle when the arm is in configuration $q$, and $d_{0}$ and $\bar{d}$ are fixed parameters.

The performance of SLASHDP on this scenario was compared to two others: a bidirectional RRT [52], and a naive method based on a low-dimensional projection. The naive method calculates a two-dimensional feedback plan for the end-effector that optimizes a similar objective to the one above, but only considering the end-effector coordinates independently of the rest of the arm. It then lifts this plan to joint space by mapping the feedback controller's desired end-effector velocity to joint velocities via the pseudoinverse of the Jacobian. The RRT consists of a standard bidirectional RRT using linear interpolation as a local planning method and Euclidean distance as a distance metric.

Qualitative results for this experiment are shown in Figure 9.2. Each subfigure shows the set of points swept out by a 36 -dimensional arm as it travels along the solution trajectory obtained with each method. The naive method produces a smooth solution, but it collides with an obstacle, since the geometry of the arm was not taken into consideration during construction of the low-dimensional path. The RRT produces a collision-free path, but it is very complicated. The arm sweeps out a large area as it travels along this trajectory, and it comes very close to collision many times. By contrast, the SLASHDP solution is smooth and collision-free while maintaining a large amount of clearance to all obstacles.

These qualitative observations are supported by the quantitative results in Fig. 9.3. This experiment compared the three methods as the dimensionality of the arm scaled up to 144 joints. The cost of an initial solution found via each method is shown in Figure 9.3a (note log scale). As expected, the naive solution always had a higher cost than the other methods, since it consistently collided with obstacles. The RRT produced a lower cost due to its lack of collision, but the length of these solutions coupled with their occasional proximity to obstacles still caused them to have a relatively high cost. SLASHDP produced solutions that were consistently on the order of 100 to 200 times less costly than the RRT for very high-dimensional problems. Furthermore, the cost of the SLASHDP solutions actually decreased monotonically as the the dimensionality of the problem increased while the other methods exhibited either a stable or generally increasing trend. This implies


Figure 9.2: Subjective comparison of different methods applied to a 36 -dimensional arm planning problem. Black shapes represent obstacles. Colored/shaded areas represent set of points visited by each method's solution path. Naive method produces a smooth solution, but it collides with an obstacle. RRT produces a feasible solution, but it is complicated and passes near many obstacles. SLASHDP generates a high-quality solution that is smooth and maintains a large amount of clearance to obstacles.
that SLASHDP, instead of being confused by the extra dimensions, was able to exploit the extra degrees of freedom to further decrease the cost.

Each of these solutions was then post-processed using a local smoothing method (in particular, an elastic band [71]), not stopping until a local optimum was found. All of the methods benefited from this step, though to varying degrees. Post-processing decreased the RRT and naive costs by a factor of 5-20-still not enough to surpass the quality of the initial plans generated by SLASHDP. It decreased the SLASHDP cost by a factor of 1.5 at most, obtained for the arm of the lowest dimensionality and decreasing to just 1.1 for the 144 -dimensional arm. The opposite trend was observed for the RRT, where the higher-dimensional cases were the farthest from optimality.

The total time spent processing the path with each method, including post-processing, is given in Figure 9.3c. The performance of SLASHDP was characterized by a large fixed cost of performing DP on a $100 \times 100 \times 10 x 100$ lattice and the cost of computing the sampled cost values and gradients, which scales linearly in the dimension, and which dominated the cost for large dimensions. Finding an initial plan with SLASHDP was therefore quite expensive compared to the other methods for small dimensions, but less so for large dimensions. Looking at the total time including post-processing, however, yields a very different picture. Though the RRT is still faster for small dimensions, SLASHDP is roughly twice as fast for dimensions greater than 36. This is due to the fact that the elastic band spends an inordinate amount of time optimizing the RRT solution-which is very far from even local optimality-while it terminates very quickly for the SLASHDP solution, which is probably already close to optimal.

### 9.4.2 Planning for a deformable robot

SLASHDP was also evaluated on a challenging high-dimensional deformable robot problem. The robot is assumed to live in a two-dimensional space, where it can translate freely and deform in a way that is controlled by a high-dimensional set of configuration parameters $q$. Specifically, the robot boundary is given by a function $r(\theta, q)$ that gives the distance of the boundary from a reference point at angle $\theta$, when the robot is in configuration $q$. This function is given as a Fourier series expansion:

$$
r(\theta, q)=r_{0}+\sum_{k=1}^{N} s_{2 k} q_{2 k} \cos k \theta+s_{2 k+1} q_{2 k+1} \sin k \theta
$$

where the $s_{k}$ are constant scale parameters.


Figure 9.3: Results of experiments comparing different methods applied to very highdimensional arm planning problem pictured in Figure 9.2, as a function of problem dimension. 9.3a shows cost of solution produced by each method. 9.3b shows cost of each solution after post-processing with elastic band. 9.3c shows total time to find each solution, including post-processing. Note log scale of $y$ axes. Paths produced by SLASHDP, even without post-processing, are much superior to those produced by the other methods with post-processing included. SLASHDP solutions also decrease in cost as dimension is increased. The other methods are faster for low-dimensional problems, but SLASHDP easily outperforms RRT + post-processing for very high-dimensional problems.

The position $x(\theta, q)$ of a point on the robot boundary is then given as

$$
x(\theta, q)=\binom{q_{2 N+2}}{q_{2 N+3}}+r(\theta, q)\binom{\cos \theta}{\sin \theta} .
$$

A cost function $C(x)$ is defined to penalize the proximity of any point on the boundary to an obstacle:

$$
C(x)=1+e^{-\left(d_{o b s}(x)-d_{0}\right) / \bar{d}}
$$

For computational purposes, $K$ evenly-spaced $\theta_{k} \in[0,2 \pi)$ were chosen along the boundary of the robot. Let $x_{k}(q)=r\left(\theta_{k}, q\right)$ denote the position of the $k$ th sample. Given this notation, we can write the cost functional as

$$
J[q]=\int_{0}^{1}\left(\sum_{k=1}^{K} C\left(x_{k}(q)\right)\right)\|\dot{q}\| d t
$$

SLASHDP was then applied to the problem in this form.
Figure 9.4 shows the three-dimensional basis learned for a specific instance of this problem consisting of a maze-like environment. As would be expected, the first two basis vectors encode just the position of the robot with no deformation, while the last encodes a useful-looking deformation with no translation component. To understand why this is so, it is useful to view the nature of the maze depicted in Figure 9.5, which consists of corridors running in the "northeast-southwest" and "northwest-southeast" directions. It is clear that the third basis vector encodes a deformation that the robot to travel through either type of corridor with low cost. This is confirmed in the generated paths, in which the robot deforms between these shapes, transitioning through a circular phase at the corners.

To demonstrate the usefulness of SLASHDP to generate a feedback plan, a path was initially generated using a query pair of configurations, depicted in Figure 9.5a. The generated value function was then used to quickly produce solutions for a variety of final configurations, two of which are shown in Figure 9.5b. Although the initial plan was very costly to create (on the order of tens of minutes), subsequent plans were generated very quickly (on the order of milliseconds) thanks to the availability of the value function.

Quantitative results are given in Figure 9.6. This series of experiments tested the ability of SLASHDP both to find an initial path and to post-process a path by the generalized shortcuts heuristic presented in Section 9.3.4. For a baseline, a plan for the center of mass was computed that merely translated the robot through the maze without deformation. This path was then post-processed with different methods in order to lower its cost. Neither an elastic band nor the naive shortcut heuristic described in Section 9.3.4 was able to significantly decrease the cost of this initial plan. Post-processing with SLASHDP

(a) Random configurations

(b) Learned basis

Figure 9.4: Visualization of random configurations vs. learned basis for deformable robot planning problem. Randomly sampled configurations are shown for reference in 9.4a. Learned basis vectors are displayed in 9.4b. Each row represents one of three learned basis vectors. Shape in each column represents the shape with coordinates $t u_{i}$, where $u_{i}$ is the $i$ th learned basis vector. Central arrows show magnitude and magnitude of translation. First two basis vectors encode position of robot with no deformation. Third basis vector encodes a useful deformation of robot with no translation.


Figure 9.5: Visualization of results of experiment in planning for deformable robot. Figure 9.5a shows plan obtained by planning for a given query configuration. Fig. 9.5b shows additional plans obtained very quickly from the value function computed for 9.5a.


Figure 9.6: Results of planning for a deformable robot. Left group of bars represent cost of plans obtained by using different methods to post-process an initial plan generated for the robot's centroid with no deformation (bar labeled COM plan). Right group of bars represent cost of plans obtained by post-processing initial plan generated by SLASHDP (bar labeled SLASHDP). +smoothing indicates post-processing with an elastic band, +naive shortcuts post-processing with a straight-line shortcut heuristic, and + SLASHDP shortcuts post-processing by using SLASHDP to generate local shortcuts.
shortcuts, however, reduced the cost of this plan by nearly $40 \%$.
Similar results were obtained starting with an initial plan generated by SLASHDP. The initial path was a modest $10 \%$ better than the path obtained via translation-only planning. This path was not significantly improved by post-processing with an elastic band or naive shortcuts. SLASHDP shortcuts, however, again yielded a large decrease in the objective value, settling to a value very close to that obtained by smoothing the purely translational path with SLASHDP.

## Chapter 10

## Learning dimensional descent

SLASHDP, as described at length in Chapter 9, is based on the intuition of computing a compressed value function from a compressed cost function. This approach is particularly useful as a multiple-query or feedback method, in the sense that a single lengthy planning phase generates a value function from which an (approximately) optimal control law can be extracted very efficiently. For the single-query variant of the motion planning problem, on the other hand, we can develop a more path-centric approach to obtain much improved results. This method generates a succession of monotonically improving solutions by optimization over a series of low-dimensional submanifolds of the ambient space. From an optimization perspective, the procedure can be thought of as Hilbert-space coordinate descent in learned coordinates, leading to the choice of the moniker Learning Dimensional Descent (LDD) to describe the algorithm.

### 10.1 Motivation

LDD may be motivated in various ways, not the least of which is a desire to somehow factorize the planning problem into more easily-solved subproblems. A classic example of this kind of decomposition is discussed in [16], which proposes generation of a plan for a mobile manipulator's high-dimensional configuration space given a plan in the robot's lowdimensional workspace. Further examples come from legged locomotion, where a number of approaches are based on some variation of the following program: first plan a trajectory for the robot's center of mass, subsequently find a high-dimensional plan in the space of footstep locations to follow this path, and finally generate a plan in the high-dimensional configuration space of the robot that is compatible with the footstep plan [51, 106, 42, 78].

The success of such methods generally hinges on finding good decompositions that allow
the later, higher-dimensional plans to be generated using simpler methods that are likelybut not guaranteed-to succeed if initialized well. Such decompositions are built manually based on the prior knowledge of a designer. From the perspective of machine learning, or the practitioner tasked with building such systems, this degree of manual intervention is very undesirable. An ideal planning algorithm would perform these decompositions automatically based on an examination of the problem data, adapting itself to different scenarios with minimal human intervention.

Some previous work along these lines comes from the literature in deterministic search, where abstraction hierarchies have been employed to accelerate search in different ways. Hierarchical A* [43] automatically builds hierarchies of abstractions of discrete planning problems. The heuristic of a state at a given level of abstraction is computed as the optimal value of the state in the next-higher level of abstraction. A similar method [20] also builds hierarchies of abstractions, but employs them differently; instead of using the hierarchy to compute heuristics, the hierarchy is used to plan in a coarse-to-fine manner, first generating a plan at the most abstract level, and subsequently generating more detailed plans at more concrete levels. Although these methods derive from a similar philosophy as that which motivates LDD, they differ fundamentally in that LDD is able to leverage the continuous structure of motion planning problems to find more suitable abstractions.

Like SLASHDP, LDD finds a basis that best explains the variation of the cost function. However, instead of using this basis explicitly to compress the cost function, LDD regards the (ordered) basis as a way of discovering interesting decompositions that enables and automates the idea of planning first for the most important degrees of freedom and only later planning for the less important degrees of freedom, once the most critical aspects of the path have been determined. The learning part of LDD hence consists of finding the basis, while the dimensional descent part uses the basis to sequentially plan through the dimensions of the learned basis, in order of descending importance.

By borrowing the basis learned bySLASHDP for this purpose, LDD essentially generalizes SLASHDP, and hence inherits both the sufficient conditions for optimality and the practical ability to solve very high-dimensional planning problems. However, it is notable that the method of variational dynamic programming (VDP) previously employed an idea very similar to dimensional descent, albeit using a changing, completely random basis in lieu of a learned basis [8]. As a result, VDP does not admit the same theoretical optimality guarantee that LDD possesses when applied to problems with low-dimensional cost functions. Furthermore, the experiments later in this chapter show that the choice of basis is crucial in practice, since a poor basis may result in early convergence to a poor local minimum. Lastly, VDP is not guaranteed to produce monotonic improvement in the continuum limit, in contrast to LDD, because it does not account for important metric
transformations between iterations, as described in Section 10.2.1.

### 10.2 Method

LDD will be developed in such a way as to make it a generalization of SLASHDP, up to some minor details. Making this precise, SLASHDP can be thought of as optimizing the following objective, where $J\{x\}$ is the holonomic-optimal-control cost functional in Eq. (9.3.1),

$$
\begin{equation*}
x^{*}(t)=\underset{a(t), s(t)}{\arg \min } J\{W a(t)+y s(t)\}, \tag{10.2.1}
\end{equation*}
$$

and, as before, $y$ is assumed to be the goal, the other endpoint is assumed to be the origin, and $W$ is the basis learned by SLASHDP, as described in Section 9.3.1.

Generalizing this to implement the previously described program of dimensional descent is conceptually very simple. Given the learned basis $W$, instead of searching once over the set of paths described by Eq. 4.4.1, we find a sequence of paths $\bar{x}^{k}$ defined by

$$
\begin{equation*}
\bar{x}^{k+1}(t)=\underset{a^{k}(t), s(t)}{\arg \min } J\left\{W^{k} a^{k}(t)+\bar{x}^{k}(s(t))\right\}, \tag{10.2.2}
\end{equation*}
$$

where each $W^{k}$ is a matrix comprised of a strict subset of the columns of $W$. Assuming that $W^{k}$ is an $N \times d$ matrix, each step involves solving a $d+1$-dimensional DP problem. The algorithm begins by setting $W^{0}$ to the first $d$ columns of $W$, and by setting $\bar{x}^{0}(s)=y s$-in that sense, generalizing SLASHDP. Subsequent steps set $W^{k}$ equal to the next $d$ columns of $W$, and so on. After all the columns of $W$ are exhausted in this way (or we reach a predefined limit), we again set $W^{k}$ equal to the first $d$ columns of $W$. LDD proceeds in this way until convergence or some other termination condition is met.

Choosing the $W^{k}$ in this way ensures that we choose directions to optimize in order of their relative importance. This intuitively minimizes the chance that LDD will get stuck in some poor local minimum early on. Furthermore, by Theorem 4.4.1, it ensures that LDD will terminate with a globally optimal solution in one step provided that the conditions of that theorem are met and we choose a large enough $d$. Note also that whether or not a global optimum is achieved, LDD has the important property of monotonic convergence towards a local optimum. This is a simple consequence of the fact that the solution in one iteration is contained within the feasible set of the next iteration.

The geometry of the LDD optimization is illustrated in Fig. 10.1. The set of paths considered by LDD at each iteration is the (generally non-linear) submanifold of $\mathbb{R}^{N}$ obtained by sweeping $\bar{x}^{k}$ in the directions spanned by the column space of $W^{k}$. When $W^{k}$ is


Figure 10.1: Illustration of one step of dimensional descent algorithm. Given initial path $\bar{x}^{k}$, the next path $\bar{x}^{i+1}$ is the optimal path constrained to lie on a manifold obtained by sweeping $\bar{x}^{k}$ along $W^{k}$.
a single column, this submanifold is simple ruled surface, as illustrated.

### 10.2.1 Technical details

The rest of this section is devoted to the subtle technical issue of solving the LDD optimization (10.2.2) via dynamic programming. A naive attempt to solve (10.2.2) might involve sampling $a^{k}$ and $s$ on a finite lattice, evaluating the cost function at these points as $C\left(a^{k}, s\right)=C\left(W^{k} a^{k}+\bar{x}^{k}(s)\right)$, and applying the Euclidean FMM to the resulting problem. However, this will not yield the correct result, due to the aforementioned non-Euclidean geometry of the problem (Fig. 10.1). If we wish to optimize paths under the original objective, which specifies the Euclidean metric in $\mathbb{R}^{N}$, we must therefore take into account how this metric transforms under a change to the coordinates $\left(a^{k}, s\right)$ of the submanifold that is the set of feasible paths.

To derive the correct procedure, we must therefore substitute the expression for the path in manifold coordinates $x(t)=W^{k} a^{k}(t)+\bar{x}^{k}(s(t))$ into the Euclidean metric to derive
its expression in terms of manifold coordinates. Doing so yields

$$
\begin{align*}
\dot{x}^{T} \dot{x}= & \dot{a}^{k}(t)^{T} \dot{a}^{k}(t)+\dot{a}^{k}(t)^{T}\left(W^{k}\right)^{T} \frac{d}{d t}\left[\bar{x}^{k}(s(t))\right] \\
& +\frac{d}{d t}\left[\bar{x}^{k}(s(t))\right]^{T} \frac{d}{d t}\left[\bar{x}^{k}(s(t))\right] \tag{10.2.3}
\end{align*}
$$

To render optimization with respect to this metric a problem amenable to optimization by standard methods, which generally assume the Euclidean metric, note that we can make some helpful simplifying assumptions. First, note that we can equivalently parameterize the manifold by replacing $\bar{x}^{k}$ with $\bar{x}^{k}-W^{k}\left(W^{k}\right)^{T} \bar{x}^{k}$, yielding

$$
\begin{equation*}
x(t)=W^{k} a^{k}(t)+\left(I-W^{k}\left(W^{k}\right)^{T}\right) \bar{x}^{k}(s(t)) \tag{10.2.4}
\end{equation*}
$$

(Note that this choice of manifold coordinates is also illustrated in Fig. 10.1.) Substitution of this expression into the metric causes the cross-term to cancel, resulting in the following diagonal metric, after simplification. Let $P=I-W^{k}\left(W^{k}\right)^{T}$. Then

$$
\begin{equation*}
\dot{x}^{T} \dot{x}=\dot{a}^{k}(t)^{T} \dot{a}^{k}(t)+\dot{s}^{2} \frac{d \bar{x}^{k^{T}}}{d s} P^{T} P \frac{d \bar{x}^{k}}{d s} \tag{10.2.5}
\end{equation*}
$$

At this point, we wish to make the coefficient of $\dot{s}^{2}$ equal to one. Note that we can achieve this simply by assuming that $\bar{x}^{k}(s)$ has the arc-length parameterization after projection onto the subspace orthogonal to $W^{k}$. We must therefore take care to make this assumption in all expressions involving $\bar{x}_{i}$-specifically when applying (10.2.4) and when calculating the cost function in terms of manifold coordinates, $a^{k}$ and $s$. Doing so ensures that we can safely apply an efficient Euclidean FMM solver to optimize the correct objective.

### 10.3 Simulation results

LDD was evaluated in simulation on two high-dimensional motion planning problemsfirst, the problem of planning for a deformable robot; and second, the problem of planning for a planar arm in the presence of obstacles. For all of the results described below, the $d$ parameter was set to one.

### 10.3.1 Deformable robot planning

A robot was simulated with the ability to translate freely in the plane as well as deform its shape. The deformation is controlled by a set of 16 parameters corresponding to Fourier series coefficients of the radius of the boundary of the robot, as a function of angle. To make the problem more challenging, a random rotation was applied to these coordinates,
randomly mixing together all the degrees of freedom. LDD was applied to the problem in these new coordinates.

Results are shown in Fig. 10.2. Dimensional descent in the original, randomized coordinates, was compared to LDD in the learned coordinates; that is, LDD was applied to the problem in random coordinates, which then transformed the problem into learned coordinates. Dimensional descent in the random coordinates produced a very poor solution in which the robot did not correctly deform or translate to fit the maze, whereas LDD correctly translated the robot through the maze, while deforming the shape of the robot to fit the corridors of the maze.

Convergence properties of LDD versus dimensional descent (DD) in the random coordinates are shown as insets in Fig. 10.2. DD in random coordinates yielded a significant improvement in cost in the first iteration, but no noticeable improvement in subsequent iterations, indicating that DD quickly fell into a poor local minimum. LDD also made the bulk of its progress in its first iteration, but its solution after the first iteration was several orders of magnitude less costly than that of DD. After the first iteration, it continued to make steady improvement, especially up until the 10th iteration. After this iteration, LDD seemed to slowly approach a locally optimal solution.

### 10.3.2 Arm planning

LDD was additionally applied to the problem of planning a collision-free trajectory for an 11-DOF planar arm with no self-collisions. The intent was to thoroughly study the ability of LDD to find high-quality solutions in very cluttered environments, especially in relation to other commonly used approaches.

Figures 10.3 and 10.4a depict solutions obtained with LDD in highly constrained environments, the likes of which are usually very challenging for typical motion planning algorithms. Fig. 10.4a depicts the scenario used for quantitative evaluation. In this evaluation, collision-free paths were found between given initial and final configurations, while varying the number of obstacles in the environment. The label on each obstacle indicates the trial in which that obstacle was first introduced; each obstacle was then kept in the environment for subsequent trials, making later trials more challenging than earlier ones.

Quantitative results are given in Fig. 10.4b. LDD was compared to several other methods, including A* with different heuristics and RRT-based planners. Fig. 10.4b shows the cost of the solution obtained with each method for each trial. RRT is a standard bidirectional RRT, provided as a baseline for comparison, though it does not perform any optimization. All other methods attempt some optimization. S-RRT is a bidirectional RRT that has been post-processed with an elastic band planner to improve the quality of its solution, as is common practice. S-TSRRT is a bidirectional variant of Task-Space


Figure 10.2: Results of experiment in 18-dimensional shape planning (see text for details). Consecutive shapes shown as overlapping, translucent objects. Insets show log cost as a function of iteration number. Although both methods converge to some solution, LDD converges to a solution several orders of magnitude better than DD.

RRT [84], also post-processed with an elastic band. DD refers to dimensional descent in the original coordinates. A*Proj refers to A* using the heuristic of the distance of the end-effector to the goal, while A*Full is A* using the Euclidean distance heuristic in the configuration space.

Both A* variants and TS-RRT were only able to find solutions for the trials with less than seven obstacles in a reasonable amount of time - these methods ran for more than eight hours on a 3 GHz Intel Xeon test machine without finding a solution. By its nature, DD always found some solution, but it was not always feasible, as evidenced by the fact that the cost of its solutions far exceeded that of the baseline RRT in many trials. The standard bidirectional RRT (pre- and post-smoothing) and LDD were therefore the only methods consistently able to find solutions for the most difficult problems.

In terms of solution quality, LDD outperformed every other method in every trial, usually by a large margin. The performance gap between LDD and S-RRT was as great as a factor of five. For the most difficult trials, S-RRT still managed to find a solution of roughly half the cost of S-RRT. In order to find any solutions with A* in a reasonable amount of time, a high heuristic weighting factor had to be applied, which generally caused the solutions to be very suboptimal.

### 10.4 Experimental results

LDD was evaluated experimentally via application to the problem of arm planning for the Willow Garage PR2 robot. The PR2 is designed primarily for mobile manipulation tasks, and is equipped with a variety of sensors and two arms, each of which has seven DOF in addition to a one-DOF gripper. Experiments consisted of planning for the seven-DOF arm in two scenarios featuring cluttered environments: one in which the gripper was placed in a small window and made to plan a path to another window, and another in which the arm was made to move between two highly-constrained configurations in the midst of pole-like obstacles. LDD was compared to SBL [81] (a bidirectional sampling-based planner with lazy collision checking), using the open-source OMPL [1] implementation of the latter. Since SBL itself attempts no optimization, like most sampling-based planners, the output of SBL was smoothed using a standard trajectory filter package, which employs a combination of cubic splines and shortcuts.

The objective was mainly to compare the consistency and quality of the solutions obtained with both methods. Measured arc length of the seven-dimensional joint trajectories was used as a primary metric in order to compare the quality of the solutions. In the limit of infinitesimal $d_{s}$ and $d_{0}$ set appropriately, LDD can be made to optimize arc length, roughly speaking; these parameters were therefore set aggressively in order to obtain as


Figure 10.3: Sample solution trajectory in highly constrained environment


Figure 10.4: Fig. 10.4a: visualization of results applying LDD to an arm planning problem. End configurations represented as green and blue solid lines, intermediate configurations by lighter lines. Fig. 10.4b: comparison of several methods applied to the problem in 10.4a. Abscissa shows number of obstacles, and ordinate shows cost of found solutions (note log scale). Obstacles were added in order shown by 10.4a (obstacle 13 outside area shown). See Sec. 10.3.2 for details.


Figure 10.5: Setup of windows experiment. Scale of windows and approximate crosssections of end-effector at start and goal locations are shown.
close an approximation to arc length minimization as possible, although the relatively crude collision modeling employed and numerical issues that arise in this limit were limiting factors in this respect. It is also worth noting that both methods are randomized, as LDD relies on a sampling-based method to estimate the $W$ matrix, and this was re-learned from a blank slate for each trial.

Details of the windows experimental setup are shown in Fig. 10.5. The gripper is roughly 11 cm in width, while both the start and goal configurations were inside windows with a smallest measured side length of 19 cm . For all experiments, the robot's own laser scanner was used to provide an obstacle voxel grid with a resolution of 2 cm .

Some qualitative results of the windows experiment are shown in Fig. 10.6. Although differences between the results of the different methods were sometimes not obviously visible in real-time, analysis of videos taken during the experiment revealed several cases where the smoothed SBL trajectory was inefficient in obvious ways. In one case, the arm was observed to move to first back away to the left before proceeding to the goal on the right. In another case, the wrist needlessly rotated the gripper parallel to the ground,


Figure 10.6: Video stills showing extraneous motions remaining in sampling-based planner's trajectory post-smoothing.
despite the fact that the gripper was constrained to be perpendicular to the ground in the start and goal states. By contrast, trajectories generated by LDD could not be improved in an obvious way. Furthermore, the LDD trajectories were basically indistinguishable between trials, giving confidence that the $W$ matrix had been learned well, with little variance.

Qualitative results for the poles experiment are depicted in Fig. 10.7. The observed differences here were often more dramatic and very obvious in real-time. In particular, the smoothed SBL trajectory occasionally took the arm on a circuitous route around and over the pole on the left. The LDD trajectories by comparison were again always very direct and again basically indistinguishable across trials.

Quantitative analysis of the experimental data corroborated these observations, as shown in Fig. 10.8 (see caption for detailed interpretation). Visualizing the joint trajectories bore out the observation that the LDD trajectories were usually nearly identical across trials. Smoothing the LDD trajectories also provided fairly little benefit in terms of decreasing arc length (vertical axis), indicating that these solutions were probably nearly optimal to begin with-this is again despite the fact that arc length was not minimized


Figure 10.7: Video stills from poles experiment comparing worst-case performance of LDD and SBL (both with smoothing). End-effector is highlighted in intermediate frames. SBL path takes a long detour behind and over the pole on the left, whereas LDD takes a much more direct path.
exactly due to a number of factors. As expected, the SBL trajectories were chaotic, inconsistent, and long across trials before smoothing. Smoothing these sometimes yielded dramatic decreases in arc length, but results were generally unpredictable. In the windows experiment, the unsmoothed LDD output always surpassed even the smoothed output of SBL. This was mostly true as well for the poles experiment, save for one trial in which smoothed SBL performed just as well as smoothed LDD. Finally, note that the consistency of SBL between trials was not obviously improved by smoothing.

For a number of reasons, a direct comparison of observed computation times would not be fair at this point, but to give a rough idea, SBL plus smoothing usually terminated in a few seconds, while LDD plus smoothing usually terminated in $12-20$ seconds. The planning times of LDD were entirely dependent on how many iterations we allowed it to run, resolution parameters, number of samples drawn in the learning phase, and other details. These parameters were set to produce reliable, high-quality results for the given experiments. However, LDD could also be tuned to terminate in roughly 1-3 seconds to find plans for relatively simple problems, and even some slightly more challenging ones. Parallelization of the cost computation and learning steps, which together account for most of the planning time, are anticipated to decrease planning times even further.


Figure 10.8: Experimental comparison of repeatability and quality of trajectories obtained with LDD and SBL, with and without post-smoothing, for each of two scenarios. Each plot shows 35 joint angle trajectories: one for each matching of seven joints and five trials. Trajectories are plotted with (offset-normalized) joint angle on the horizontal axis and arc length (i.e., time) on the vertical axis. A thick horizontal line is plotted to mark the arc length at the end of each seven-dimensional trajectory. SBL produces trajectories that are generally long and highly variable across trials. Although smoothing helps to narrow the gap somewhat, LDD produces trajectories that are much more smooth and consistent across trials. Post-smoothing has little effect on the LDD trajectories, presumably because they are nearly optimal in the first place.

## Chapter 11

## Assumptions, guarantees, and limitations

The purpose of this chapter is to compare and summarize the algorithms presented in the second part of this work, with the intent of elucidating some of the guarantees and limitations of the methods. By doing so, it is hoped that the relative advantages and disadvantages of these methods with respect to others might be made more clear, while also providing indications of areas that could stand to benefit from further research.

### 11.1 SLASHDP

As recounted in Chapter 9, it was observed experimentally that SLASHDP found very high-quality solutions for some very high-dimensional problems by performing DP in a three- or four-dimensional space. In this sense, SLASHDP was observed to dominate other methods in the sense that it either found solutions of higher quality or in higher-dimensional spaces than other methods. It is worth emphasis that this is likely only possible because SLASHDP profits from making additional assumptions that are not made in traditional motion planning algorithms based either on sampling-based algorithms or deterministic search. Either of the latter methods, for instance, can be applied to problems in arbitrary metric spaces (given a means of sampling the space appropriately), whereas SLASHDP, at least as presented, is limited to Euclidean spaces (or spaces approximately mappable to Euclidean spaces, such as the case in arm planning). SLASHDP is furthermore currently limited to solving problems with holonomic constraints only, whereas more common methods can handle non-holonomic constraints easily.

Additionally, there is no reason to expect that SLASHDP will perform well for problems that do not possess approximate low-dimensional cost structure. This expectation is rooted
in the fact that the entire motivation for SLASHDP rests on the observation that if the cost function is exactly low-dimensional, then a globally optimal solution may be found by performing DP in a low-dimensional space-i.e., if $C(x)=C\left(W W^{T} x\right)$, $\forall x$, for $W \in \mathbb{R}^{N \times d}$, then the optimal solution may be found by performing DP in a $d+1$-dimensional space. If such a relation holds only approximately, then we can still expect SLASHDP to produce approximately optimal solutions.

One way to formalize this intuition is by analyzing a slightly modified version of SLASHDP, where we assume the compressed cost $C^{\prime}$ obeys $C^{\prime}(x) \geq C(x)$ and $C^{\prime}(x)-$ $C(x) \leq \delta, \forall x$. We then have the following result.

Theorem 11.1.1. Suppose that SLASHDP is applied to a problem with cost function $C$, and a compressed cost $C^{\prime}$ is obtained such that $C^{\prime}(x) \geq C(x)$ and $C^{\prime}(x)-C(x) \leq \delta$, $\forall x$. Let $J\{x ; C\}=\int\|\dot{x}\| C(x) d t$ denote the cost functional evaluated on path $x$ using cost function $C$, let $x^{*}=\arg \min _{x} J\{x ; C\}$, and let $x^{*}=\arg \min _{x} J\left\{x ; C^{\prime}\right\}$. Then, assuming $C(x) \geq 1, \forall x$,

$$
\begin{equation*}
\frac{J\left\{x^{\prime *} ; C\right\}-J\left\{x^{*} ; C\right\}}{J\left\{x^{*} ; C\right\}} \leq \delta . \tag{11.1.1}
\end{equation*}
$$

Proof. We first have that for any path $x(t)$,

$$
\begin{equation*}
J\left\{x ; C^{\prime}\right\}-J\{x ; C\}=\int\left(C^{\prime}(x)-C(x)\right)\|\dot{x}\| d t \leq \delta \int\|\dot{x}\| d t \tag{11.1.2}
\end{equation*}
$$

We also observe that

$$
\begin{equation*}
J\left\{x^{*} ; C\right\} \leq J\left\{x^{* *} ; C\right\} \leq J\left\{x^{\prime *} ; C^{\prime}\right\} \leq J\left\{x^{*} ; C^{\prime}\right\} . \tag{11.1.3}
\end{equation*}
$$

Therefore,

$$
\begin{align*}
J\left\{x^{\prime *} ; C\right\}-J\left\{x^{*} ; C\right\} & \leq J\left\{x^{*} ; C^{\prime}\right\}-J\left\{x^{*} ; C\right\} \leq \delta \int\left\|\dot{x}^{*}\right\| d t \\
& \leq \delta J\left\{x^{*} ; C\right\}, \tag{11.1.4}
\end{align*}
$$

which is equivalent to the desired result.
This simple analysis intuitively connects our ability to compress the cost function with our ability to obtain a provably suboptimal solution. However, it suffers from a few significant shortcomings in practice. First, it requires that the compressed cost function be an upper bound on the true cost function, which would be impossible to ensure in practice without making further assumptions. Furthermore, it is only useful if we expect that a small maximum difference between the compressed and true cost functions, which may not be reasonable in many cases - we may often obtain a good compression of the cost function
overall that greatly differs from the true cost function only in sets of small total measure, in which case this analysis is overly pessimistic.

In practice, there are problems with insufficient low-dimensional structure to exploit via performing DP once in three or four dimensions, which is approximately the maximumsize DP problem that can be solved with the fast marching method. This was seen in Section 9.4.2, where only modest benefits were obtained using SLASHDP over a naive method for deformable robot planning. Performing SLASHDP locally was observed to be one way in which to partially overcome this problem. A more insightful view, however, led to the development of LDD.

### 11.2 LDD

As previously mentioned, LDD is mostly a generalization of SLASHDP, except for the small detail of how the cost function is compressed. Given a $d$-dimensional learned basis, SLASHDP produces a compressed cost function that is a function only of the projection of the state onto this basis. A $d+1$-dimensional value function is then computed, from which any value in the state space may be computed via Corollary 4.3.2. LDD simply computes the $d+1$-dimensional value function associated with the cost restricted to the $d+1$-dimensional subspace in which the path would be guaranteed to lie (assuming fixed endpoints), if the cost were a function only of the $d$-dimensional basis.

The subspace searched by SLASHDP is therefore the same subspace as that searched in the first iteration of LDD, for a fixed basis. LDD finds the optimal path restricted to that subspace, while SLASHDP finds the optimal path with respect to the compressed cost function, which happens to lie in the same subspace. LDD's solution after a single iteration is consequently guaranteed to be of equal or lesser cost than that of SLASHDP. Note, however, that applying SLASHDP may still have merit for the multiple-query problem, in which case it may not be optimal to compress the cost function by restricting it to a particular subspace.

Since it is guaranteed to produce a better solution for a fixed basis and endpoints, Theorem 11.1.1 may be applied to the first iteration of LDD as well. LDD, however, continues to improve this solution by optimization over a sequence of submanifolds. In the continuous version of this procedure, monotonic convergence to a local optimum is guaranteed, since the feasible set of each optimization problem (i.e., the submanifold) contains the optimal solution of the last. When implemented on a computer with finite resources, however, this property may no longer hold due to discretization effects.

A potential concern with LDD arises with regard to the nature of the local minima found. Obviously, one can always choose a basis of large enough dimension such that
no local minima are possible, but doing so might require searching over the entire state space. The interesting local minima are therefore those that occur while searching over low-dimensional submanifolds.

Figure 11.1a illustrates a local minimum that occurs in a three-dimensional problem if two-dimensional search submanifolds are employed. The figure illustrates a cage-like obstacle created by carving cubes out of a large hollow cube at the corners. It is easy to verify that the learned basis in this case, assuming a reasonable cost function, is aligned precisely to the faces of the cage, and the associated eigenvalue spectrum is flat due to symmetry-this problem consequently has no low-dimensional cost structure. The endpoints are illustrated as green spheres, and the initial solution is illustrated as a red tube drawn between them. In the first iteration, the plane illustrated by the grid is searched for an improved solution; however, no progress is possible, as no collision-free path exists between the endpoints lying in the plane. The same is true for all subsequent iterations, by symmetry. A local minimum is therefore immediately encountered, and LDD terminates without having improved the initial solution at all. Figure 11.1b shows how this local minimum is fairly unstable with respect to changes in the cost function. In this case, a hole in the cage at the level of any search plane ensures that LDD is able to find a better, collision-free local minimum. It is also worth noting that introducing the hole creates an asymmetry that breaks the flatness of the eigenvalue spectrum, thus creating at least some low-dimensional structure.

In practice, minima of the sort encountered in Fig. 11.1a could be escaped by introducing a small amount of random noise either into the path or the basis selection, which would likely result in a submanifold containing collision-free paths. Alternatively, raising the dimension of the searched submanifolds by one would cure the problem, although this would entail searching a space of dimensionality equal to the original space for this particular problem-unless we regard the figure as illustrating a projection of a higherdimensional problem with low-dimensional strucutre, in which case a net efficiency gain would be possible.

### 11.3 Fermat heuristic

Chapter 8 described how low-dimensional structure might be leveraged in the context of more traditional methods such as deterministic search and randomized planning. A particular advantage of doing so is that it enables us to relax the assumption of holonomic constraints that limits SLASHDP and LDD, since the Fermat heuristic is defined as the optimal value of the original problem with non-holonomic constraints relaxed (and with low-dimensional structure enforced).

(a) Poor local minimum

(b) Better local minimum

Figure 11.1: Illustration of LDD local minima with respect to two-dimensional search submanifolds. Fig. 11.1a shows a poor local minimum for a certain problem, while Fig. 11.1b shows a much better local minimum obtained in a slightly modified problem. Black arrows show learned basis vectors.

In the case of deterministic search, by constructing the cost function associated with the Fermat heuristic to lower-bound the true cost, we obtain an admissible heuristic, which guarantees that an optimal solution will be found when $A^{*}$ is equipped with it. Applying weighted $\mathrm{A}^{*}$ with this heuristic therefore guarantees at least a provably suboptimal solution. Finally, the Fermat heuristic is consistent, guaranteeing that A* will not reopen states when used in conjunction with it.

Although this was not discussed previously, SLASHDP might be applied to automatically find the low-dimensional structure used to compute the Fermat heuristic. Doing so would likely be cost-effective for high-dimensional problems without good alternative heuristics, and we would further obtain a method making fewer assumptions while providing stronger optimality guarantees.

## Chapter 12

## Future directions: structured planning

It is believed that the current work has only scratched the surface of what advances may be possible by reformulating the planning problem in terms of how best to discern and exploit structure in an automated manner. Some general thoughts are given here as to how one might proceed to further develop the methods presented here.

### 12.1 Structured motion planning

A disadvantage of most of the methods presented here, is that they are not directly applicable to problems with non-holonomic constraints. One way to avoid this issue is by using these methods to generate strong heuristic functions to guide deterministic (or randomized) search, as discussed in Chapter 8. Nonetheless, it is hoped that there exist more direct ways of exploiting the symmetries present in non-holonomic problems. The fact that such symmetries necessarily simplify the planning problem in a precisely defined way, as related in Section 7.3.3, provides strong inducement to explore this issue more in earnest from an algorithmic perspective.

Non-holonomic constraints might also be addressed by adopting a technique similar to that exploited by navigation functions [79]; namely, trading optimality of the value function for smoothness. Although explicit enumeration of the configuration-space obstacles impeded the application of such methods to arbitrary problems in high-dimensional spaces, the methods presented here can be seen as approximately enumerating them in lower-dimensional spaces; perhaps ideas from these methods could therefore be combined to overcome such difficulties.

Computational efficiency of the methods developed here is a bit shy of being competitive with sampling-based planners. Fortunately, much of this could be addressed with more efficient implementations and parallelization, since the learning and cost evaluation steps are embarrassingly parallel. The ultimate performance bottleneck is therefore expected to be the computation of the low-dimensional value function via the Fast Marching Method. Surprisingly, no equivalent of the A* algorithm seems to exist currently for the FMM. If such a method could be developed, it might be the key to achieving real-time performance. Furthermore, a heuristic-based FMM may enable the use of higher-dimensional optimization steps, which may be necessary to avoid local minima in more difficult problems. The analogous MDP version, on the other hand, may produce more interesting behavior.

Theoretically, many open questions remain as to how best to characterize the approximation performance of the algorithms. For SLASHDP, a desirable result might involve bounding the worst-case approximation error in terms of the spectrum of the matrix of second moments of the cost gradient and possibly other cost statistics. For LDD, a better understanding of the nature of the local minima encountered and a way to quantify its convergence rate would be beneficial.

### 12.2 Structured discrete planning

A large part of the distinctive character of this work was the exploitation of structure that arises directly as a result of the continuous nature of the motion planning problems that many other methods simply ignore. However, the nature of the algorithms that resulted provide clues as to how one might develop analogous methods for discrete planning problems.

In fact, results very similar to those presented in Chapter 4 can be derived for minimum-cost-path problems on graphs with a product structure. Namely, if a graph is composed of an arbitrary product of graphs and the cost function factorizes over the product in such a way as to be independent of some of the graphs in the product, then the value function can again be expressed as a lower-dimensional object; this time parameterized not by norm of the projection onto the cyclic subspace, but by the sum of values associated with those graphs in the product upon which the cost does not depend. As the resulting nature of the optimal paths is somewhat trivial, it is yet to be seen whether this leads to a useful principle for discrete planning.

Another avenue for future exploration might be to develop LDD-like algorithms for discrete planning. One could also envision solving high-dimensional problems again by solving a sequence of low-dimensional dynamic programming problems, this time over subgraphs instead of submanifolds. As long as consecutive subgraphs were to contain the
current path, monotonic convergence could be guaranteed, much as in LDD. It is unclear, however, what the equivalent of the LDD learning procedure would entail in this context.

## Chapter 13

## Conclusions

The central tenet of this work, as it is hoped the reader will have gathered by this point, is that low-dimensional structure is key to understanding and solving problems associated with the surprisingly diverse class of high-dimensional Lagrangian systems. This tenet presupposes a continuous setting for the analysis of such systems often neglected by researchers in both machine learning and motion planning; quite unfortunately so, as making this simple assumption allows us to bring to bear powerful tools and basic concepts from physics and optimal control. More fundamentally, it induces us to think about ways to impose interesting structures on these problems that otherwise would have been lost in the discrete abstractions commonly used to analyze such systems.

Taking such a view initially enabled us to discover a more natural way of thinking about the low-dimensional structure of physical sequence data. Rather than the common approach of modeling high-dimensional systems as marionettes controlled by lowdimensional actors, we saw how the presence of conservation laws induces a factorization of the high-dimensional dynamics into complex low-dimensional dynamics and conditionally simple high-dimensional dynamics. A concrete algorithm (FCA) was given to discover this factorization from trajectory data that addressed many potential issues with the concept of learning structured Lagrangians.

Solving the problem of inference in this model was shown to be equivalent to solving a high-dimensional motion planning problem with a certain structured cost function. We saw that the value function associated with this problem, is symmetric about rotations that preserve the cost function, enabling us to directly compute a value function in a compressed form. Equivalently, the optimal paths were shown to lie in low-dimensional subspaces.

These results led immediately to novel practical algorithms for motion planning in
high-dimensional spaces. In the course of developing these, we developed novel, higherdimensional generalizations of the Euclidean heuristic commonly used in heuristic search (Theorem 8.1.3) and the shortcut heuristic used in sampling-based planning (Section 9.3.4).

Two notable ideas were also developed regarding how to automatically learn and exploit structure in motion planning problems, along with algorithms inspired by these ideas. The first was based on the idea of learning an optimal basis in which to compress a cost function, and subsequently computing a correspondingly compressed value function. The second took the view of learning a good set of coordinates in which to perform Hilbert-space coordinate descent, which had an intuitive geometric description as solving a sequence of low-dimensional dynamic programming problems over ruled surfaces. We saw how this enabled the solution of a variety of problems, ranging from simulated 144 -dimensional planar arms to motion planning for a physical robot. Significantly, it is anticipated that the development of these methods will lead to promising new avenues of research in the motion planning community.

Finally, it is hoped that the view espoused here of learning physical dynamics and motion planning as being unified under the common framework of structured Lagrangian analysis will lead to further advances and fruitful opportunities for cross-fertilization between the fields.

## Bibliography

[1] The Open Motion Planning Library (OMPL), 2010.
[2] P. Abbeel and A.Y. Ng. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning, page 1. ACM, 2004.
[3] R.K. Ahuja and J.B. Orlin. Inverse optimization. Operations Research, pages 771783, 2001.
[4] R.K. Ahuja and J.B. Orlin. Combinatorial algorithms for inverse network flow problems. Networks, 40(4):181-187, 2002.
[5] A. Aizerman, E.M. Braverman, and LI Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. Automation and remote control, 25:821-837, 1964.
[6] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In Proceedings IEEE International Conference on Robotics $\mathcal{E}^{8}$ Automation, pages 113-120, 1996.
[7] V. I. Arnold. Mathematical Methods of Classical Mechanics. Springer-Verlag, 1989.
[8] J. Barraquand and P. Ferbach. Path planning through variational dynamic programming. In IEEE International Conference on Robotics and Automation, pages 1839-1846. IEEE, 1994.
[9] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. The International Journal of Robotics Research, 10(6):628, 1991.
[10] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. The annals of mathematical statistics, 41(1):164-171, 1970.
[11] R. E. Bellman. Dynamic Programming, chapter Preface, page ix. Princeton University Press, 1957.
[12] Richard Bellman. A Markovian decision process. Indiana Univ. Math. J., 6:679-684, 1957.
[13] Richard Bellman and Stuart Dreyfus. Functional approximations and dynamic programming. Mathematical Tables and Other Aids to Computation, 13(68), October 1959.
[14] D. Berenson, S.S. Srinivasa, D. Ferguson, A. Collet, and J.J. Kuffner. Manipulation planning with workspace goal regions. In ICRA, pages 618-624. IEEE, 2009.
[15] S. Boyd, V. Balakrishnan, E. Feron, and L. ElGhaoui. Control system analysis and synthesis via linear matrix inequalities. In American Control Conference, 1993, pages 2147-2154. IEEE, 1993.
[16] O. Brock and L.E. Kavraki. Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces. In ICRA, 2001.
[17] MA Brubaker and DJ Fleet. The Kneed Walker for human pose tracking. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pages 1-8. IEEE, 2008.
[18] M.A. Brubaker, D.J. Fleet, and A. Hertzmann. Physics-based person tracking using simplified lower-body dynamics. In 2007 IEEE Conference on Computer Vision and Pattern Recognition, pages 1-8. IEEE, 2007.
[19] M.A. Brubaker, D.J. Fleet, and A. Hertzmann. Physics-based person tracking using the anthropomorphic walker. International journal of computer vision, 87(1):140155, 2010.
[20] V. Bulitko, N. Sturtevant, and M. Kazakevich. Speeding up learning in real-time search via automatic state abstraction. In Proceedings of the National Conference on Artificial Intelligence, 2005.
[21] D. Burton and P.L. Toint. On an instance of the inverse shortest paths problem. Mathematical Programming, 53(1):45-61, 1992.
[22] S. Chitta, B. Cohen, and M. Likhachev. Planning for autonomous door opening with a mobile manipulator. In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 1799-1806. IEEE, 2010.
[23] A. Clark and F. Thollard. PAC-learnability of probabilistic deterministic finite state automata. The Journal of Machine Learning Research, 5:473-497, 2004.
[24] B.J. Cohen, S. Chitta, and M. Likhachev. Search-based planning for manipulation with motion primitives. In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 2902-2908. IEEE, 2010.
[25] C. Connolly and R. Grupen. The application of harmonic potential functions to robotics. Journal of Robotic Systems, 10(7):931-946, 1993.
[26] I.A. Şucan and L.E. Kavraki. On the implementation of single-query sampling-based motion planners. In ICRA, pages 2005-2011, Anchorage, Alaska, May 2010.
[27] J.C. Culberson and J. Schaeffer. Pattern databases. Computational Intelligence, 14(3):318-334, 1998.
[28] S. Dalibard and J.P. Laumond. Control of probabilistic diffusion in motion planning. Algorithmic Foundation of Robotics VIII, pages 467-481, 2009.
[29] Per-Erik Danielsson. Euclidean distance mapping. Computer Graphics and Image Processing, 14(3):227-248, 1980.
[30] D.P. De Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. Operations Research, pages 850-865, 2003.
[31] D.P. de Farias and B. Van Roy. On constraint sampling in the linear programming approach to approximate dynamic programming. Mathematics of Operations Research, pages 462-478, 2004.
[32] M.P. Deisenroth, C.E. Rasmussen, and J. Peters. Gaussian process dynamic programming. Neurocomputing, 72(7-9):1508-1524, 2009.
[33] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner. Bispace planning: Concurrent multi-space exploration. $R S S, 2008$.
[34] J. Gaschnig. A problem similarity approach to devising heuristics: First results. In Proceedings of the 6th international joint conference on Artificial intelligence-Volume 1, pages 301-307. Morgan Kaufmann Publishers Inc., 1979.
[35] J. G. Gaschnig. Performance measurement and analysis of certain search algorithms. PhD thesis, Carnegie Mellon University, 1979.
[36] Roland Geraerts and Mark Overmars. Clearance based path optimization for motion planning. In ICRA, pages 2386-2392, 2004.
[37] Z. Ghahramani and S.T. Roweis. Learning nonlinear dynamical systems using an EM algorithm. In Advances in Neural Information Processing Systems 11: Proceedings of the 1998 Conference, pages 431-437, 1999.
[38] G.J. Gordon. Stable function approximation in dynamic programming. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 1995.
[39] J. W. Grizzle and S. I. Marcus. Optimal control of systems possessing symmetries. IEEE Transactions on Automatic Control, AC-29(11), 1984.
[40] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. Systems Science and Cybernetics, IEEE Transactions on, 4(2):100-107, 1968.
[41] J. Haugeland. Artificial intelligence: The very idea. The MIT Press, 1989.
[42] K. Hauser, T. Bretl, J.C. Latombe, and B. Wilcox. Motion planning for a six-legged lunar robot. Algorithmic Foundation of Robotics VII, pages 301-316, 2008.
[43] R.C. Holte, M.B. Perez, R.M. Zimmer, and A.J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. In Proceedings of the National Conference on Artificial Intelligence, pages 530-535, 1996.
[44] D. Hsu, S.M. Kakade, and T. Zhang. A spectral algorithm for learning hidden markov models. Arxiv preprint arXiv:0811.4413, 2008.
[45] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. International Journal Computational Geometry ${ }^{8}$ Applications, 4:495-512, 1999.
[46] R. E. Kalman. When is a linear control system optimal? Trans. ASME, J. Basic Eng., 1964.
[47] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In RSS, Zaragoza, Spain, June 2010.
[48] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation, 1996.
[49] O. Khatib. Commande dynamique dans l'espace opérational des robots manipulateurs en présence d'obstacles. PhD thesis, Ecole Nationale de la Statistique et de l'Administration Economique, France, 1980.
[50] J. Ko and D. Fox. Learning GP-BayesFilters via Gaussian process latent variable models. Autonomous Robots, pages 1-21, 2009.
[51] J.Z. Kolter, M.P. Rodgers, and A.Y. Ng. A control architecture for quadruped locomotion over rough terrain. In Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, pages 811-818. Ieee, 2008.
[52] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2000.
[53] S. M. LaValle. Planning Algorithms. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.
[54] S. M. LaValle. Planning Algorithms, chapter 6: Combinatorial Motion Planning. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.
[55] Neil D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, Advances in Neural Information Processing Systems 16. MIT Press, Cambridge, MA, 2004.
[56] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), pages 262-271, 2005.
[57] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In NIPS, 2004.
[58] M. Likhachev and A. Stentz. R* search. In AAAI, 2008.
[59] C.K. Liu, A. Hertzmann, and Z. Popović. Learning physics-based motion style with nonlinear inverse optimization. ACM Transactions on Graphics (TOG), 24(3):10711081, 2005.
[60] Lennart Ljung. Theory and practice of recursive identification. MIT Press, 1983.
[61] Lennart Ljung. System Identification: Theory for the User. Prentice-Hall, 1987.
[62] E. Mossel and S. Roch. Learning nonsingular phylogenies and hidden Markov models.
In Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, pages 366-375. ACM, 2005.
[63] J. Mostow and A.E. Prieditis. Discovering admissible heuristics by abstracting and optimizing: a transformational approach. In Proceedings of the 11th international joint conference on Artificial intelligence-Volume 1, pages 701-707. Morgan Kaufmann Publishers Inc., 1989.
[64] Radford M. Neal. Bayesian Learning for Neural Networks. PhD thesis, University of Toronto, 1994.
[65] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous inverted helicopter flight via reinforcement learning. Experimental Robotics IX, pages 363-372, 2006.
[66] A.Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In Proceedings of the Seventeenth International Conference on Machine Learning, pages 663-670, 2000.
[67] N. J. Nilsson. Principles of Artificial Intelligence. Tioga Publishing Company, Wellsboro, PA, 1980.
[68] P. V. Overschee and B. De Moor. Subspace Identification of Linear Systems. Kluwer Academic Publishers, 1996.
[69] P. Poupart and C. Boutilier. Value-directed compression of POMDPs. Advances in Neural Information Processing Systems, pages 1579-1586, 2003.
[70] Armand E. Prieditis. Machine discovery of effective admissible heuristics. Machine Learning, 1993.
[71] Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. In In Proceedings of the International Conference on Robotics and Automation, pages 802-807, 1993.
[72] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 1989.
[73] N. Ratliff, D. Bradley, J.A. Bagnell, and J. Chestnutt. Boosting structured prediction for imitation learning. Advances in Neural Information Processing Systems, 19:1153, 2007.
[74] N. Ratliff, M. Zucker, J.A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pages 489-494. IEEE, 2009.
[75] Nathan Ratliff, J. Andrew (Drew) Bagnell, and Martin Zinkevich. Maximum margin planning. In International Conference on Machine Learning, July 2006.
[76] H.E. Rauch, CT Striebel, and F. Tung. Maximum likelihood estimates of linear dynamic systems(Maximum likelihood estimates of states of linear dynamic systems in presence of additive Gaussian noise). AIAA journal, 3:1445-1450, 1965.
[77] B. Ravindran and A. G. Barto. SMDP homomorphisms: an algebraic approach to abstraction in semi Markov decsion processes. In Eighteenth International Joint Conference on Artificial Intelligence (IJCAI), 2003.
[78] J.R. Rebula, P.D. Neuhaus, B.V. Bonnlander, M.J. Johnson, and J.E. Pratt. A controller for the littledog quadruped walking on rough terrain. In Robotics and Automation, 2007 IEEE International Conference on, pages 1467-1473. IEEE, 2007.
[79] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential fields. IEEE Transactions on Robotics E Automation, 8(5):501-518, October 1992.
[80] N. Roy and G. Gordon. Exponential family PCA for belief compression in POMDPs. Advances in Neural Information Processing Systems, pages 1667-1674, 2003.
[81] G. Sánchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In Proceedings International Symposium on Robotics Research, 2001.
[82] M. Sekimoto, S. Arimoto, S. Kawamura, and Ji-Hun Bae. Skilled-motion plannings of multi-body systems based upon Riemannian distance. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2008.
[83] J A Sethian. A fast marching level set method for monotonically advancing fronts. Proceedings of the National Academy of Sciences of the United States of America, 93(4):1591-1595, 1996.
[84] A. Shkolnik and R. Tedrake. Path planning in 1000+ dimensions using a task-space Voronoi bias. In ICRA, 2009.
[85] Jonas Sjöberg, Qinghua Zhang, Lennart Ljung, Albert Benveniste, Bernard Delyon, Pierre-Yves Glorennec, Håkan Hjalmarsson, and Anatoli Juditsky. Nonlinear blackbox modeling in system identification: a unified overview. Automatica, 31(12):1691 - 1724, 1995. Trends in System Identification.
[86] E. Solak, R. Murray-Smith, W.E. Leithead, D.J. Leith, and C.E. Rasmussen. Derivative observations in Gaussian process models of dynamic systems. Advances in Neural Information Processing Systems, pages 1057-1064, 2003.
[87] A. Stentz. Optimal and efficient path planning for partially-known environments. In Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, pages 3310-3317. IEEE, 1994.
[88] A. Stentz. The focussed $\mathrm{D}^{\wedge *}$ algorithm for real-time replanning. In International Joint Conference on Artificial Intelligence, volume 14, pages 1652-1659. Citeseer, 1995.
[89] I. A. Sucan and L. E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In WAFR, Guanajuato, Mexico, 2008.
[90] I.A. Sucan and L.E. Kavraki. On the implementation of single-query sampling-based motion planners. In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 2005-2011. IEEE, 2010.
[91] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. The MIT press, 1998.
[92] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves masterlevel play. Neural computation, 6(2):215-219, 1994.
[93] J.N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. Automatic Control, IEEE Transactions on, 40(9):1528-1538, 1995.
[94] Raquel Urtasun, David J. Fleet, and Pascal Fua. 3d people tracking with Gaussian process dynamical models. In Proc. of Computer Vision and Pattern Recognition, 2006.
[95] M. Valtorta. A result on the computational complexity of heuristic estimates for the A* algorithm. Information Sciences, 34(1):47-59, 1984.
[96] P. Van Overschee and B. De Moor. N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems* 1. Automatica, 30(1):75-93, 1994.
[97] B. Van Roy. Learning and value function approximation in complex decision processes. PhD thesis, Citeseer, 1998.
[98] Paul Vernaza and Daniel D. Lee. Efficient dynamic programming for highdimensional, optimal motion planning by spectral learning of approximate value
function symmetries. In IEEE International Conference on Robotics and Automation (ICRA), 2011.
[99] Paul Vernaza and Daniel D. Lee. Learning dimensional descent for optimal motion planning in high-dimensional spaces. In Twenty-Fifth AAAI Conference on Artificial Intelligence, 2011.
[100] Paul Vernaza, Daniel D. Lee, and Seung-Joon Yi. Learning and planning highdimensional physical trajectories via structured Lagrangians. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2010.
[101] Paul Vernaza, Maxim Likhachev, Subhrajit Bhattacharya, Sachin Chitta, Aleksandr Kushleyev, and Daniel D. Lee. Search-based planning for a legged robot over rough terrain. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2009.
[102] Jack Wang, David Fleet, and Aaron Hertzmann. Gaussian process dynamical models. In Y. Weiss, B. Schölkopf, and J. Platt, editors, Advances in Neural Information Processing Systems 18, pages 1441-1448. MIT Press, Cambridge, MA, 2006.
[103] J.M. Wang, D.J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. IEEE transactions on pattern analysis and machine intelligence, pages 283-298, 2007.
[104] C.J.C.H. Watkins and P. Dayan. Q-learning. Machine learning, 8(3):279-292, 1992.
[105] Martin Zinkevich and Tucker Balch. Symmetry in markov decision processes and its implications for single agent and multi agent learning. In In Proceedings of the 18th International Conference on Machine Learning, pages 632-640. Morgan Kaufmann, 2001.
[106] M. Zucker, J.A. Bagnell, C.G. Atkeson, and J. Kuffner. An optimization approach to rough terrain locomotion. In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 3589-3595. IEEE, 2010.


[^0]:    Daniel D. Lee, Associate Professor
    Electrical and Systems Engineering Supervisor of Dissertation

