Improving Robot Locomotion Through Learning Methods for Expensive Black-Box Systems

Matthew Tesch

CMU-RI-TR-14-1

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

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

January/2014

Thesis Committee:
Howie Choset, Jeff Schneider, Drew Bagnell, Stefan Schaal, Jared Cohon

Copyright © 2014 by Matthew Tesch. All rights reserved.

Abstract

The modular snake robots in Carnegie Mellons Biorobotics lab provide an intriguing platform for research: they have already been shown to excel at a variety of locomotive tasks and have incredible potential for navigating complex terrains, but much of that potential remains untapped. Unfortunately, many techniques commonly used in robotics prove inapplicable to these snake robots. This is because of the robots complex multi-modal locomotion dynamics, which are difficult to model, and their small size and frequent impacts, which preclude the addition of many standard sensors.

The motivation to expand the capabilities of these robots stems from experiencing several failures and limitations in real world tests. In an archaeological expedition near the Red Sea, the robot was able to move further than a human could into a collapsed cave containing four-millenia-old ship timbers. However, a gradual sandy slope prevented the robot from moving further and potentially making an archaeological discovery. At a disaster response training site, the robot was able to navigate a narrow passage underneath a rubble pile, but was unable to pass over a four inch high piece of wood which lay across its path once the passage widened.

This thesis addresses the improvement of these capabilities by developing methods for the optimization of functions which are expensive (requiring significant time, money, computation, or other resources), black-box (providing no gradient or derivative information), need not be convex or linear, and may have many local optima. Objectives evaluated through tests on physical robotic systems often fit these categories.

Several approaches are derived and tested for optimization of snake robot gait motion, leading to improved locomotion across flat and sloped terrain. Additional unique challenges posed by robotic systems are addressed, including stochasticity in the objective, consideration of multiple conflicting objectives, and the desire to adapt to changing environments.

Although gaits are the motion of choice for traversing long distances over uniform terrain, real-world environments will rarely be completely uniform. Instead, complex motions also must be learned and optimized that enable navigation over complex terrain and large obstacles. To address this challenge, I describe an approach to record, simplify, and parameterize demonstrated trajectories from expert and novice users. As the settings which require such motions usually can only quantify the result of the motion in terms of success and failure rather than a numerical score, I derive extensions to the optimization framework used for improving gaits to handle stochastic binary functions, and use this to optimize robustness of trajectories for moving over obstacles.

Overall, these algorithms allow snake robot locomotion through any type of environment to be optimized. Furthermore, the generality inherent in the black-box approach allows these techniques to be applicable to a wide variety of problems in robotics.

Acknowledgments

This journey would not have been possible without the support of many people over the past years.

I would like to thank the members of the Biorobotics Lab, those who helped develop the robot and so many of the tools that made it usable, those who weren't afraid to tell me when my crazy ideas were not going to work, and those who kept the robot working so I always had a reliable platform. And of course, thanks go to Peggy Martin, without who the lab would certainly cease to function!

I also owe thanks to the collaborators I have worked with and discussed ideas with outside of the lab, especially those at the Museum of Science and Industry for being so supportive during three days of data collection. I also appreciate the support of members of CMU's Auton lab for graciously adopting me during conference preparation and for supplying computation resources.

I am grateful for the support of my committee, especially for the advice and time given by my co-advisors, Howie Choset and Jeff Schneider.

Finally, I thank my family for their love and support. My parents have encouraged and helped me through so many years of school and life – from the hours spent with me on my elementary school class papers and science fair projects to those teaching bigger life lessons. I now realize how they laid a solid foundation for all that I do. This thanks is also especially deserved by my wife for her generosity in proofreading everything I wrote and her endless patience as she endured late nights in the lab, pre-deadline stresses, and uncertain future plans.

Contents

List Of Figures **VI**

List Of Algorithms **VIII**

List Of Tables **IX**

List of Abbreviations and Symbols **X**

1 Introduction **1**

2 Background **6**
2.1 Gaussian Processes for Regression 6
2.1.1 Gaussian Process Priors 6
2.1.2 Gaussian Process Definition 10
2.1.3 Sampling from the Prior 11
2.1.4 Regression and the Gaussian Process Posterior 12
2.1.5 Gaussian process Posterior Visualization 15
2.1.6 GP Model Selection 16
2.1.7 Fitting GPs to Stochastic Data 18
2.1.8 Gaussian Process Summary 20
2.2 Gaussian Processes for Classification 20
2.3 Snake Robot Control 23
2.4 Expensive Optimization 26
2.5 Multi-objective Optimization 30

3 Related Work **34**
3.1 Expensive Optimization 34
3.1.1 Active Learning 35
3.1.2 Bandits 35
3.1.3 Optimization of Expensive Stochastic Functions 37
3.2 Environmentally Adaptive Optimization 37
3.3 Multi-objective Optimization 40
3.4 Locomotion Over Obstacles 41
3.5 Stochastic Binary Optimization 43

4 Expensive Optimization for Robot Locomotion **45**
4.1 Notation and Problem Statement . . . 48
4.2 Improvements For Robust and Effective Optimization . . . 50
4.2.1 Model Selection: Hyperparameters . . . 51
4.2.2 Model Selection: Mean and Covariance Function . . . 55
4.2.3 Fitting GPs to Stochastic Data . . . 57
4.2.4 Demonstration of Algorithm Improvements . . . 58
4.3 Extensions for Stochastic Objectives . . . 62
4.3.1 Baseline Approaches . . . 63
4.3.2 Bayesian Approaches . . . 65
4.3.3 Stochastic Optimization Results . . . 67
4.4 Optimization of Snake Robot Locomotion . . . 71

5 Multiple Objectives **75**
5.1 Objective Space . . . 77
5.2 Extending EI to Multiple Objectives . . . 79
5.3 Synthetic Test Problem Results . . . 82
5.4 Robot Results . . . 84
5.5 Conclusions and Future Work . . . 88

6 Environmentally Adaptive Optimization **90**
6.1 Problem Definition and Notation . . . 93
6.2 Proposed Methods . . . 96
6.2.1 Policy Generation . . . 98
6.2.2 Experiment Selection: Unbiased Expected Improvement . . . 100
6.2.3 Experiment Selection: Expected Policy Score Improvement . . . 101
6.2.4 Method Comparison and Discussion . . . 103
6.3 Experimental Results . . . 104
6.3.1 Synthetic Test Functions . . . 104
6.3.2 Simulation and Physical System Results . . . 108
6.3.3 Application for Real-Time Control . . . 111
6.4 Conclusion . . . 119

7 Generating Parameterized Non-gait Motion from Demonstrated Input **121**
7.1 Kinesthetic Input for Recording Demonstrated Input . . . 123
7.1.1 Expert Input . . . 124
7.1.2 Novice Input . . . 126
7.2 Identifying Keyframes For Single Recorded Expert Trajectory . . . 132
7.2.1 Method . . . 133
7.2.2 Results . . . 135
7.3 Finding Parameterized Trajectories Using Large Collections of Novice Trajectories . . . 137
7.4 Conclusions and Future Work . . . 139

8 Stochastic Binary Optimization **141**
8.1 Binary Stochastic Problem Definition . . . 143
8.2 Approach . . . 144
8.2.1 Baseline Algorithms . . . 144
8.2.2 Expected Improvement for Binary Responses . . . 146
8.3 Empirical Results on Synthetic Functions . . . 148
8.3.1 Experimental Setup . . . 148
8.3.2 Measuring Performance . . . 150
8.3.3 Comparison of Results . . . 150
8.4 Physical Robot Experiments . . . 152
8.5 Exploiting Task Structure to Solve Difficult Problems . . . 155
8.6 Conclusion and Future work . . . 159

9 Conclusion **160**
9.1 The Future of Expensive Optimization . . . 160
9.2 Future Work . . . 163

Appendices **165**

A Test Function Definitions **166**
A.1 Adaptive Control Test Functions . . . 166
A.2 Stochastic Binary Test Functions . . . 167

B Example Optimization Scripts **168**
B.1 Model Selection Example Scripts . . . 168
B.2 Stochastic EI Example Scripts . . . 169
B.3 Stochastic Binary Example Scripts . . . 170

C Further Information **172**
C.1 Marginal Likelihood . . . 172

List of Figures

1.1 Overview of thesis goals 2

2.1 GP mean functions 7
2.2 GP covariance functions 8
2.3 GP hyperparameters 9
2.4 Bivariate and multivariate Gaussian samples 10
2.5 Conditioning to obtain GP posterior distributions 14
2.6 GP visualization 15
2.7 GP hyperparameter likelihood surface; maximum likelihood hyperparameters 17
2.8 Fitting stochastic data with GPs 19
2.9 Uses for snake robots 24
2.10 Concept of expected improvement and probability of improvement 29
2.11 Pareto dominance, Pareto fronts, and hypervolume 32

4.1 Notation for expensive optimization and selection metrics 49
4.2 Non-discriminative GP hyperparameter marginal likelihood for sparse data . 54
4.3 Model complexity vs. likelihood tradeoff 56
4.4 Automatic model selection demonstration 57
4.5 Multimodal likelihood function 58
4.6 Effects of model selection during optimization 61
4.7 Ambiguity in "best previous experiment" in the presence of noise 64
4.8 Synthetic test functions for stochastic optimization 68
4.9 Stochastic optimization results 70
4.10 Computational requirements of stochastic selection metrics 71
4.11 Optimization of snake robot sidewinding motion on flat ground 72
4.12 Optimization of snake robot sidewinding motion on slope 73
4.13 Obstacles on which to demonstrate locomotive optimization 73
4.14 Optimization of snake robot motion over small obstacles 74

5.1 Multi-objective terminology 76
5.2 Multi-objective example problem 76
5.3 Objective space projection 78
5.4 Components of the expected improvement in hypervolume 80
5.5 Example of expected improvement in hypervolume 81

5.6 Synthetic test functions for multi-objective algorithm comparison 82
5.7 Comparison of multi-objective algorithm performance on synthetic test functions 84
5.8 Surrogate function examples for multi-objective robot optimization 85
5.9 Results of multi-objective optimization of robot locomotion 85
5.10 Head stability objective for robot locomotion trials 86

6.1 Diagram of the adaptive expensive optimization problem 91
6.2 Visualization of adaptive optimization terminology and notation 92
6.3 Comparison of optimal and suboptimal policies 95
6.4 Comparison of various selection metrics for the adaptive optimization problem 99
6.5 Conceptual diagram for Unbiased Expected Improvement 101
6.6 Conceptual diagram for Expected Policy Score Improvement 102
6.7 Analytic test functions for empirical algorithm comparisons 104
6.8 Average run time comparison for adaptive optimization selection metrics . . 106
6.9 Empirical comparison of various adaptive optimization selection metrics on synthetic test functions 107
6.10 Empirical analysis of selection metric parameters on algorithm performance . 108
6.11 Empirical comparison of proposed selection metric to random point selection on a physical robot 110
6.12 Diagram of robot test course setup 112
6.13 Photo of training apparatus for learning adaptive control policy for snake robot sidewinding up and down slopes 112
6.14 Objective and selection metric during learning of adaptive policy for snake robot 114
6.15 Resulting policy and predicted objective for adaptive slope sidewinding task 115
6.16 Obtained linear fit for slope angle estimation 118
6.17 Frames from video of snake robot demonstrating optimal policy on test course 118
6.18 Results showing performance of optimized policy to various static policies . . 119

7.1 Kinesthetic input concept and museum data collection overview 122
7.2 Obstacles for collection of expert demonstrations 124
7.3 Fidelity of expert demonstration playback 125
7.4 Images from data collection at the Museum of Science and Industry 129
7.5 Parameterization and subsequent optimization of expert trajectory 134
7.6 Playback comparison for expert demonstration and simplified versions 137

8.1 Results of binary optimization on 3.5 and 11 inch tall beams 142
8.2 Synthetic test functions for stochastic binary optimization 148
8.3 Comparison of algorithm performance on stochastic binary functions 152
8.4 Comparison of algorithm performance on stochastic binary functions 153
8.5 Robot overcoming obstacle using optimized motion 154
8.6 Improving solution variety by biasing against confidence 156
8.7 Results from optimization of robot motion, sharing information across tasks 158

List of Algorithms

1 Basic Black-Box Optimization 50
2 Robust Black-Box Optimization 59
3 Robust GP Fitting 60

List of Tables

7.1 Results of Replaying Recorded Trajectories 126
7.2 Aggregate Success Percentage by Gender and Approximate Age 129
7.3 Results of Simple Parameterized Trajectory Controllers 136

List of Abbreviations and Symbols

A short description of the terms used in this thesis, this is intended to be a quick reference to assist with understanding of formulae and abbreviations. More detailed descriptions are given upon initial use of the terms.

Notation	Description	Page List
CDF	cumulative density function	21, 22, 30, 144, 147
EI	expected improvement	29, 30, 34, 37, 41, 46, 47, 59, 62, 63, 65, 66, 68, 69, 72, 79, 80, 83, 89, 93, 97, 100, 101, 103, 106, 107, 120, 143–148, 150, 152, 153, 155, 159, 163, 168, 169
EIHV	expected improvement in hypervolume	80, 81, 83, 84, 88, 89
EPSI	Expected Policy Score Improvement	IV, 99, 101–103, 105–109
GP	Gaussian process	XIII, 6–16, 18–22, 27, 28, 34, 35, 38, 46–48, 50–53, 55–61, 65–68, 78, 81, 88, 97, 98, 102, 103, 105, 109, 113, 115, 120, 144–146, 149–151, 153, 155, 157, 161–163, 172
GPC	Gaussian processes for classification	20, 21, 143, 144, 147, 157, 159
LOO	leave-one-out	51–53, 59, 61, 161

Notation	Description	Page List
MLE	maximum likelihood estimate	17, 53, 55, 56, 59, 162
MOO	multi-objective optimization	31, 41, 75, 77, 79
PDF	probability density function	10–12, 17, 21, 29, 30, 67, 69, 145
RL	reinforcement learning	38–40, 42, 163
UCB	upper confidence bound	28, 34, 145, 146
UEI	Unbiased Expected Improvement	IV, 99–101, 103, 107, 111, 113
k	Covariance function of the GP; must be positive definite. $k\colon X \times X \to \mathbb{R}$	7–9, 11, 12
K	Shorthand for the GP covariance function k of sets of variables. $K(\{x_1, x_2\}, \{x_3, x_4\}) = \begin{bmatrix} k(x_1, x_3) & k(x_1, x_4) \\ k(x_2, x_3) & k(x_2, x_4) \end{bmatrix}$	12, 14, 17, 19
m	The mean function of the GP. $m\colon X \to \mathbb{R}$	7, 11, 12
M	Shorthand for the GP mean function m of a set of variables. $M(\{x_1, x_2, x_3\}) = \begin{bmatrix} m(x_1) & m(x_2) & m(x_3) \end{bmatrix}^T$	12, 14, 17
f^*	Predictions for the objective f.	6, 48, 172
$\hat{f}_\mu$	The GP regression's mean prediction for the objective f based on the sampled points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$. $\hat{f}_\mu\colon X \to \mathbb{R}$	28, 48, 52, 53, 63–65, 67, 68, 98, 100, 102, 103
$\hat{f}_{\sigma^2}$	The GP regression's predictive variance for the objective f based on the sampled points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$. $\hat{f}_{\sigma^2}\colon X \to \mathbb{R}^+$	28, 52, 53, 65, 98
p_Y^x	The GP regression's predictive distribution for the value of the objective f at x based on the sampled points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$. In equations this may refer to the distribution or the PDF of the distribution in particular. $p_Y^x \sim \mathcal{N}(\hat{f}_\mu(x), \hat{f}_{\sigma^2}(x))$	21–23, 29, 30, 48, 62, 65, 100, 147
$p_Y^{\mathbf{x}}$	The GP regression's predictive distribution for the value of the objective f at a set of points $\mathbf{x}$ based on the sampled points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$. $p_Y^{\mathbf{x}}$ is a multivariate normal distribution.	48
$\hat{f}$	The GP regression of the objective f based on the sampled points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$. This is used as a general descriptive term for the surrogate function, and in formulas the more precise terms such as p_Y^x will be used.	XI, XII, 19, 21, 22, 27–30, 48–50, 52, 53, 59, 60, 63–69, 79, 80, 96–98, 100, 102, 103, 105, 114, 144, 145, 147, 151

Notation	Description	Page List
$\hat{f}^x$	The random variable defined by p^x_Y.	28, 29, 65–67, 69, 79, 147
$p^x_{\mathbf{Y}}$	In multiobjective problems, the joint predictive distribution for the value of the objective $\mathbf{f}$ (based on the GP regression for each individual objective) at x, based on the sampled points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$.	80
$\hat{\mathbf{f}}^x$	The multivariate random variable defined by $p^x_{\mathbf{Y}}$.	79, 80
x^t	A point at which to sample a GP.	11, 12, 100–103
Y^t	The random variable in a GP associated with x^t.	11
$\mathbf{x}^t$	A set of points in X at which to sample a GP.	11–14, 19
$\mathbf{Y}^t$	The set of random variables in a GP corresponding to $\mathbf{x}^t$.	11–14
X	Parameter space (domain) of f. For environmentally-adaptive optimization (Chapter 6), $X_c \times X_e = X$.	6, 7, 10, 11, 13, 14, 26–28, 48, 89, 90, 96, 100, 101, 103, 143, 144, 146–148
f	Objective function to be maximized. $f\colon X \to \mathbb{R}$	XI–XIII, 12–14, 16, 18, 19, 26, 27, 48, 49, 58, 62, 63, 65, 68, 77, 78, 90, 91, 94–101, 104, 106, 113, 144, 166
x_r	After the experiment selection phase of an optimization is complete, an algorithm uses the available information to recommend the point x_r as the best prediction for $\operatorname{argmax}_{x\in X} f(x)$ (or in the binary case, for $\operatorname{argmax}_{x\in X} \pi(x)$)	143, 144, 150
$y_{\max}$	The objective evaluation with the highest value; $\max \tilde{\mathbf{y}}$ for deterministic f. For stochastic f, see $Y_{\max}$.	28–30, 62–67, 69, 145, 146, 155
$Y_{\max}$	A distribution over the estimated maximum objective evaluation; analogous to $y_{\max}$ for stochastic f.	65–67, 69
$\tilde{\mathbf{y}}$	A set of sampled values of the objective f, corresponding to the points $\tilde{\mathbf{x}}$	XI–XIII, 16–19, 21, 22, 49–53, 59, 60, 62–66, 79, 80, 100, 101, 172
$\tilde{\mathbf{x}}$	A set of points in X at which the objective f has been sampled to obtain $\tilde{\mathbf{y}}$	XI–XIII, 16–19, 21, 22, 49–53, 59, 60, 63–67, 78, 143, 145, 146, 148, 172
x_{opt}	Optimal input parameter in X for f, or $\operatorname{argmax}_X f(x)$.	48, 49

Notation	Description	Page List
$\hat{x}_{opt}$	Prediction for the optimal input parameter x_{opt}, usually based on the GP fit for $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$.	49
$\mathbf{f}$	The vector-valued function of multi-objective outputs; $\mathbf{f}\colon X \to \mathbb{R}^n$. $\mathbf{f}(x) = \{f_1(x), f_2(x), \ldots, f_n(x)\}$	78, 80
γ	For environmentally-adaptive optimization (Chapter 6), a policy γ defines which control parameters to select for each environment. $\gamma\colon X_e \to X_c$	94–96, 98, 100–104, 113
γ^*	The optimal policy γ^* is the policy which maximizes the policy score $\mathcal{S}$.	95, 96
$\mathcal{S}$	The score of a policy γ, $\mathcal{S}(\gamma)$ is a measure of its (weighted) combined performance over all environments.	94–96, 98, 101, 102
X_c	For environmentally-adaptive optimization (Chapter 6), the control subspace of X. $X_c \subset X$	93, 94, 96, 98, 100, 106
x_c	A single control setting; a point within X_c. $x_c \in X_c$	93, 96, 100, 101, 111, 113, 166
X_e	For environmentally-adaptive optimization (Chapter 6), the environment subspace of X. $X_e \subset X$	94–96, 98, 100, 102, 106
x_e	Parameters that define a single environment; a point within X_e. $x_e \in X_e$	94–96, 98, 100–102, 111, 113, 166
$\hat{\pi}_{max}$	The maximum of the predicted underlying $\hat{\pi}$ at the sampled points $\tilde{\mathbf{x}}$	146, 147
$\hat{\pi}$	In the binary stochastic setting (Chapter 8), $\hat{\pi}$ is the estimate of π. This is analogous to $\hat{f}$ in the standard regression case, and is represented by a GPC. However, the joint distributions defined by this surrogate are not Gaussian (indeed, they are not in general symmetric).	21, 144, 145, 147, 150, 151, 154
$\bar{\pi}$	In the binary stochastic setting, the expectation of p_π^x, or $\bar{\pi}(x) = \mathbb{E}[p_\pi^x]$. $\bar{\pi}\colon X \to (0,1)$	23, 145–147, 155
p_π^x	In the binary stochastic setting, the GPC regression's posterior predictive distribution for the value of underlying success probability π at x based on the sampled points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$. In equations this may refer to the distribution or the PDF of the distribution in particular.	21–23, 145, 147, 150, 155
σ	When using a classification setting of Gaussian processes (GPs), σ is the response function that converts between linear and probit or logistic regression. $\sigma\colon (-\infty, \infty) \to (0,1)$	21, 22, 144, 145, 147
π	In the binary stochastic setting (Chapter 8), π is the unknown underlying probability of success. $\pi\colon X \to [0,1]$	22, 143–154, 156, 157

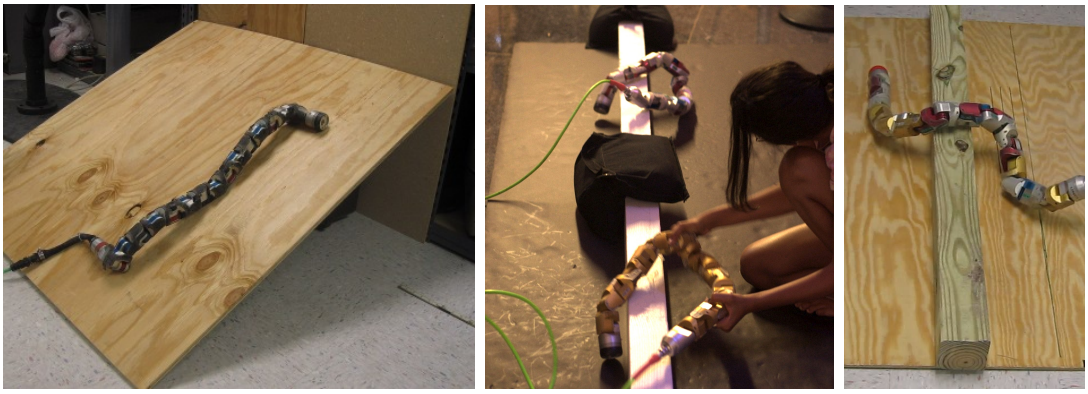
Chapter 1

Introduction

The modular snake robots in Carnegie Mellon's Biorobotics lab provide an intriguing platform for research: they have already been shown to excel at a variety of locomotive tasks and have incredible potential for navigating complex terrains, but much of that potential remains untapped.

The motivation to expand the capabilities of these robots stems from experiencing several failures and limitations in real world tests. For example, this robot was able to navigate a narrow passage underneath a rubble pile at a disaster response training site, but was unable to pass over a four inch high piece of wood which lay across its path once the passage widened. In an archaeological expedition near the Red Sea, the robot was able to move further than a human could into a collapsed cave containing four-millenia-old ship timbers. However, a gradual sandy slope prevented the robot from moving further and potentially making an archaeological discovery.

To extend the capabilities and learn improved controllers for these robots, a model must be found on which to evaluate the controller. Unfortunately, analytic snake robot models have proven overly complex due to the robots' complex, multi-modal locomotion dynamics, and after pages of formulae these models only provide qualitatively accurate results.



(a) (b) (c)

Figure 1.1: This thesis develops learning algorithms for robotics applications; this includes **(a)** optimization of motion in steady-state settings, **(b)** learning from demonstration, and **(c)** overcoming obstacles with limited feedback.

Numerical simulation models evaluated through computer-based simulation engines provide a simpler approach. However, the accuracy of these models begins to break down as the frequency of collisions increases, and constraint violations due to numerical issues can cause meaningless results.

The only course remaining is to learn through evaluations on the physical robot itself. This is an *expensive* optimization problem due to the relatively limited experimental budget when working with a physical system as compared to a computer model. In this thesis I develop machine learning techniques which address the challenges associated with such optimization problems. This work extends research from other fields, simultaneously advancing the robotics literature as well as those fields from which the methods originate.

This thesis seeks to add low-level autonomy to improve robot capabilities and simplify the task for the operator, rather than providing complete autonomous control. In the desired applications for the snake robot system which motivate this work (archaeology, search and rescue), a human is crucial for analyzing the scene for objects of interest or potential victims and providing high level directional control. By lowering the cognitive workload necessary to control the robot and understand the scene, I hope to allow the operator to place more

focus on the task rather than the tool, improving mission performance. However, elements of this work could still be used for full autonomy in other scenarios.

Although the methods developed herein are primarily designed for improving snake robot teleoperative performance, they are applicable to a wider variety of problems with similar underlying challenges. By addressing problems faced by the current robot with algorithmic and learning advances rather than hardware development, I motivate the following more general theoretic problems (Fig. 1.1):

1. Optimization of *noisy*, *expensive*, *black-box* functions, with extensions for *multiple objectives* and learning *multiple related tasks*.

2. Collection and generalization of *demonstrated input* for *underactuated systems*.

3. Optimization of expensive functions with *stochastic binary* outcomes.

Below, I outline the particular contributions of this work.

Improving Gaits

Control of snake robots is difficult in part because of the large number of degrees of freedom in the system. One method of reducing the dimensionality to a more manageable level is through coupling the control of these individual actuators by using *gaits*. A gait is a cyclic motion in the robot's shape space which produces some (possibly trivial) motion of the robot in the world. Gaits are the foundation of snake robot locomotion; their effective overall motion is easily characterized, and controllability proofs show that cyclic motion is optimal for locomotion over large distances [14].

Empirically, gaits have produced impressive results when moving over uniform terrain, including pipes, or terrain with low amplitude disturbances. However, we have no reason

to believe that our current set of gaits is optimal or complete; in addition, we know these existing gaits cannot handle all of the challenges that the robot has faced.

The gait optimization work in this thesis uses a surrogate function to create a model of gait performance as a function of gait parameters. Applying this method enables the robots to move more robustly over a set of small obstacles, and to climb steeper slopes than previously possible. I describe improvements I add to existing surrogate function optimization approaches to enable the optimization to run robustly, without finicky tuning parameters. Also, as stochasticity exists in physical systems being run in the real world, I describe several approaches to explicitly take into account this noise and show the effects of these methods.

As robotic systems often have multiple conflicting notions of optimality, I describe a multi-objective formulation of the optimization setting above to generate Pareto optimal sets of gait parameters; these sets contain optimal parameter choices for multiple simultaneous (and potentially conflicting) objectives, such as speed and robustness.

I also extend the basic optimization setting to include parameters describing the environment or task, such as slope or crevice width. This extension seeks a control parameter policy over these environment parameters, as opposed to a single optimal gait parameter setting. The result has been demonstrated in a simple autonomous behavior by incorporating state estimation that allows the system to measure and react to its environment.

Learning and Optimizing Non-Gait Shape Changes

Although gaits are the motion of choice for traversing long distances over uniform terrain, more interesting environments will rarely be uniform for significant lengths. Instead, complex motions must be learned that enable navigation over complex terrain and large obstacles.

To address this challenge, I describe an approach to record, simplify, and parameterize demonstrated trajectories from expert and novice users. As the settings where these motions

are needed usually measure the result of the motion in terms of success and failure rather than a numerical score, I derive extensions to the optimization framework used for improving gaits to handle stochastic binary functions, and use this to optimize robustness of trajectories for moving over obstacles.

Chapter 2

Background

2.1 Gaussian Processes for Regression

In this thesis, Gaussian processes (GPs) are used extensively as a regression method. The following section provides a brief overview of GPs, the notation used in this thesis, and their application to regression.

Intuitively, a GP can be viewed as a probability distribution over functions. Given a domain X (referred to as the *parameter space*), a GP assigns each function $f^*: X \to \mathbb{R}$ a likelihood. The GP can then be used to sample functions from this implied distribution, to condition the function distribution on observations, and to provide predictive output distributions for given points $x \in X$.

We will reach a more precise definition of GPs after first describing the intuition for two key concepts, the mean and covariance functions associated with a GP.

2.1.1 Gaussian Process Priors

As stated above, a GP is a distribution over functions which map X to $\mathbb{R}$. The GP's *prior distribution* over this infinite set of mappings is completely described by a mean function

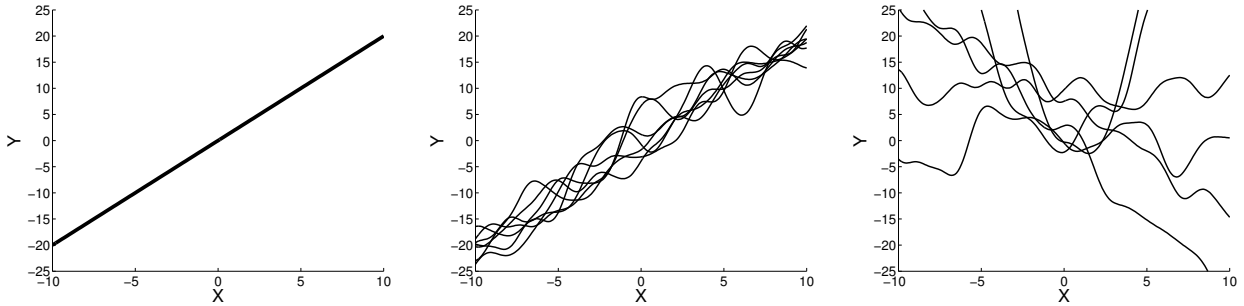
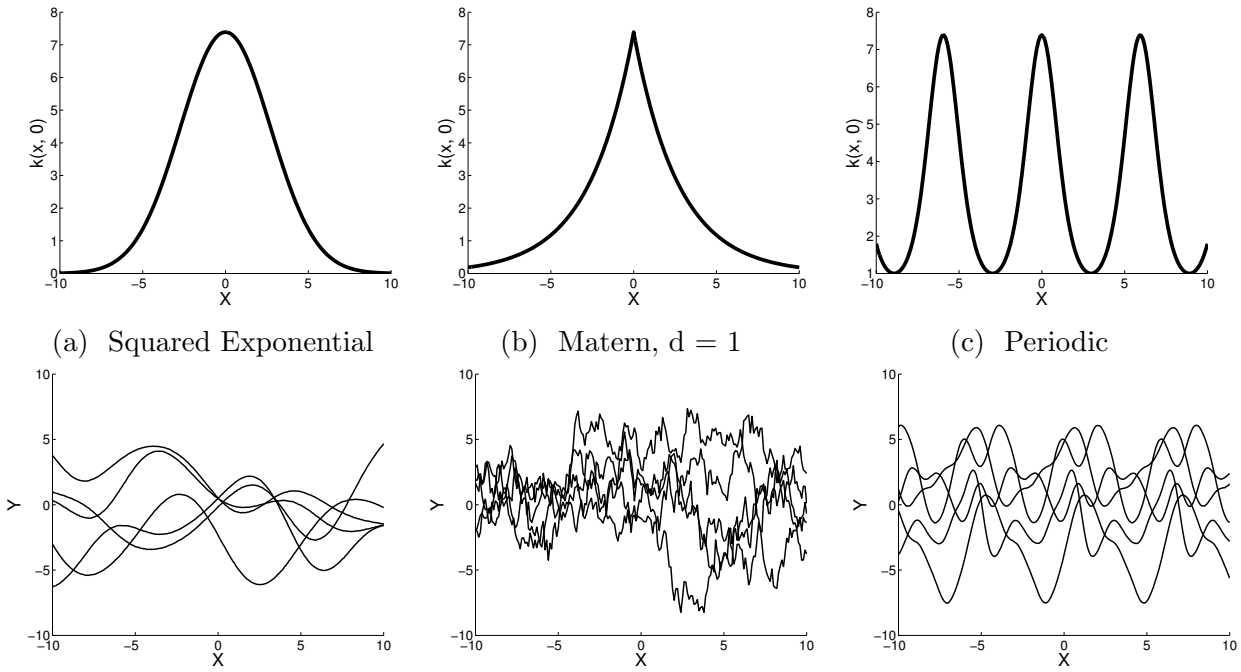


Figure 2.1: The mean function describes the expected function value. If the mean function is a simple linear function such as **(a)**, then functions such as **(b)** have higher likelihoods than those given in **(c)**, because they are closer to the expected value.

$m\colon X \to \mathbb{R}$ and a positive definite covariance function $k\colon X \times X \to \mathbb{R}$.

Mean function: The mean function gives the overall trend of typical functions in the GP's distribution (Fig. 2.1). When a GP is used to estimate a function, such as the regression applications in this thesis, the mean function can be used to encode prior knowledge such as linear trend, a constant bias term, or a step response at some location in the space. Often in the literature, it is assumed to be identically zero at all points in the space.

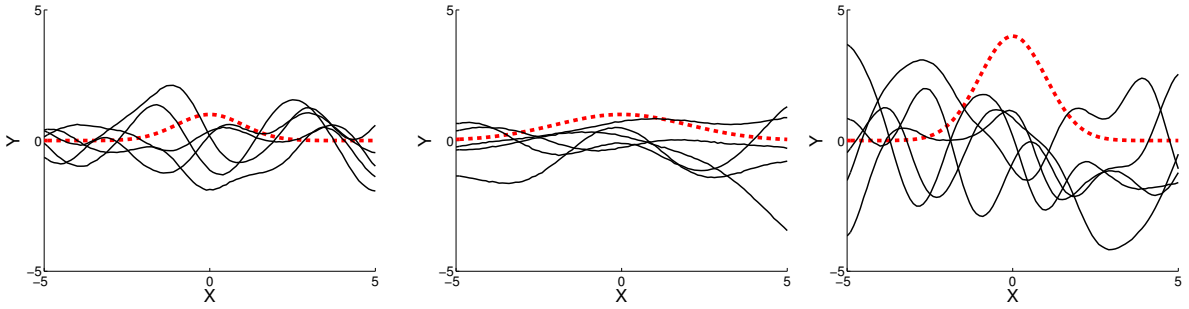
Covariance function: The covariance function describes other properties of functions in the distribution, such as smoothness and scale (Fig. 2.2). This is also used to encode prior knowledge, but the effect on function behavior is not as directly intuitive. More accurately, the covariance function describes how related different points in X are, or how the expected function values are correlated, or *co-vary*, at these points. For example, if typical functions in the GP's distribution change very quickly, then the covariance function drops off quickly as x and x' diverge; if typical functions change less quickly, then this results in a covariance function with a slower dropoff. This can even describe periodic behavior, by enforcing high positive covariance between points spaced equal distances apart.



(a) Squared Exponential (b) Matern, d = 1 (c) Periodic

(d) Sampled Functions (e) Sampled Functions (f) Sampled Functions

Figure 2.2: The covariance function describes how points in the space are correlated. Each figure in the top row shows the covariance function output $k(x, x')$, where x' is set to zero and x is allowed to vary. Higher values indicate higher correlation between function values at these points. Each bottom row figure shows samples from a GP prior with a zero mean function and the covariance function pictured above it.



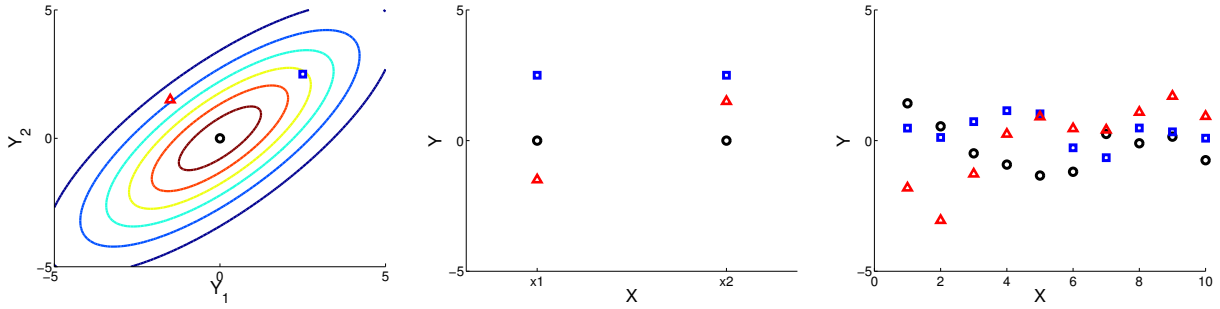
(a) $\ell = 1$ $\sigma_f^2 = 1$ (b) $\ell = 2$ $\sigma_f^2 = 1$ (c) $\ell = 1$ $\sigma_f^2 = 2$

Figure 2.3: The black solid lines are samples drawn from GP priors with a squared exponential covariance function and three different hyperparameter values. The red dashed line is the covariance function $k(x, x')$ with $x' = 0$. **(a)** The length scale parameter is set to 1, as is the signal variance. **(b)** When the length scale increases, typical function samples are smoother and vary less quickly – values are more correlated across the spatial dimension. **(c)** When the signal variance increases, the amplitude of typical function samples increases.

Hyperparameters: Covariance functions are often parameterized so that they can describe an entire family of function behaviors. The overall characteristics are the same, but details of the resulting distribution – typical amplitudes, rate of changes, etc – may change. For example, the *squared exponential* covariance function,

$$k(x, x') = \sigma_f^2 \exp(-\frac{(x - x')^2}{2\ell^2}), \tag{2.1}$$

describes functions which generally vary smoothly. It has a length scale parameter ℓ and a signal variance parameter σ_f^2. As ℓ decreases, typical functions drawn from the GP's distribution vary more quickly. As σ_f^2 increases, functions drawn from the distribution are more likely to have a larger scale, or range of output values. As these parameters affect the prior distribution of functions rather than directly act as parameters of a single function, they are referred to as *hyperparameters*. Sampled functions from GPs with the same covariance function family but different hyperparameters are shown in Fig. 2.3.



(a) (b) (c)

Figure 2.4: **(a)** A bivariate Gaussian PDF with three points selected. **(b)** Each of the random variables from (a) is associated with a point in X. The three points shown in (a) are shown as sets of points here, including the circles which represent the mean, or most likely values. **(c)** As more points are added, and the multivariate Gaussian becomes harder to visualize, each sample from that Gaussian approximates a function from X to Y; this is shown here with 10 points.

2.1.2 Gaussian Process Definition

The discussion above has provided a basic level of intuition on GPs – in particular, they are a distribution over functions from X to $\mathbb{R}$, and are described by a mean function and a covariance function. Here we provide a formal definition, and describe the basic equations and notation needed to work with GPs.

Now consider a joint Gaussian probability distribution between two random variables, Y_1 and Y_2. As shown in Fig. 2.4a and 2.4b, we associate Y_1 with $x_1 \in X$ and Y_2 with $x_2 \in X$. This multivariate distribution describes the relation of Y_1 and Y_2; as shown in the figure, certain pairs of values are more or less likely, and the most likely set of values represents the mean of the Gaussian probability density function (PDF).

This is simple to extend to more points, although the dimension of the Gaussian PDF because too large to visualize. In Fig. 2.4c, we shown several samples from a 10-dimensional Gaussian PDF, again with each random variable Y_i associated with a point $x_i \in X$. Again, the mean of this distribution gives the most likely set of joint random values.

A GP simply extends the above concept to an infinite[1] number of points. In fact, the formal definition of a GP, taken from page 13 of [81], is

Definition (GP)**.** A GP is a collection of random variables, any finite number of which have a [joint] Gaussian distribution.

To equate this definition with our understanding of a GP as a distribution over functions, we add the notion of an *index set* X. Each of the random variables Y_i in a GP's collection is the distribution of function values at some point $x_i \in X$. A subset of these random variables, $\{Y_1, Y_2, \ldots, Y_k\}$, is associated with the joint distribution of function values at a set of points $\{x_1, x_2, \ldots, x_k\}$. The example from Fig. 2.4c might then be a subset of 10 of a GP's random variables.

2.1.3 Sampling from the Prior

As we initially stated, a GP can be thought of as a prior over functions. Suppose we wish to sample from this distribution. If $X \subset \mathbb{R}^n$, then it contains an infinite number of points; the definition of a GP only guarantees a joint distribution for a finite number of points[2]. Fortunately, for practical purposes, it is sufficient to sample at a finite but potentially large number of points. We consider a set of test points $\mathbf{x}^t = \{x_1^t, x_2^t, \ldots, x_k^t\}$ in the domain of the function at which to jointly sample the GP to draw from this function distribution. The collection of random variables in the GP corresponding to $\mathbf{x}^t$ is $\mathbf{Y}^t = \{Y_1^t, Y_2^t, \ldots, Y_k^t\}$.

To form the multivariate Gaussian distribution which the GP describes at these points, we must define the mean of the Gaussian as well as its covariance matrix. Recall that the GP is completely described by a mean function m and a covariance function k. The joint

[1] Technically, a GP need not be over an infinite set, but is simply a multivariate Gaussian distribution if not.

[2] This is reasonable – as the number of dimensions of a Gaussian approaches infinity, the value of the PDF approaches zero everywhere; the value is undefined when the PDF is over an infinite dimensional space.

prior distribution of function values at $\mathbf{x}^t$ is given as

$$\mathbf{Y}^t|\mathbf{x}^t \sim \mathcal{N}\left(\begin{bmatrix} m(x_1^t) \\ m(x_2^t) \\ \vdots \\ m(x_k^t) \end{bmatrix}, \begin{bmatrix} k(x_1^t, x_1^t) & k(x_1^t, x_2^t) & \cdots & k(x_1^t, x_k^t) \\ k(x_2^t, x_1^t) & k(x_2^t, x_2^t) & \cdots & k(x_2^t, x_k^t) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_k^t, x_1^t) & k(x_k^t, x_2^t) & \cdots & k(x_k^t, x_k^t) \end{bmatrix}\right), \tag{2.2}$$

or in a shorthand notation

$$\mathbf{Y}^t|\mathbf{x}^t \sim \mathcal{N}\left(M(\mathbf{x}^t), K(\mathbf{x}^t, \mathbf{x}^t)\right). \tag{2.3}$$

Recall that when the idea of the covariance function k was introduced above, it was called out that k must be positive definite, whereas no restriction was made on the mean function. The reason for this is now apparent – the covariance function defines the entries of a Gaussian distribution's covariance matrix, and as this matrix must be positive definite, the function which produces it also must be.

Finally, we note that (a) the view of a GP as a collection of random variables in Def. 2.1.2 and (b) the more casual statement that a GP is completely described by a mean and covariance function can be related as follows. Given a mean and covariance function, one can define a Gaussian PDF for any finite test set $\mathbf{x}^t$; similarly, given a collection of random variables, one can use the definitions of expectation and covariance to obtain the values of the mean and covariance functions evaluated from any finite set of those random variables.

2.1.4 Regression and the Gaussian Process Posterior

In regression, the goal is to fit a model to data from an unknown function f. As GPs provide a distribution over functions, they work ideally as such a model, producing a most likely estimate of the underlying function (the mean) as well as pointwise variances (the covariance

function evaluated at each point independently) to represent uncertainty. However, to use GPs in this role, we cannot simply use the prior distribution described above – we must be able to *condition* on the sampled data from f to obtain a *posterior* distribution over functions.

Computing this posterior is possible because the GP provides a joint Gaussian distribution for any finite set of random variables $\mathbf{Y}^t$, and conditioning Gaussian distributions is well understood. We first restate this general result, and then apply it to the derivation of a GP posterior.

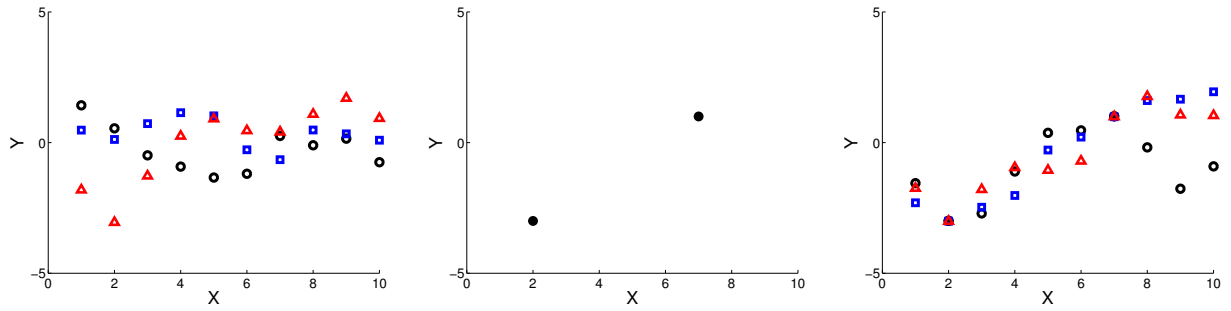
For a set of $m+n$ random variables divided into two groups $\mathbf{Z_A} = [Z_1, Z_2, \ldots, Z_m]^T$ and $\mathbf{Z_B} = [Z_{m+1}, Z_{m+2}, \ldots, Z_{m+n}]^T$, suppose their joint distribution is given as

$$\begin{bmatrix} \mathbf{Z_A} \\ \mathbf{Z_B} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix}, \begin{bmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{bmatrix} \right). \tag{2.4}$$

If the true values of the variables in $\mathbf{Z_B}$ are observed to be z, we can update the distribution of the values of $\mathbf{Z_A}$ with this information, giving

$$\begin{aligned} (\mathbf{Z_A}|\mathbf{Z_B} = z) &\sim \mathcal{N}\left(\bar{\mu}, \bar{\Sigma}\right) \qquad \text{where} \\ \bar{\mu} &= \mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(z - \mu_B), \\ \bar{\Sigma} &= \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA}. \end{aligned} \tag{2.5}$$

Similarly, if we have two sets of points in X, $\mathbf{x}^t$ at which we wish to predict the function and $\mathbf{x}$ at which we will sample the true objective f, the prior distribution over the joint set is



(a) (b) (c)

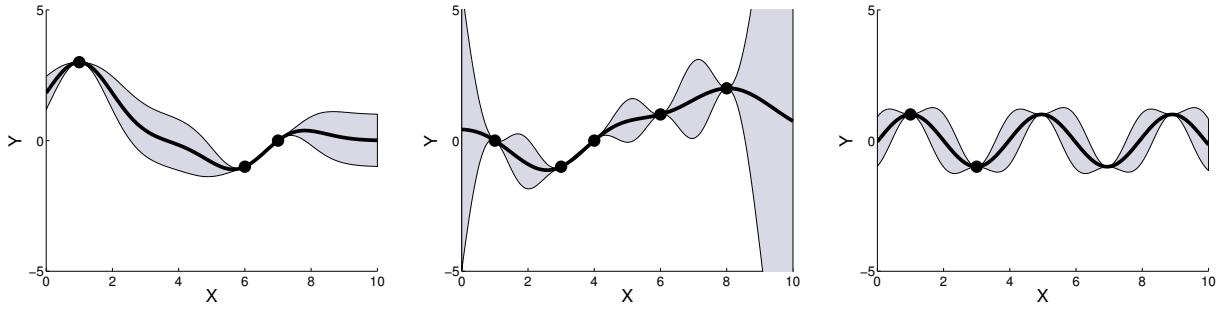
Figure 2.5: **(a)** A GP with a squared exponential covariance function and zero mean is used to generate a multivariate Gaussian over random variables associated with points $\{1, 2, \ldots, 10\} \subset X$. Three samples are drawn from this distribution, and plotted as the black circles, red triangles, and blue squares. **(b)** Two points are chosen in $X \times Y$ (perhaps evaluations of some unknown function). **(c)** The GP from a is conditioned on these points, and three new samples are drawn from the posterior. Notice that the samples all interpolate these points; the posterior distribution's variance has collapsed to zero in these dimensions.

$$\begin{bmatrix} \mathbf{Y}^t \\ \mathbf{Y} \end{bmatrix} \mid \mathbf{x}^t, \mathbf{x} \sim \mathcal{N}\left(\begin{bmatrix} M(\mathbf{x}^t) \\ M(\mathbf{x}) \end{bmatrix}, \begin{bmatrix} K(\mathbf{x}^t, \mathbf{x}^t) & K(\mathbf{x}^t, \mathbf{x}) \\ K(\mathbf{x}, \mathbf{x}^t) & K(\mathbf{x}, \mathbf{x}) \end{bmatrix} \right), \tag{2.6}$$

and by using Equation (2.5) the posterior distribution is

$$\begin{aligned} (\mathbf{Y}^t | \mathbf{x}^t, \mathbf{x}, \mathbf{Y} = \mathbf{y}) &\sim \mathcal{N}\left(\bar{\mu}, \bar{\Sigma}\right), \qquad \text{where} \\ \bar{\mu} &= M(\mathbf{x}^t) + K(\mathbf{x}^t, \mathbf{x}) K(\mathbf{x}, \mathbf{x})^{-1} (\mathbf{y} - M(\mathbf{x})), \\ \bar{\Sigma} &= K(\mathbf{x}^t, \mathbf{x}^t) - K(\mathbf{x}^t, \mathbf{x}) K(\mathbf{x}, \mathbf{x})^{-1} K(\mathbf{x}, \mathbf{x}^t). \end{aligned} \tag{2.7}$$

In Fig. 2.5, this process is demonstrated. A number of samples from the GP prior are drawn at a test set of 10 points. After evaluating two points from the true function f, a number of samples is drawn from the posterior obtained by conditioning on these evaluated points.



(a) (b) (c)

Figure 2.6: A useful, albeit incomplete, visualization of a GP is obtained by plotting the mean function along with a shaded confidence region of the pointwise standard deviations. **(a)** Posterior from a GP with a squared exponential covariance and hyperparameters $\ell = 1$ and $\sigma_f^2 = 1$. **(b)** Squared exponential covariance with $\ell = 1.4$ and $\sigma_f^2 = 4$. **(c)** Periodic covariance with $\ell = 2$, $\lambda = 5$, and $\sigma_f^2 2$.

2.1.5 Gaussian process Posterior Visualization

Roughly, a GP can either be thought of as (a) an infinite-dimensional Gaussian, or (b) a distribution over functions. Unfortunately, neither of these permits a complete yet simple visualization. One approach might be to draw several samples, as in previous figures from this chapter; this can quickly result in cluttered and hard to understand images. A more common, straightforward technique used to form a partial visualization of the GP and its posterior is a solid line with a shaded 'confidence region' around it (see Fig. 2.6).

The solid line in the center is the mean of the GP function distribution. In the case of a GP prior, this is just the mean function; in the case of a posterior, it is the mean of the posterior distribution discussed above. The gray shaded region is ± 1 standard deviation of the individual Gaussian predictive distribution at each point.

The information that is missing in this representation is any detail about the joint distribution – how points in this space co-vary. In other words, this picture does not describe the full function distribution of the GP; for example, the picture would be identical for two GP priors using a squared exponential covariance with different length scale hyperparameters.

Even with this limitation, these images provide reasonable intuition about the function

distribution. and will be heavily used in the following chapters. Also, for regression applications, this visualization captures key information about the prediction: the most likely function (the mean) and the uncertainty in this estimate at various points (the pointwise variances).

2.1.6 GP Model Selection

As mentioned above, the mean and covariance functions and corresponding hyperparameters determine the GP's prior distribution over functions; therefore, they also affect the posterior distribution. When using GPs for regression of an unknown function f, a central question is then which functions and hyperparameters to choose?

First, and most importantly, this choice should reflect prior information. Do you know that f is periodic? Is there an expected linear trend in f? Adding such information will focus the resulting predictive function distribution more closely around f.

There are also principled probabilistic techniques one can use to make this selection. Assume you have evaluated f at several points $\tilde{\mathbf{x}} = \{x_1, x_2, \ldots, x_n\}$ to obtain $\tilde{\mathbf{y}} = \{y_1, y_2, \ldots, y_n\}$. The idea behind the probabilistic selection approach is relatively simple: determine which model (hyperparameters, covariance function, etc.) make the evaluated $\tilde{\mathbf{y}}$ most likely. Alternatively, this can be thought of as finding which model most likely could have generated the data $\{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}\}$. Intuitively, if the data clearly shows a sharp linear trend, then this data is more likely to have come from a GP with a similar linear mean function than one with a zero mean function. If the data is periodic with a period of 1, it is more likely to come from a period covariance function with a period of 1 than one with a period of 1.5.

To compute this *marginal likelihood*[3] of the data, assume you have a GP and associated hyperparameters. This defines a Gaussian prior $\mathbf{Y}|\tilde{\mathbf{x}}$ over the random variables

[3]The term *marginal* refers to marginalizing out possible function values; a more detailed derivation using this intuition is given in Appendix C.1.

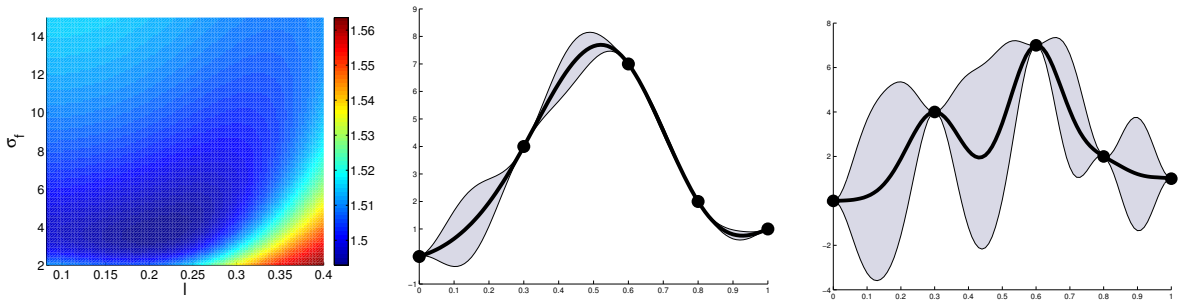


Figure 2.7: **(a)** A region of the likelihood surface for the two hyperparameters of a squared exponential covariance, ℓ and σ_f, which contains the maximum likelihood estimate. For numerical optimization reasons, the negative log likelihood is usually considered; this figure is also shown with arctan scaling to make the contours more apparent. **(b)** The GP posterior using the maximum likelihood hyperparameters, $\ell = 0.21$ and $\sigma_f^2 = 3.6$. **(c)** The GP posterior using a set of non-maximum likelihood hyperparameters, $\ell = 0.082$ and $\sigma_f^2 = 2.1$.

$\mathbf{Y} = \{Y_1, Y_2, \ldots, Y_n\}$ associated with $\tilde{\mathbf{x}}$, as given by Equation (2.3). Using the standard equation for a multivariate Gaussian PDF, the value of this PDF at $\tilde{\mathbf{y}}$ directly gives the likelihood that $\tilde{\mathbf{y}}$ could have been drawn from this distribution,

$$p(\mathbf{Y} = \tilde{\mathbf{y}}|\tilde{\mathbf{x}}) = \Sigma^{-1/2}(2\pi)^{-n/2} e^{(-\frac{1}{2}(\tilde{\mathbf{y}}-\mu)^T \Sigma^{-1}(\tilde{\mathbf{y}}-\mu))}, \tag{2.8}$$

where $\Sigma = K(\tilde{\mathbf{x}}, \tilde{\mathbf{x}})$ and $\mu = M(\tilde{\mathbf{x}})$.

Using the marginal likelihood, one can compare different potential hyperparameters, mean functions, and covariance functions. Finding those which result in the highest likelihood of the data $\tilde{\mathbf{y}}$ is referred to as *maximum likelihood estimation*, and the optimal setting is the maximum likelihood estimate (MLE). This is commonly done to choose the best set of hyperparameters given a fixed mean and covariance function; plotting the likelihood for each value of the hyperparameters results in a *likelihood surface* such as that in Fig. 2.7. Due to the non-linearity and potential non-convexity, finding the optimum of this surface requires careful optimization (discussed more in §4.2).

2.1.7 Fitting GPs to Stochastic Data

Although the regression examples above involved deterministic functions, GPs can also be used in situations where the sampled data is stochastic. Let us assume objective function evaluations can be viewed as realizations of a random noise variable added to an underlying latent function; each sampled value $y_i \in \tilde{\mathbf{y}}$ can be written as $f(x_i) = f_{\text{lat}}(x_i) + \epsilon$ (where x_i is the corresponding element of $\tilde{\mathbf{x}}$ and ϵ is the realization of the random process noise variable).[4]

To include this in the GP model, a diagonal element can be added to the covariance matrix, as is common in the literature [81, 43, 104, 77]:

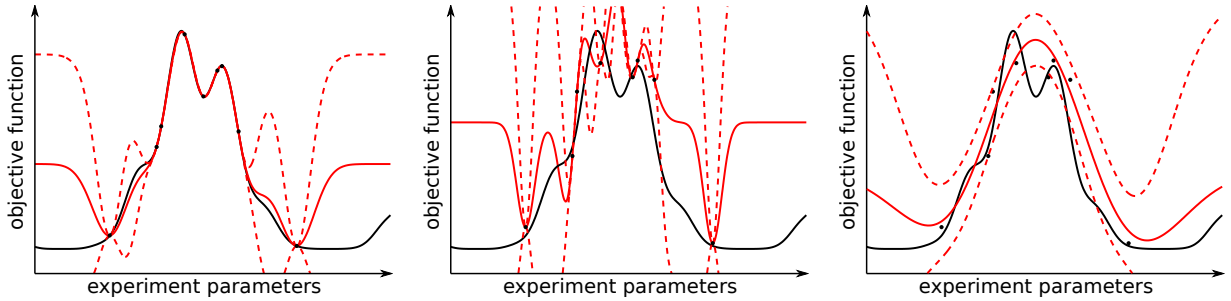
$$k_n(x_i, x_j) = k(x_i, x_j) + \sigma_n^2 \delta_{ij} \tag{2.9}$$

In this equation, $k(x_i, x_j)$ is the (underlying noiseless) covariance function between x_i and x_j, and δ_{ij} is the Kronecker delta function which serves to only add noise to the diagonal element of the resulting covariance matrix. The σ_n^2 term serves as an additional hyperparameter for the GP, and represents the variance of the process noise random variable ϵ. The addition of this term prevents overfitting by relaxing the constraint that the predicted mean value of the GP exactly interpolate the data given, as it would without this term (Fig. 2.8).

Often, the end user of the GP regression (e.g., the optimization algorithm or an online parameter selection method) should decouple the estimated value of the process noise (e.g., the variance of ϵ) and the uncertainty in the estimate of the latent function f_{lat}. This allows for reasoning about which regions of the parameter space are most informative to sample (e.g., the uncertainty in f_{lat}) during optimization (see §4.3), as well as reasoning about risk (e.g., the stochasticity of the system) during online parameter selection.

Note that these terms can be separated during the inference; the uncertainty in the

[4] This follows the approach described in [81], and the equations in this section are adapted from this reference to fit with the notation I use in this thesis.



(a) A GP fit to samples from a deterministic function, using no noise hyperparameter.

(b) A GP fit to samples from a stochastic function, using no noise hyperparameter.

(c) A GP for to samples from a stochastic function, using a noise hyperparameter.

Figure 2.8: Effects of stochasticity in the objective and noise covariance function hyperparameters on GP regression. The black solid line is the function, the red line is the GP posterior mean, and the red dotted lines are one standard deviation uncertainty bounds of the GP predictive distribution. Fitting stochastic data without an explicit noise hyperparameter leads to overfitting (see **(b)**), as the GP attempts to interpolate the observations exactly. **(c)** Using a noise hyperparameter eliminates this overfitting.

estimate of f_{lat} at a set of test points $\mathbf{x}^t$ given the stochastic objective evaluations $\tilde{\mathbf{y}}$ at $\tilde{\mathbf{x}}$ is

$$\mathbb{V}(\hat{f}_{\text{lat}}(\mathbf{x}^t)) = K(\mathbf{x}^t, \mathbf{x}^t) - K(\mathbf{x}^t, \tilde{\mathbf{x}}) \left(K(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}) + \sigma_n^2 I\right)^{-1} K(\tilde{\mathbf{x}}, \mathbf{x}^t). \tag{2.10}$$

The posterior distribution of samples from the stochastic function (including the estimation uncertainty as well as the noise term) is simply the sum of two independent normal distributions. The variance of the sum of independent normal distributions is the sum of the variances, and therefore the variance of this distribution is

$$\mathbb{V}(\hat{f}(\mathbf{x}^t)) = \mathbb{V}(\hat{f}_{\text{lat}}(\mathbf{x}^t)) + \sigma_n^2 I. \tag{2.11}$$

This method of adding a noise term to GP regression assumes independent, uniform Gaussian noise. An active area of research in the community at large is the investigation of heteroscedastic noise models (non uniform noise variance over the parameter space). These often can provide improved accuracy for physical systems, as noise in the objective is likely to be higher in some regions (e.g., faster, more stochastic dynamics); however, the increased

computational penalty during inference is significant. Further, these heteroscedastic models are difficult to train on the small training datasets inherent with the expensive optimization problems discussed in this thesis (c.f. the model complexity discussion in §4.2.2). Because of this difficulty, I feel that the benefits to this thesis of increased noise model accuracy are outweighed by the increased challenges inherent in inference and hyperparameter optimization with these models. The homoscedastic models discussed in this section are used instead when obtaining results, but the methods I describe remain applicable for either type of model.

2.1.8 Gaussian Process Summary

Overall, GPs provide a data-driven regression tool based on a Bayesian treatment of the data. One must understand that although technically a non-parametric technique, GPs do have parameters that must be carefully selected – the mean, covariance, and hyperparameters.

The goal of this section was to give background for the use of GPs for regression in this thesis. I have described the intuition behind GPs, the formal definition, and covered the notation that I use in this thesis. Also, an overview is given of using GPs for the regression of unknown functions, including evaluating posterior distributions, explanation of a common visualization method, and basic model selection.

Finally, I stress that this is only a brief overview necessary for this thesis; more detail and other explanations can be found in several resources, such as [27, 95, 81].

2.2 Gaussian Processes for Classification

A strength of using GPs is the probabilistic modeling of an unknown function. When the observed function outcomes are binary (0 or 1) and stochastic (sampled from an underlying Bernoulli distribution), standard GPs are not appropriate. In this section, I describe Gaus-

sian processes for classification (GPCs), which are an adaptation of GPs to this noisy binary classification setting. Gaussian processes for classification (GPCs) provide a similar probabilistic model for the underlying function in the stochastic binary case, which I elaborate on in Chapter 8. More in-depth coverage of this background material may be found in Chapter 3 of [81].

Adapting GPs for a space of binary response variables uses concepts from linear binary classification. *Linear logistic regression* and *linear probit regression* use the logistic and the probit, respectively, as *response functions* σ to convert a linear model with a range of $(-\infty, \infty)$ to an output that lies within $[0, 1]$ (i.e., a valid probability)[5]. Therefore, given a linear regression model $y = w^T x$, the predicted class probability $\hat{\pi}(x)$ is $\sigma(w^T x)$. The choice of w for the latent linear regression model is typically accomplished via maximizing the likelihood of the data given the model.

Similarly, a GP $\hat{f}$ can generate outputs in the range $(-\infty, \infty)$, and by using a response function σ can convert these outputs to values which can be interpreted as class probabilities. In particular, the latent $\hat{f}$ defines a Gaussian PDF p_Y^x for each $x \in X$ (as well as joint Gaussian PDFs for any set of points in X). We define the corresponding probability density over class probability functions as p_π^x.

Note that although the response function σ maps from the latent space F to the class probability space Π, $p_\pi^x(\bar{y}) \neq p_Y^x(\sigma^{-1}(\bar{y}))$ (where $\bar{y}$ is a class probability in Π, not a 0/1 sample). Instead, due to the change of variables,

$$p_\pi^x(\bar{y}) = p_Y^x(\sigma^{-1}(\bar{y})) \frac{\delta \sigma^{-1}}{\delta \bar{y}}(\bar{y}). \tag{2.12}$$

Finally, because we do not observe values of the latent function, the inference step for conditioning our GP posterior on the sampled observations $\tilde{\mathbf{x}} = \{x_i\}$ and $\tilde{\mathbf{y}} = \{y_i\}$ requires

[5]Later in this thesis, I use the standard normal CDF for σ; however, any monotonically increasing function mapping from $\mathbb{R}$ to the unit interval can be used.

computing the following integral to determine the posterior $\hat{f}$ at x^*:

$$p(\hat{f}^*|\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, x^*) = \int p(\hat{f}^*|\tilde{\mathbf{x}}, x^*, \mathbf{f}^*)p(\mathbf{f}^*|\tilde{\mathbf{x}}, \tilde{\mathbf{y}})d\mathbf{f}^* \tag{2.13}$$

In this equation, $\mathbf{f}^*$ represents the GP prior on the latent function at x^*. Unfortunately, the second term in the integrand represents a non-Gaussian likelihood which makes this integral analytically intractable; approximate inference methods for GP classification rely on approximating this with a Gaussian. Advantages and disadvantages of different approximations are discussed in [75]; I use Minka's expectation propagation (EP) method [69] due to its accuracy and reasonable speed.

Expectation of Posterior on Success Probability

As noted above, $p^x_\pi(\bar{y}) \neq p^x_Y(\sigma^{-1}(\bar{y}))$; therefore, the expectation of the posterior over the success probability, $\mathbb{E}[p^x_\pi]$, is not generally equal to $\sigma(\mathbb{E}[p^x_Y])$. To calculate the former, we use the definition of expectation along with a change-of-variables substitution ($\pi = \sigma \circ f$ and $\bar{y} = \sigma(z)$) to take this integral in the latent space (where approximations for the standard normal CDF can be used):

$$\begin{aligned}
\mathbb{E}[p^x_\pi] &= \int_0^1 \bar{y} p^x_\pi(\bar{y}) d\bar{y} && (2.14)\\
&= \int_0^1 \bar{y} p^x_Y(\sigma^{-1}(\bar{y})) \frac{\delta \sigma^{-1}}{\delta \bar{y}}(\bar{y}) d\bar{y} \\
&= \int_{\sigma^{-1}(0)}^{\sigma^{-1}(1)} \sigma(z) p^x_Y(z) \frac{\delta \sigma^{-1}}{\delta \bar{y}}(\sigma(z)) \frac{\delta \sigma}{\delta z}(z) dz \\
&= \int_{-\infty}^{\infty} \sigma(z) p^x_Y(z) dz && (2.15)
\end{aligned}$$

As noted in section 3.9 of [81], if σ is the Gaussian cumulative density function (CDF)

this can be rewritten as follows (for notational simplicity, we define $\bar{\pi}(x) = \mathbb{E}[p_\pi^x]$ for use later in this thesis):

$$\mathbb{E}[p_\pi^x] = \Phi\left(\frac{\mathbb{E}[p_Y^x]}{\sqrt{1+\mathbb{V}[p_Y^x]}}\right). \tag{2.16}$$

2.3 Snake Robot Control

The work in this thesis was motivated by the goal of improving the locomotive capabilities of Carnegie Mellon's snake robots [107, 108, 48]. There have been many such robots, perhaps many inspired by the pioneering work of Hirose [40]; an in-depth survey of these systems is given by [42]. Even compared to this large body of prior robot design work, the CMU robots have demonstrated impressive locomotive capabilities (see Fig. 2.9). These modular robots are about 5 centimeters in diameter – small enough to fit through a chain link fence – and can range in length from 1 to 2 meters. This size allows the robot to fit into small channels and spaces (such as void spaces in collapsed rubble) too confined for other mechanisms. Much like their biological counterparts they use cyclic control trajectories called *gaits* to move across relatively regular terrain. Examples of regular terrain are a flat expanse of grass, a roughly uniform diameter pole, or a regular grid of poles.

Although the space of cyclic controls is infinite, the CMU robots are usually controlled by motions within a finite dimensional constrained control trajectory subspace (the *gait model* described in [101]):

$$
\begin{aligned}
\alpha(n,t) &= \begin{cases} \beta_{\text{odd}} + A_{\text{odd}}\sin(\xi_{\text{odd}}) & \text{odd} \\ \beta_{\text{even}} + A_{\text{even}}\sin(\xi_{\text{even}} + \eta) & \text{even} \end{cases} \\
\xi_{\text{odd}} &= \psi_{\text{odd}} n + \nu_{\text{odd}} t \\
\xi_{\text{even}} &= \psi_{\text{even}} n + \nu_{\text{even}} t
\end{aligned}
\tag{2.17}
$$

This model is general enough to command the snake to slither, sidewind, roll in an arc, wrap around a tree or pole in a helix and climb, turn in place, and traverse via many other motions. Similar controllers for the non-climbing gaits were discovered in parallel by [31]. Other groups have also found that controlling with sinusoidal joint angle inputs (which drive a discrete serpenoid backbone curve) is effective; among them Kuwada et al. use this strategy for planar motions within pipes [61].

Several researchers have investigated control paradigms that differ from simple cyclic position control, as well. A "backbone curve" approach to biologically-inspired hyper-redundant robot locomotion is given by [19]. This curve can be thought of as the spine of a biological snake. For practical application on a system, a fitting step must be run that is particular to a robot's morphology; [38] and [18] present examples of such fitting methods.



(a) (b) (c)

Figure 2.9: **(a)** Snake robot locomoting along a tree branch. **(b)** Time-lapse snapshots of the sidewinding gait. **(c)** Robot deployed in a mock building collapse.

Liljebäck et al. have designed a coupled follow-the-leader and jam detection/resolution control scheme for obstacle-aided locomotion [63, 64], but it is limited to 2-D motion, and makes several assumptions about the system sensors and environment, requiring motion capture setups that are impractical for use outside of a laboratory setup. Furthermore, their control scheme is designed for infinitely thin segments, requiring further work to improve its general performance on real systems.

Snake robots of the same form factor as CMU's (e.g., [109, 76]) have as of yet had difficulty in overcoming large obstacles. The cyclic gaits mentioned above are not as well suited to overcoming large obstacles or irregular terrain. Other approaches for locomotion over obstacles have focused on different robot form factors ([36, 74]); these systems have different size, complexity, and mobility tradeoffs not addressed in this work.

Instead of relying on pure undulation of the robot's shape for locomotion, [32], [34], and others have investigated the use of external propulsion. This is most often accomplished by the addition of powered wheels or tracks on the exterior of a chain of modules. The control challenges and resulting methods for such robots differ significantly from those considered in this work.

However, in no way have any of the above methods produced motions which were shown to be optimal with respect to any objective. Although all of these methods have some sort of gain, stiffness, amplitude, or other tuning parameters in their control law, they do not answer the question of how to set that parameter in order to produce optimal motion. The optimal value for such a parameter is not immediately clear, and in most cases is not something that can be derived from first principles because of the numerous unmodeled effects that play into any notion of optimality.

Optimality could be claimed by gait generation techniques for simple three-link analytic kinematic systems, such as those techniques proposed by Shammas et al. using height functions inspired by differential geometry [94]; these allow a user to individually visualize the

resulting magnitudes of motion in the x, y, and θ directions, but the underlying methods were flawed due to not accounting for the SE(2) structure of the group: as θ changes, the directions of x and y do as well. These methods were corrected and improved by Hatton et al. in [37], which used connection vector fields to help gait designers visualize the resulting motion, and [39], which reduced the error in x and y by choosing coordinates that bounded values of θ. Currently these planar analytic methods do not scale to higher dimensional complex 3-D systems which do not have analytic models.

2.4 Expensive Optimization

Optimization of the control of physical systems such as snake robots is an example of expensive, black-box optimization. An *expensive* function is one for which evaluations take significant resources (time, money, computation, or other resources). A *black-box* function is one which provides no gradient or derivative information when sampled; it can be treated as a "black box" to which an input is given and an output is returned, but no other information about the inner workings of the function is available. Furthermore, these functions need not have guarantees of convexity or linearity; one must search for a global optimum over a function which is in all likelihood nonlinear and non-convex.

The goal of expensive optimization is to recommend x_r which best approximates x_{best}, the maximizer[6] of the expensive, black-box function $f\colon X \to \mathbb{R}$. This is done through careful sequential selection of points at which to evaluate f, attempting to minimize the number of total evaluations while maximizing the resulting $f(x_r)$. In cases where the evaluations of f are costly (hours to days), computational requirements of the optimization algorithm are not a significant issue; careful choice of the sample is more important than the speed of its selection.

[6]When referencing other work in this field, note that often the goal is to minimize rather than maximize f.

Optimization of functions which fall into this class cannot be accomplished through use of many standard techniques. For example, gradient-ascent approaches would require a number of samples around a sampled point to find an approximation to the gradient; even then, without knowing the basic behavior of the function it would be difficult to pick points that provided an accurate and stable estimate of the gradient.

This has motivated the development of a class of "gradient free" optimization techniques; these include local approaches, such as a Nelder-Mead simplex search (c.f. [73]), and global approaches such as genetic algorithms [7] or simulated annealing [55]. Naturally, a globally optimal controller is preferred to a locally optimal one, but unfortunately most methods which search for such global optima require a large number of function evaluations, which is prohibitive if these evaluations are expensive.

To address this problem, a subset of these techniques is based on the idea of predicting the entire unknown expensive function from limited sampled data. These techniques have different names depending on the field of use, but are commonly termed *response surface* or *Bayesian optimization* methods. They rely on a data-driven probabilistic model $\hat{f}$ (often a GP) as a *surrogate* for the underlying expensive function f. Such algorithms are iterative, sequential experiment selection methods – at each step i, they update the surrogate $\hat{f}$ based on data from previous evaluations of f (Fig. 2.10a), select a next point to evaluate $x_i \in X$, and evaluate x_i on the true function f to obtain y_i. The point x_i is chosen by optimizing a selection metric (sometimes referred to as an infill criterion) at each iteration, which can consider information provided by the surrogate $\hat{f}$ such as predicted function value and model uncertainty at different x_i within X. A comprehensive survey on this subject is given by Jones [49].

In surrogate function approaches to the expensive optimization task, the central challenge is the determination of the *fitness metric*, or how to balance exploration (sampling in unknown regions) and exploitation (sampling in known good regions) during the optimization.

At one extreme, an approach would be to choose the current maximum of the surrogate,

$$x_{\text{next}} = \underset{x \in X}{\operatorname{argmax}}\, \hat{f}_{\mu}. \tag{2.18}$$

Simple examples show that this method is not guaranteed to converge, even to a local optimum. At the opposite extreme, one might choose to reduce uncertainty in the function estimate by selecting the point of the response surface with maximum uncertainty,

$$x_{\text{next}} = \underset{x \in X}{\operatorname{argmax}}\, \hat{f}_{\sigma^2}, \tag{2.19}$$

or the point which maximizes expected information gain (described in [8]). These methods would eventually obtain a very close fit to the true function, and therefore generate a good prediction for the global optimum. However, the goal of optimization is not to have a perfect function fit, but to find the optimum. This method wastes precious function evaluations on improving the function estimate, even in low, uninteresting regions of the objective function that are very unlikely to improve upon the best value found so far.

To improve the quality of search, the uncertainty of the estimated function value should be used in conjunction with that estimated value. To this end, upper confidence bound (UCB) algorithms have been developed (e.g., [3, 23, 71]), which pick a subsequent experiment based on a weighted sum of the estimated function and its uncertainty. Unfortunately, this requires tuning parameters, in particular the balance between exploration and exploitation.

Another approach that incorporates a natural tradeoff is to maximize the *probability of improvement* [60, 105]. Improvement $\mathrm{I}(x)$ refers to the distance above the maximum value of the objective found so far, $y_{\max}$. Given the random variable $\hat{f}^x$ associated with the GP surrogate $\hat{f}$ at x, improvement is defined as (a random variable)

$$\mathrm{I}(x) = \max(\hat{f}^x - y_{\max}, 0). \tag{2.20}$$

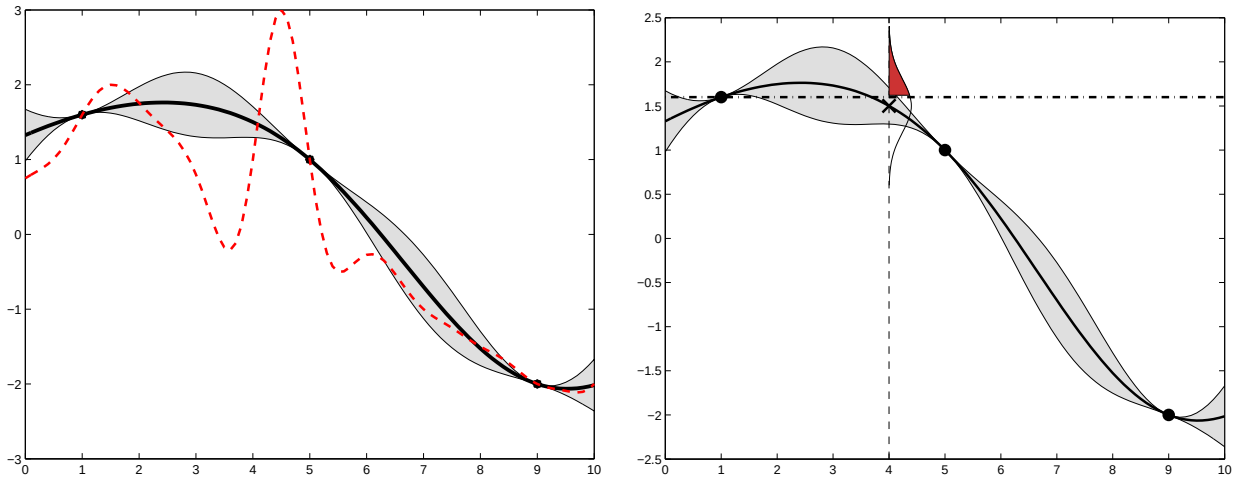


Figure 2.10: **(a)** A surrogate function (dark line) interpolates sampled points of an unknown underlying function (dotted red line). The surrogate quantifies uncertainty in its prediction, as shown by the shaded region. **(b)** Experiment selection metrics such as probability of improvement and expected improvement consider the predictive distribution at a potential sample point x (vertical line), compared to the best previous sample (horizontal line).

The probability of improvement is the probability that this random variable is greater than zero. By considering the PDF associated with $\hat{f}^x$, p_Y^x, this can be calculated as

$$\mathrm{PI}(x) = \int_{y_{\max}}^{\infty} p_Y^x(y)\, dy. \tag{2.21}$$

This can be visualized as the integral of the tail of the predictive distribution above the maximum value of the objective found so far (Fig. 2.10b). However, this metric is biased towards a local search, as the probability of improvement will be maximized near the highest predicted function value. As noted in [49], to encourage exploration one usually measures the probability of improvement above some threshold over the optimal point sampled so far, but this still requires carefully tuning a parameter for best algorithm performance – in this case, the value of the threshold.

A more principled method to address this tradeoff is the idea of expected improvement (EI) [70], popularized by Jones et al.'s *efficient global optimization* (EGO) algorithm [50].

This is a slight alteration of the probability of improvement metric; instead of the area of the tail of the predictive distribution, the center of mass of this tail is computed. This provides a statistical measure that has been shown to effectively balance the trade-off between exploration and exploitation, without requiring an algorithm parameter to be carefully tuned.

Calculating EI requires taking the expectation over improvement from Equation (2.20):

$$\begin{aligned} \mathrm{EI}(x) &= \mathbb{E}[I(x)] && (2.22) \\ \mathrm{EI}(x) &= \int_{-\infty}^{\infty} p_Y^x(y) \max(y - y_{\max}, 0)\, dy && (2.23) \\ &= (\hat{f}_\mu^x - y_{\max}) \left(1 - \Phi((y_{\max} - \hat{f}_\mu^x)/\hat{f}_\sigma^x)\right) && (2.24) \\ &\quad + \hat{f}_\sigma^x \phi((y_{\max} - \hat{f}_\mu^x)/\hat{f}_\sigma^x) \end{aligned}$$

where ϕ and Φ are the PDF and CDF of the standard normal distribution and $\hat{f}_\mu^x$ and $\hat{f}_\sigma^x$ are the mean and standard deviation of p_Y^x.

2.5 Multi-objective Optimization

An important consideration of real-world systems is the existence of competing objectives. For example, faster speed is usually desirable for a locomoting system. Often these systems carry on-board power, and must therefore simultaneously minimize energy usage. These objectives are in direct competition, and so the question arises of how to measure true optimality in such cases; to address this question, one of two ideologies is followed.

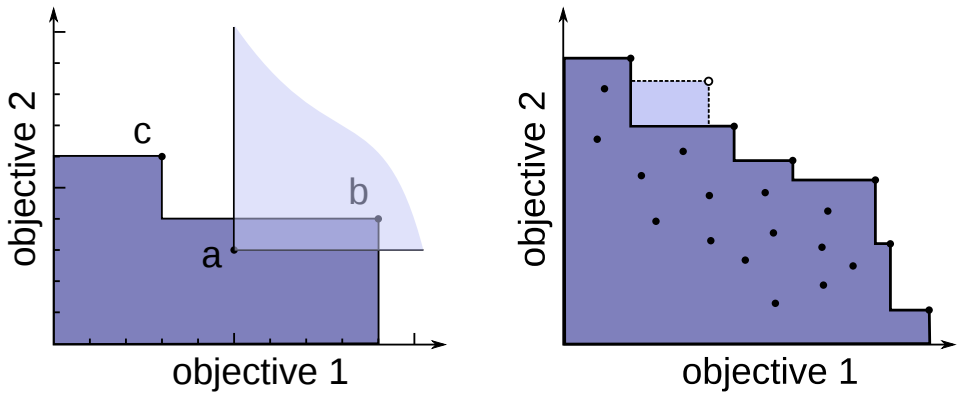
The first, and arguably the more popular in robotics, is the creation of a simple aggregate function that combines (through addition, multiplication, or more complex arithmetic operators) these objectives. This new aggregate is then defined as the "true" objective, and standard single objective optimization techniques are used for its optimization. The for-

mation of the aggregate is ideally informed by some estimate of relative importance by the user, but more practically is an untuned naïve combination. Unfortunately, weights in the resulting formula must be tuned to create a function with the desired behavior, and often this function is brittle to changes in system requirements. To make things worse, there are maximizers of nonlinear aggregate functions that cannot be found with any combination of weights for a simple weighted linear sum aggregate objective.

One example of complications that might arise from use of a simple aggregate function is demonstrated in generating a simple measure of efficiency. One choice for the aggregate objective might be distance traveled, d, divided by energy used, E. To obtain a more numerically stable optimization, d/E might be changed to $d/(E+\epsilon)$, where ϵ is a constant that must be tuned to prevent seeking low-energy, low distance, but high "efficiency" motions. Perhaps the solutions generated are then slower than desired, and a design decision is made to make speed more important. Then the objective is changed to $d^\gamma/(E+\epsilon)$. Generation of complex aggregate objectives commonly results in this increasing number of parameters and adjustment of the objective weights. This is especially costly if you are dealing with expensive-to-evaluate functions, such as physical robot locomotion. Each new objective requires a new optimization (although information could theoretically be reused to reduce the wasted cycles).

The second ideology is that embraced in the field of multi-objective optimization (MOO), which searches for a set of *Pareto optimal* solutions. This set, named after economist Vilfredo Pareto, includes all solutions which cannot be improved in one objective without a corresponding decrease in another. For example, let f_1 and f_2 be objectives that we wish to maximize (each over the same domain), and a, b, and c points in the domain of f_i, as shown in Fig. 2.11a. Clearly, b is preferred to a, as it is at least as good in every objective, and better in at least one. The point b is said to *dominate* a, written $b \succ a$ or $a \prec b$.

Alternatively, note $f_1(b) > f_1(c)$ and $f_2(b) < f_2(c)$. Therefore, we cannot say that either



(a) (b)

Figure 2.11: Objective-space illustrations of a multi-objective optimization problem. Sampled points are solid circles, and the Pareto front (or frontier) is created by connecting Pareto optimal samples; it is illustrated by the bold jagged line. **(a)**: A simple example of Pareto dominance. The dark shaded region shows the area that is *Pareto dominated* by the Pareto optimal set $\{b, c\}$. The light shaded region indicates areas of the space that *Pareto dominate* a. **(b)**: The dark shaded area represents the hypervolume of the currently known Pareto front; the lightly shaded area is the hypervolume increase resulting from a sample at the location of the circle.

of these points is better than the other unless we have defined an explicit relative importance of the objectives. The points b and c are incomparable; this is written $b \sim c$ or (reflexively) $c \sim b$. Considering a set P, the *Pareto optimal* subset of P is $\{p \in P \mid \forall\, q \in P, (p \sim q) \vee (p \succ q)\}$. In the example above, $\{b, c\}$ would be the Pareto optimal subset of $\{a, b, c\}$. For detailed coverage of these ideas, see [21].

This notion of Pareto optimality allows us to define the best set of parameters given no particular relative importance of the objectives. Once knowledge of this optimal set is obtained, it is straightforward to select the best parameters for any given objective tradeoffs or constraints. Finding such optimal sets has been important for a number of real world applications, including modeling grasshopper foraging behavior [89], rehabilitation of water distribution networks [17], design of airfoils [72], and optimization of spacecraft trajectories [22].

In Chapter 5 of this thesis, I adopt this second ideology and focus on expensive opti-

mization of multiple competing objectives. These algorithms search for the Pareto set of solutions rather than a single optimum.

Chapter 3

Related Work

Optimization of robotic systems poses many unique challenges: stochasticity in objective evaluations, consideration of multiple conflicting objectives, adaptation to changing environments, and optimization of tasks which are difficult to score in more detail than a simple true or false. Below, I discuss existing research pertaining to these challenges, focusing on work which considers expensive systems.

3.1 Expensive Optimization

Recall from §2.4 that expensive optimization [12, 49] is the global optimization of functions which are costly to evaluate, restricting the number of available function evaluations; the choice of which point to evaluate is more important than the speed at which a point can be chosen. Many methods use stochastic processes [83, 33, 41] such as GPs as a surrogate for the expensive function, and select the next point to evaluate by maximizing a heuristic such as the UCB, probability of improvement, or EI.

Because these algorithms depend on the surrogate when selecting points for evaluation, the quality of information provided by the surrogate is paramount to the resulting perfor-

mance of the optimization algorithm. However, the focus of existing work is typically on the choice and optimization of the selection heuristic, leaving effective implementation of the underlying regression as an exercise for the reader. Commonly, the assumption is of constant GP covariance function hyperparameters.

3.1.1 Active Learning

The reader may note a superficial similarity of expensive optimization to the field of *active learning* [93], in which the expensive cost of evaluating the true function (or labelling the data) motivates a careful sequential sample selection process in order to learn efficiently. The primary difference is that active learning methods are not seeking an optimum, but an improvement in the quality of the model (often a classifier, as in [52]). In expensive global optimization, the overall quality of the model is irrelevant, especially in suboptimal areas of the search space; rather, the primary concern is the prediction of the optimum. However, many methods in both active learning and expensive global optimization using surrogate functions are related to ideas in optimal experiment design [29].

3.1.2 Bandits

A slang term for a casino slot machine is the "one-armed bandit," referring to the single lever on the side as well as the fact that it will (likely) take your money if used. This has inspired a number of classic problems spanning probability, decision theory, and machine learning, perhaps most famously the *multi-armed bandit* (or k-armed bandit) problem [84]: given a number of slot machines, each with an unknown payout distribution, what is the selection strategy which will maximize the expected payout? Pulling the i^{th} arm returns a reward from its unknown payout distribution; at each timestep an arm must be pulled. This problem has been well studied (c.f., [9]), leading to a number of extensions.

At a basic level, the multi-armed bandit problem is about the tradeoff between exploration and exploitation. Given the current knowledge of the payouts of each of the arms, should the sampling strategy favor the current best arm, or try another arm to learn about other arms which may have better distributions? These problems can be seen almost as a discrete version of the expensive optimization problem described above: which point should be selected to best optimize the unknown function, and which arm should be pulled to maximize the expected reward.

However, there are several differences between expensive optimization and the bandit literature. First, the vast majority of bandit literature is focused on asymptotic bounds on the *cumulative regret*, the sum of each arm that is pulled, rather than the *simple regret* of the expensive optimization problem, in which only the final recommendation reflects the quality of the algorithm. The recent work of [15] begins to investigate bounds on the simple regret as compared to bounds on cumulative regret, but the results in this paper aim to characterize the spaces in which cumulative regret can be minimized rather than the definition of practical algorithms for the simple regret case.

Second, the payouts from arms of a bandit problem are usually assumed to be independent distributions, and therefore information about one arm's payout distribution is not used to learn other arms' payouts. In the expensive optimization problem a central assumption is that the objective function is (at least mostly) continuous; nearby points should have related values. A particularly relevant set of subtopics in the bandit literature that begin to address this is continuous-armed bandits [2, 4, 57] or metric bandits [16]; these have a more similar problem structure to that of expensive optimization. Metric-armed bandits embed the "arms" of the classic multi-arm bandit problem into a metric space, allowing a potentially uncountably infinite number of arms. These arms are often constrained to generate responses via an underlying (often Lipschitz continuous) function.

Finally, in bandit problems, the number of function evaluations is typically much larger,

potentially infinite. Therefore, the modelling of the knowledge of the arm distributions is typically less sophisticated than the surrogate functions used in expensive optimization; bandit algorithms may only store the mean and number of pulls for each arm.

3.1.3 Optimization of Expensive Stochastic Functions

One practical issue encountered when optimizing physical systems in real-world conditions is that these systems are almost never perfectly deterministic – there is some level of process or sensor noise. Although the general global optimization of stochastic functions is fairly well studied (c.f. [112]), there is far less literature on the problem when function evaluations are expensive. The incorporation of this noise into the function regression methods used is straightforward and well documented, but there is not a rigorous extension of current experiment selection metrics to account for noise. An extension of the EI metric described in [43] and further extended in [77] is termed *augmented expected improvement* (AEI) and has some desirable properties, but requires a tuning parameter and is not rigorously founded. Vazquez et al. [104] developed an informational approach relying on Monte-Carlo simulations to estimate entropy reduction; this methods perform comparably to AEI, and both are shown to significantly outperform naïve implementations of EI. However, in practice most algorithms either ignore this noise, which is potentially devastating, or sample a number of times to try to reduce it, which is impractical with expensive systems.

3.2 Environmentally Adaptive Optimization

When working with real robots, there are many situations in which finding a simple optimal parameter is not enough – rather, a policy must be optimized which determines how the robot reacts to different tasks or environments. For example, a slight change in the task – a robot hand grasping a slightly different object, a robot locomoting over a similar terrain,

or an autonomous car seeing a slightly different intersection – should not require another complete optimization, but rather a policy should be learned that can adapt to the full range of expected tasks or environments.

Prior work in multi-task learning postulates that *tabula rasa* learning for multiple similar problems is to be avoided, especially when the task has descriptive features (or parameters). Approaches using a number of techniques have been taken (e.g., [6] suggest neural network predictors for generalizing task knowledge), but perhaps the most relevant is that of [10]; this incorporates the task as additional parameters of a GP used to model the objective. Bonilla et al. take an active learning approach, attempting to efficiently and accurately model rather than optimize the objective at a new task given previous information from another task.

The field of robust controller selection [62, 106] takes a different approach. These methods do assume that single control/environment samples are expensive, but seek to find a *robust* controller, rather than an *adaptive* control policy. Their formulation treats the environment parameter as noise, which is only useful when the environment is unobservable or changes at a timescale much shorter than the control bandwidth. In addition, when the choice of optimal control parameters changes significantly across the potential range of environments, the quality of a robust controller degrades significantly.

In the bandit literature, the subtopic of contextual bandits (e.g., [65]) involves a concept of "side information" used to inform the selection of an arm that is similar to the environment parameter mentioned above. However, contextual bandits do not allow for a training phase where this context can be intelligently chosen. They also differ from expensive optimization approaches in all of the ways that standard bandits do, as described above.

Reinforcement Learning of Control Policies

The idea of reinforcement learning (RL) [99] is also applicable to the optimization of control policies, but acts on a different timescale with assumptions about observability that cannot

always be met for locomoting robots in the field. In the RL framework the system is described by an underlying Markov Decision Process (MDP); from each *state* a set of *actions* can be taken which map to other states via a set of *transition probabilities*. Each such transition generates some reward. Given some starting state distribution, RL aims to learn a policy, or a state $\rightarrow$ action map, which maximizes the expected total reward when executed on the MDP; this basic idea has many applications. For a general comparison of RL and expensive optimization, I refer the reader to [97].

Note the difference in terminology between expensive optimization over a range of tasks and RL. In expensive optimization, various *environments* or *tasks* refer to different conditions the robot may encounter and react to; in RL this concept is embodied in the *state* of the system. This notion of state usually does not include internal state of the robot such as its shape, but rather the position and orientation of the robot in the world. In RL there is a single *environment*, the entire MDP; in other words the robot can move through many different states within the environment, and can react differently to each of these.

This terminology hints at other key differences. First, RL is typically used for planning control policies which operate at a fast timescale – continually reacting to new states. This means that the notion of a single expensive experiment refers to an *episode*, or a series of actions on the MDP, where many state/action samples are taken. For the expensive optimization work described in this thesis, these control policies operate on a longer range timescale. An environment is sensed, and a controller is chosen in response that will move the robot over several seconds – several full gait cycles – or longer before a significant change is detected which will require a change in controller. In these cases, a single action is expensive, and these evaluations are limited.

A second key difference between these areas is in the effect of actions. The MDP structure of RL problems means that an action causes a *transition* to another state. This is a reasonable assumption for problems where an approximate distribution of expected states is given (some

notion of what the world is like). However, in the control policy optimization work in this thesis there is no clear mapping between states; running a certain controller on an environment simply results in a reward based on the quality of the resulting motion, rather than a transition to a different environment. If this work involved motion through a known environment, where the position within that environment could be sensed, only then would RL methods would be more appropriate.

Finally, related to both of these ideas is the fact that the goal of expensive multi-environment learning is the selection of both state *and* action for the next sample; in RL most methods assume the state is given and so focus on optimal action selection to best learn a policy. This assumption is relaxed in methods which allowing the learner to pick its state [54], but these ideas focus on minimizing the number of state selections [68].

3.3 Multi-objective Optimization

Robotic systems often have multiple competing notions of good performance (energy efficiency, speed, stability, etc.). As described in §2.5, one approach to managing this tradeoff is to search for a set of Pareto optimal solutions rather than a single optimal point.

Finding these sets of optimal points requires specialized optimization methods. For cases where the objective functions are linear, the NISE method [96] has been developed to converge quickly on a good approximation of the Pareto set, even in problems of very high dimensions. Multi-objective simplex methods such as [110], which extend the single-objective linear constrained optimization simplex method[1] developed in 1947 [24], provide exact solutions for the Pareto optimal set for linear objective functions.

For nonlinear cases there are also a number of methods; perhaps the most popular is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [25]. This empirically has been

[1]Note that this simple method differs from the Nelder Mead constrained nonlinear optimization method [73].

shown to produce good results which are well distributed over the *Pareto front* – the Pareto optimal set given in objective space coordinates.

Expensive Multi-objective Optimization

In the case of expensive experiments, there is significantly less literature on identifying the Pareto set; the evolutionary methods used in standard multi-objective optimization (MOO) are most appropriate when samples are cheap and parameter and objective spaces are very high dimensional.

Some approaches for expensive MOO attempt to extend successful single objective expensive optimization techniques. In [58], a single aggregate objective function is created through a weighted combination of individual objectives as terms in a Tchebycheff function [35]. Each iteration, a new set of weights are chosen, and the experiment which maximizes the EI of that particular aggregate objective is chosen.

Other approaches attempt to work in the full objective space rather than simplifying the problem to one objective. For example, Keane [53] attempts to directly measure the multivariate EI of a point – how much the hypervolume of the Pareto front increases (c.f. Fig. 2.11b). The expression Keane presents is a simplification of the true quantity and only measures improvement as an increase from a single point on the Pareto front; Emmerich et al. [28] redefine this improvement more rigorously (yet are still able to find a closed-form expression) using Lebesgue integration on a partition of the objective space. However, none of these methods are designed for or tested with noisy function evaluations.

3.4 Locomotion Over Obstacles

As described in §2.3, existing work in snake robot locomotion is largely focused on cyclic gait motions for uniform environments, as these provide a convenient coupling between the large

number of degrees of freedom, rather than special-purpose motions for overcoming obstacles, which make no such assumptions. To address the complexity of complete manual control of these degrees of freedom, Baker et al. instrumented a small mock-up of the robot to move by hand [5]. This kinesthetic approach is less useful for cyclic motions, and more useful for slow, deliberate inputs to move the robot over significant obstacles. The small mock-up of the robot used by Baker et al. did not have the same size, shape, or sensors of the full robot; although it provided a method to control the full system, the lack of representational fidelity could decrease the effectiveness of the tool in eliciting user demonstrations.

Imitation Learning

Outside of snake robot locomotion, such learn-from-demonstration techniques have been used to great success. A large class of machine learning techniques have been developed for teaching a robot to follow the example of a provided expert example; these are termed *learning from demonstration* or *imitation learning*.

One challenge is in selecting an appropriate representation for the learned controller; this model must be flexible and descriptive [46]. Although learning a model that tracks a demonstrated expert input works well when that input is optimal, often when learning even from experts in the field the demonstration is not optimal (or near optimal). In these cases, a second phase involving RL can be added to improve the learned trajectory. Standard RL approaches are ill-suited to this problem when the state is high-dimensional, and so specific algorithms have been developed for this task [59, 102].

In this thesis, novice input is used in addition to expert input; as such an improvement phase will be critical for any learned motions. Tuning such a reward function in any case assumes the existence of a reward function (and the availability of the data necessary to compute it). When such a function is ambiguous or not available, inverse optimal control [1, 82] methods seek to learn the teachers intent, or the implicit reward function they are

optimizing, from the observed behavior. One problem with nearly all existing imitation learning methods, as pointed out by [78], is the danger of bad performance in recovering from mistakes because of the bias in state distribution of examples seen by the learner; Ross and Bagnell [88] propose a solution to this based on Conservative Policy Iteration [51].

3.5 Stochastic Binary Optimization

After obtaining a demonstrated trajectory, it (usually) must go through an optimization process to work effectively and robustly on the robot. Many of these processes described in the previous section use a complex, hand tuned reward function during this optimization which is a function of many elements of the robot state – its position, orientation, location of objects in the environment, to name a few. Outside of using a full motion capture setup, or obtaining point cloud sensor data of the surroundings, these state variables may be difficult to obtain. For the snake robot in particular, such sensors may not fit in the constraints of the system, and when used in the field, such motion capture setups are impractical. The only data available may be the success or failure of the task upon replay.

The methods described above for optimization of demonstrated trajectories are not suited for this regime of limited feedback (especially when coupled with very few trials). However, active learning (§3.1.1) is primarily focused on learning the binary class membership of a set of unlabeled data points, but attempts to accurately learn class membership of all unlabeled points with high confidence, which is inefficient if the loss function is asymmetric (if it is more important to identify successes than failures). The active binary-classification problem discussed in [30] focuses on finding a Bayesian optimal policy for identifying a particular class, but assumes deterministic class membership (whereas the result of trials with physical systems is often stochastic).

The bandit literature also considers the similar *binary* or *Bernoulli* multi-arm bandit

problem [20, 111], where pulls from the bandit arms are from independent Bernoulli distributions (providing stochastic binary feedback). However, as described in §3.1.2 the focus of this work is to minimize bounds on cumulative regret, typically in infinite horizon settings.

Chapter 4

Expensive Optimization for Robot Locomotion

This chapter discusses the application of the expensive optimization techniques described above in §2.4 to the optimization of the performance of robotic systems. Optimization of these systems often falls under the purview of expensive, black-box optimization techniques; specifically, the objectives of interest lack an accurate generative model, do not return gradient information, and are expensive (in computation, money, or time) to sample.

Although high fidelity models exist for some robotic systems (e.g., industrial robot arms), many locomoting systems exhibit complex environment interaction which is difficult to inexpensively model with any degree of fidelity.[1] Accurate models are needed to obtain accurate estimates for objectives such as speed, energy consumption, and robustness or repeatability of a motion; these are just a few examples of objectives which fall into this category.

As the environment becomes more complex – bumpy, irregular ground or rubble piles – slip, friction, and intermittent ground contact become even more difficult to model. The only

[1] Of course, available models should not be ignored; they should be used to direct the sampling of the objectives on the robot or provide useful statistics about the objective that can be used to improve the regression.

reliable way to test these systems is to evaluate the objectives on the real robot, measuring the objective directly (e.g., how far did the robot move, or how much energy did it consume). As these experiments are conducted on the physical robot, information that would be obtained from analytic objectives, such as the gradient, is not available.

Experiments on any real robot take time to set up and run. These usually cannot feasibly be run continuously for days or even hours; current research robot designs are not intended for such constant use, and someone must manage experiment setup and execution. Use of the robot incurs a (potentially hidden) cost which also must be considered; parts wear out and must be replaced, and so efficient use of the robot is preferred. Even simulations and models may require significant computational resources if they are of high fidelity (the computational fluid dynamics models referenced in [47] can take hours for a single evaluation).

In this chapter, I address the basic application of expensive black-box optimization techniques to robotic systems. In particular, I focus on optimization of a snake robot (c.f. [101]), which has difficult to model dynamics and a clearly defined parameter space (objectives considered are functions of the parameters of the gait model). However, the challenges discussed above which are encountered with this system are representative of those experienced with other robotic systems, and I hope the advances described here can help make these tools more usable in other fields as well. Finally, I focus on optimization using the EI [70] selection metric as it has been proven effective in previous work [50, 44, 79] and has no tuning parameters, allowing for true out-of-the-box use.

The main contributions of the work in this chapter are:

1. Improvement of the robustness of standard expensive black-box optimization; these modifications allow it to run with less supervision and overcome many typical problems. In particular, the improvements affect GP model selection and hyperparameter optimization.

2. Derivation of a theoretically accurate extension of EI to handle stochastic systems, and discussion of the ramifications of using existing techniques on stochastic systems.

3. Empirical results comparing:

   - Performance of the expensive optimization algorithm with and without the suggested GP model selection improvements.
   - Performance of several approaches for expensive stochastic optimization, including a novel extension of EI as well as several alternative baselines.

4. Demonstration of the algorithms on a physical system (the snake robot described previously).

5. Performance improvements in several modes of snake robot locomotion, both over flat ground and up slopes.

Many of the results in this chapter (and in this thesis) are empirical rather than theoretical; this focus is motivated by my observations that algorithms that obtain theoretical optimality guarantees with a given set of parameters often alter these parameters in order to generate better empirical results. Convergence guarantees are also omitted here, although this is a well studied problem (c.f. Vazquez and Bect [103]). In fact, the techniques I describe for model selection (which achieve a marked improvement on algorithm performance) actually invalidate theoretical guarantees of convergence from Vazquez and Bect's work. However, note that obtaining convergence guarantees on search algorithms is trivial (e.g., uniform random search will converge), and algorithms such as epsilon-greedy or rapidly exploring random trees are able to obtain such convergence guarantees simply by adding a random step, even though they can be shown to perform arbitrarily poorly in practice.

Finally, the results in this thesis were generated using software that is published with the thesis (downloadable from `http://www.mtesch.net/thesisCode/`). Although there is

no guarantee provided with the software (as per the Community Research and Academic Programming License [67] it is released under), and it has been changed since some of the robot experiments were initially obtained, I hope that this is a useful tool that will save time for other researchers in this field.

4.1 Notation and Problem Statement

In this thesis, I use the following notation and conventions (see Fig. 4.1 for a depiction of these terms, and the glossary of terms at the beginning of this document as a quick reference). First, the domain of the objective function is given by X, where $X \subset \mathbb{R}^k$; this is typically a space of parameters that can be varied such as trajectory parameters of a robot arm or gains in a control loop. The objective function is $f\colon X \to \mathbb{R}$, and the maximizer $\operatorname{argmax}_X f(x)$ is written as x_{opt}. GPs (see §2.1) are the method used to produce a surrogate function, and they provide both an estimate of the objective f, written as $\hat{f}_\mu\colon X \to \mathbb{R}$, as well as a (normal) predictive probability distribution of possible output values at each $x \in X$, written as $p_Y^x\colon \mathbb{R} \to \mathbb{R}^+$. In other words, the predicted probability density that $f(x) = y$ is $p_Y^x(y)$. As with any probability distribution, $\int_{-\infty}^{\infty} p_Y^x(y)\,dy = 1$ for any $x \in X$. Moreover, the GP provides a joint predictive probability distribution for any set of points $\mathbf{x} = x_1, x_2, \ldots, x_n$; this is an n-dimensional multivariate Gaussian written as $p_Y^{\mathbf{x}}$.

One important piece of intuition to have about GPs is that they can be viewed as a probability distribution over functions. As noted, the GP provides a joint predictive probability distribution for any set of points $\mathbf{x} \subseteq X$. Because this set of points can be very large (practically infinite), it can be used to represent the domain of the function (e.g., $\mathbb{R}^m$), assigning a probability density to each candidate function f^*.

The goal of expensive global optimization is to sequentially choose points at which to

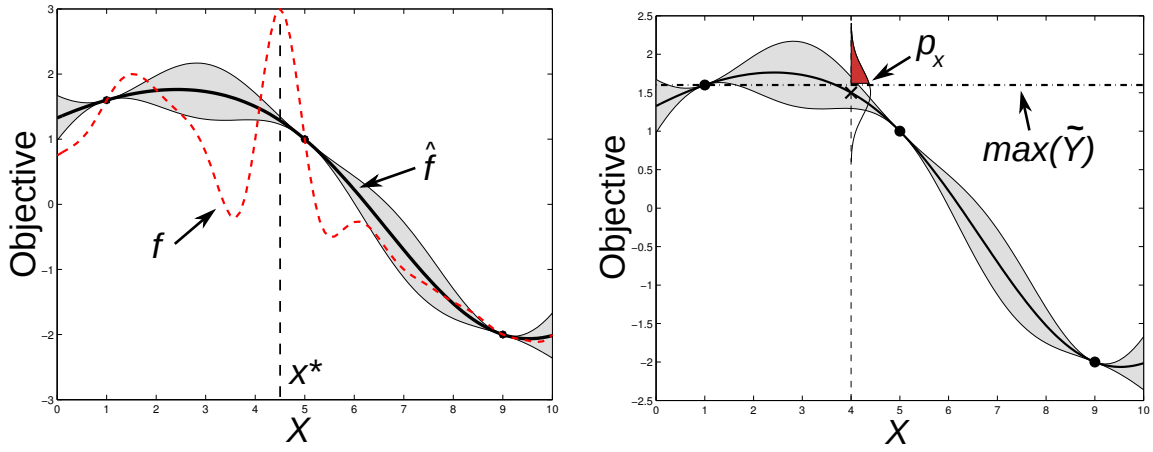


Figure 4.1: **(a)** The objective function f, surrogate $\hat{f}$, and maximizer x^*. **(b)** The best sampled point, $\max(\tilde{\mathbf{y}})$, and the predictive distribution for a potential sample location. The red shaded area of this distribution indicates the portion of this distribution considered *improvement* in this context.

sample which result in the best $\hat{x}_{opt}$ according to a loss function, usually the *simple regret*[2] $f(x_{\text{opt}}) - f(\hat{x}_{opt})$ (c.f. [15] for a comparison of simple and cumulative regret). The choice of each subsequent sample location as well as $\hat{x}_{opt}$ is informed only by the results of evaluations of the previous samples. We define $\tilde{\mathbf{x}}$ as the set of sampled points built up by the algorithm, and $\tilde{\mathbf{y}}$ represents the results of evaluating these points on the expensive objective f. $\hat{x}_{opt}$ is usually restricted to points within the set of sampled values $\tilde{\mathbf{x}}$ for practical reasons (it can be risky and unwise to choose a parameter which has never been tested on the system). In this case, the simple regret is given as $f(x_{\text{opt}}) - \max(\tilde{\mathbf{y}})$.

[2]The term regret in this context refers to the *regret* you suffer from not choosing based on complete and perfect knowledge.

Algorithm 1 Basic Black-Box Optimization

1: $\tilde{\mathbf{x}} \leftarrow$ space-filling design of k points
2: $\tilde{\mathbf{y}} \leftarrow \{\}$
3: **for** $i \leftarrow 1$ **to** k **do**
4: addToList($\tilde{\mathbf{y}}$, $f(\tilde{\mathbf{x}}\{i\})$
5: **end for**
6: **for** $i \leftarrow k+1$ **to** n **do**
7: $\hat{f} \leftarrow$ conditionGP($\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$)
8: $x_i \leftarrow \text{argmax}_X \text{metric}(\hat{f}(x))$
9: addToList($\tilde{\mathbf{x}}$, x_i)
10: addToList($\tilde{\mathbf{y}}$, $f(x_i)$)
11: **end for**

4.2 Improvements For Robust and Effective Optimization

As noted in §2.4, global optimization of noisy, non-convex, black-box functions such as robot performance metrics cannot efficiently be accomplished by many standard techniques. In this thesis, I focus on development of a class of surrogate-function based global optimization methods, often termed expensive "black-box" methods (c.f. survey by Jones [49]), which are generally described by the steps in Alg. 1.

The two important steps in this algorithm are finding the surrogate function (line 7) and choosing an appropriate selection metric (line 8). Although more focus in the literature (as well as in this thesis) is given to the latter, we first address the former. Both from my own experience and from informal discussions of other practitioners of these techniques, one of the largest hurdles for new and experienced users when using surrogate-function methods is simply obtaining consistent and reasonable regressions from the GP used for the surrogate function. Because of this difficulty, some researchers substitute other techniques such as random forests [13] in place of GPs (c.f. Hutter et al.'s Sequential Model-based Algorithm Configuration (SMAC) from [45]). While the choice of regression method for the surrogate should be based on a number of factors, I hope the work in this thesis and the provided code

will eliminate "initial frustration with and learning curve of GPs" from that list.

In this section, I describe several approaches that can be combined to improve the "fitting" process for the GP. More accurately, I describe how the mean functions, covariance functions, and hyperparameters of the underlying GP can be chosen to result in more robust high-level algorithm performance. The improvements aim to make these optimization techniques work in an "off the shelf," parameter-free manner, i.e., the user does not need knowledge of the internal workings of the optimization algorithm in order use it for their own problem, and does not need intuition from years of experience tuning algorithm parameters.

4.2.1 Model Selection: Hyperparameters

A GP is *non-parametric* in that it determines a probability distribution over functions by conditioning on sampled data; however, it is not strictly without parameters which must be tuned. To produce a quality fit of the underlying function, they rely on a reasonable choice of a *covariance function* (which defines the general function behavior) and a good selection of *hyperparameters* for that covariance function (which determine quantities such as the length scale of the function in each dimension). If a low-quality choice is made, not only is the fit of the function poor, but the performance of any algorithm that depends on it is adversely affected. Many problems encountered by users initially attempting to use GPs for regression are a result of poor selection of covariance hyperparameters.

Leave-one-out Likelihood

Recall from §2.1.6 the use of marginal likelihood of the training data $\{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}\}$ as a method to select among hyperparameters. Although the marginal likelihood provides a commonly used measure of regression model quality, cross validation techniques often replace marginal likelihood as a model quality measure to prevent overfitting to the data. Cross validation (CV) is the practice of evaluating the accuracy of a model by partitioning collected data,

using one subset to train the regression, and the remaining data (the *held out* data) to quantify the model's predictive quality. Leave-one-out CV is an extreme form of CV which holds out each data point individually (training the model on the remaining data), and averages the accuracy measure from each.

I have found that the use of leave-one-out (LOO) likelihood in place of the marginal likelihood can reduce overfitting when selecting GP hyperparameters (this has also been noted in [98]) and therefore I use this in the improved black box algorithm below; however the following sections apply to either standard or LOO likelihood, and I include options to use either in the provided code.

To compute the LOO likelihood, first define an error function *err* that describes the predictive error in the model. This is done by comparing the i^{th} observed value $\tilde{\mathbf{y}}_i$ with the GP's prediction at the i^{th} sample location $\tilde{\mathbf{x}}_i$ using training data with the i^{th} element removed, $\{\tilde{\mathbf{x}}_{-i}, \tilde{\mathbf{y}}_{-i}\}$. We denote the (scalar) mean and variance of this GP posterior at $\tilde{\mathbf{x}}_i$ as $\hat{f}_\mu{}^{-i}$ and $\hat{f}_{\sigma^2}{}^{-i}$ respectively.

Intuitively, this error function represents how well the model predicts a held out element of the training data if trained on all of the data except the held out element. A simple example of such a function is the squared loss

$$err(\tilde{\mathbf{y}}_i, \hat{f}_\mu{}^{-i}, \hat{f}_{\sigma^2}{}^{-i}) = (\tilde{\mathbf{y}}_i - \hat{f}_\mu{}^{-i})^2. \tag{4.1}$$

This represents the error in the model, but penalizes the model for errors the same whether it has high or low confidence in the returned value. A more appropriate error function is the log probability loss (see Equation 5.10 of [81])

$$err(\tilde{\mathbf{y}}_i, \hat{f}_\mu{}^{-i}, \hat{f}_{\sigma^2}{}^{-i}) = -\frac{1}{2}\log \hat{f}_{\sigma^2} - i - \frac{(\tilde{\mathbf{y}}_i - \hat{f}_\mu{}^{-i})^2}{2\hat{f}_{\sigma^2}{}^{-i}} - \frac{1}{2}\log 2\pi. \tag{4.2}$$

Regardless of the error function, it is then computed by holding out each element of the

training set; the summation of these individual terms gives the LOO likelihood

$$L_{\text{LOO}}(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \sum_{i=1}^{|X|} err(\tilde{\mathbf{y}}_i, \hat{f_\mu}^{-i}, \hat{f_{\sigma^2}}^{-i}). \tag{4.3}$$

Likelihood Optimization: Random Restarts, Intelligent Initial Conditions

Although optimization of the likelihood through simple techniques such as conjugate gradient is effective when the GP has a reasonably dense number of points in the input space, when conditioning on sparse data (as is common in expensive optimization tasks), these often terminate early or reach an incorrect local optimum due to the relatively uniform nature of this likelihood function. As seen in Fig. 4.2, this hyperparameter optimization involves a search over a potentially multi-modal likelihood function, with large areas that are essentially flat (and so have no useful gradient information for gradient-based optimization techniques).

In order to robustly find the minimum[3] of this nonlinear, non-convex optimization problem, I use local optimization with random restarts. This begins with reasonable parameter ranges for the hyperparameters, generating a large number of seeds uniformly at random in the hyperparameter space, and then uses standard gradient-based techniques such as the conjugate gradient line search included in the GPML MATLAB software [80] to maximize the likelihood from each of these seeds.

This large global search of the likelihood function is effective, but also can be slow, especially if the number of random seeds for likelihood optimization is large. I have noted that often the previous maximum likelihood estimate (MLE) hyperparameter value (from the regression with $n-1$ points) is in the basin of attraction for the current likelihood optimization. For this reason, I store the previous MLE hyperparameters to use as one of the seeds; this increases stability while allowing the number of seeds to be greatly reduced.

[3]Often, for practical reasons of numerical stability, the maximum likelihood estimator is found via minimization of the negative log likelihood.

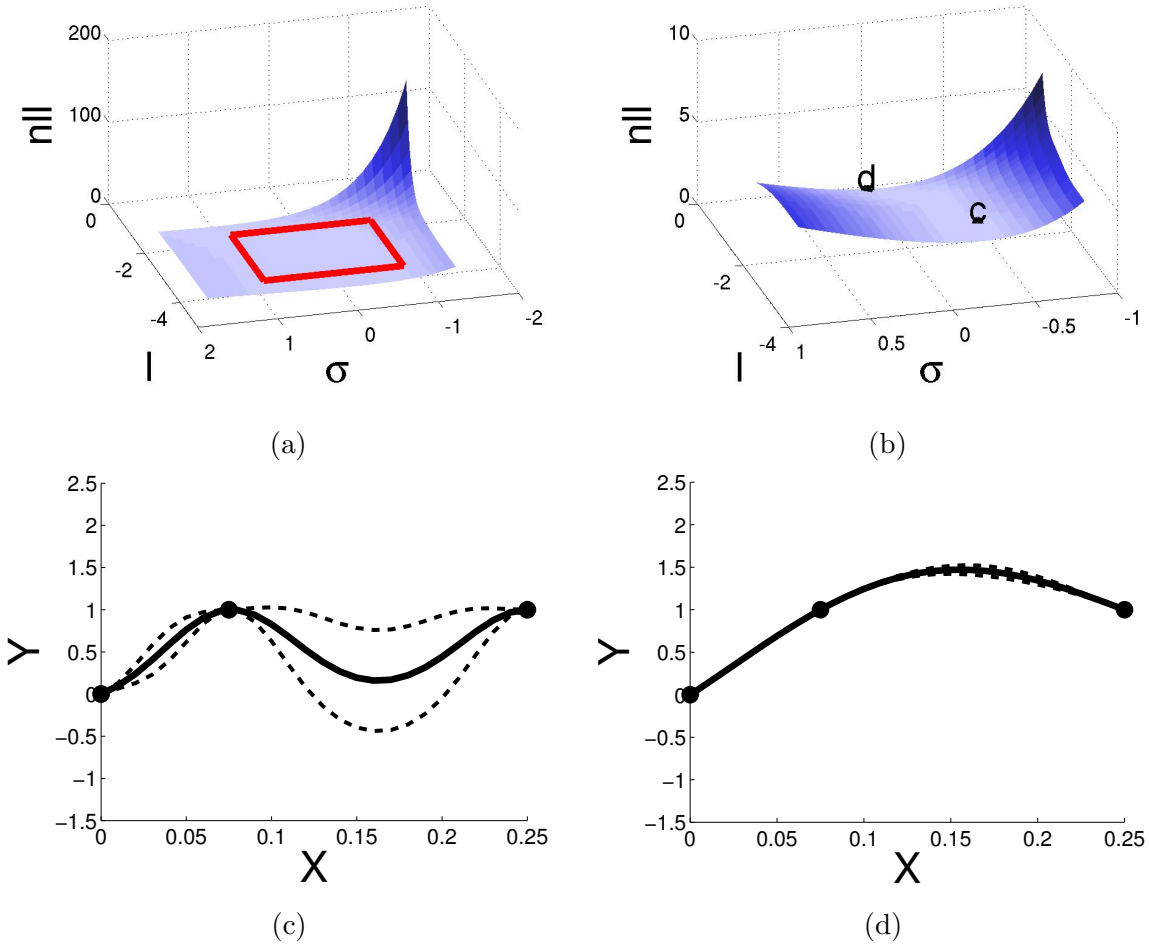


Figure 4.2: **(a)** When conditioning on sparse data, large regions of the likelihood surface can be relatively uniform, causing it to be difficult to reason between different hyperparameter values. **(b)** A zoomed in view of the red boxed region of (a); the central valley also extends much further as ℓ decreases with no significant change in the negative log likelihood value. The different markers correspond to the different GP fits in **(c)** and **(d)**, which have approximately the same likelihood but represent significantly different hypotheses.

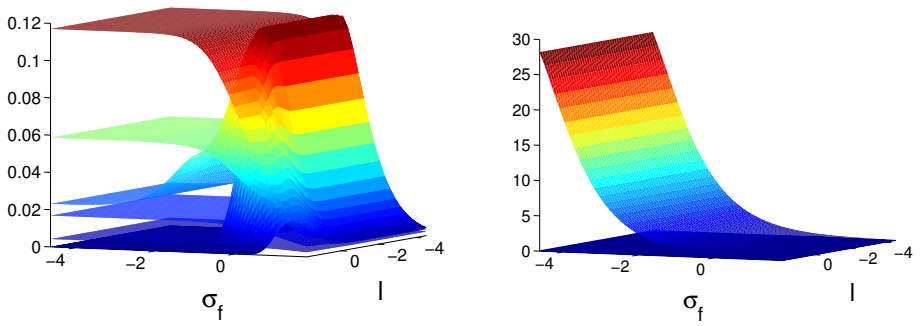
However it is not sufficient to only run a local optimization from the previous MLE hyperparameters as these can lie outside the new global optimum's basin of attraction, hence the need for the random restart process described above. Understanding that the change in the model likelihood function will be greater with the 6^{th} point than the 106^{th}, as the number of data points increases we decrease the number of random restarts (but maintaining some to allow for a non-zero chance of escape from potential local optima). The initial GP fit is one of the most important, and therefore uses the largest number of random seeds; speed is increased in later iterations through the use of fewer seeds.

The expensive optimization code included with this thesis implements this multi-resolution, random restart approach for hyperparameter likelihood optimization, and to improve performance optionally only completes the final line search with a fraction of the large number of random seeds (those with the maximum likelihood).

4.2.2 Model Selection: Mean and Covariance Function

Robustly finding the MLE hyperparameters for a given choice of mean and covariance function is important, but when the data is sparse and the model is complex (e.g., contains noise hyperparameters, and independent length scales for each dimension) the likelihood function is not very discriminative – many of the hyperparameters have comparable marginal likelihoods – and can even be multimodal. This often indicates multiple explanations for the data under the model – for example, one optimum could represent the data primarily explained as noise, and the other as points from a nearly noiseless function (as demonstrated in Fig. 4.5).

This ambiguity can indicate that the model is too complex. As Occam's Razor suggests, the simplest competing hypothesis that can explain the data should be chosen. This holds true for mean and covariance function complexity; using simple choices for these functions (e.g., no noise term and uniform length scales) in the presence of sparse data produces more robust fits with higher likelihoods. To determine the best choice of mean and covariance for



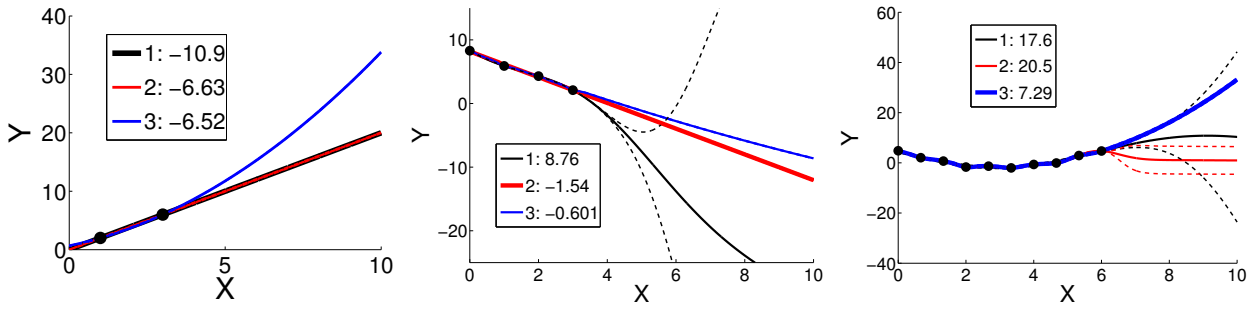
(a) (b)

Figure 4.3: **(a)** Sections of the likelihood surface for a three-hyperparameter model (isometric squared exponential covariance plus a diagonal noise term). The 2-D l and σ_f surfaces are shown for various values of σ_n. **(b)** When the simpler two-hyperparameter model (zero mean, isometric squared exponential covariance) is used, the likelihoods are much larger – the sloping surface is the likelihood for the two parameter model, whereas the slices from the three parameter model are shown at the very bottom – note the differing scales between these figures.

the given data at a particular step, I turn to model selection.

In this work, I have implemented an automatic model-selection process in order to make the GP fit more stable. This process takes place by changing the covariance function and mean function as well as varying their hyperparameters. Instead of assuming a single model family (e.g., a zero mean and isometric squared exponential covariance function), a set of potential model families (e.g., a family of zero mean and isometric squared exponential covariance functions and a family of zero mean and periodic covariance functions) is used. During the regression process the MLE hyperparameters for each family are found, and the family whose MLE hyperparameters have the highest likelihood among all of the model families is chosen. Complex model families have lower likelihoods over much of the hyperparameter space (because this likelihood is spread over more dimensions, as seen in Fig. 4.3), so in practice this leads to automatic selection of the simplest reasonable hypothesis (Fig. 4.4).

To balance speed and quality, a multiresolution sampling procedure also is applied here. For each potential mean and covariance function combination, a local search from many random restarts is run. In addition, the previous MLE hyperparameters for each model



(a) (b) (c)

Figure 4.4: Automatic model selection can be accomplished by choosing a mean and covariance function whose MLE hyperparameters result in the highest likelihood over all of the models. Each figure above is shown with GP posteriors using the MLE hyperparameters for three models of varying complexity (each used an isometric squared exponential covariance, and the mean functions were mx, $mx + b$, and $ax^2 + bx + c$), with the most likely drawn as a thicker line. The legend also displays the negative log likelihood of each. For a small number of data points, simple models are chosen; as the complexity of the data increases, more complex models are naturally chosen. This automatic model selection serves to help reduce overfitting.

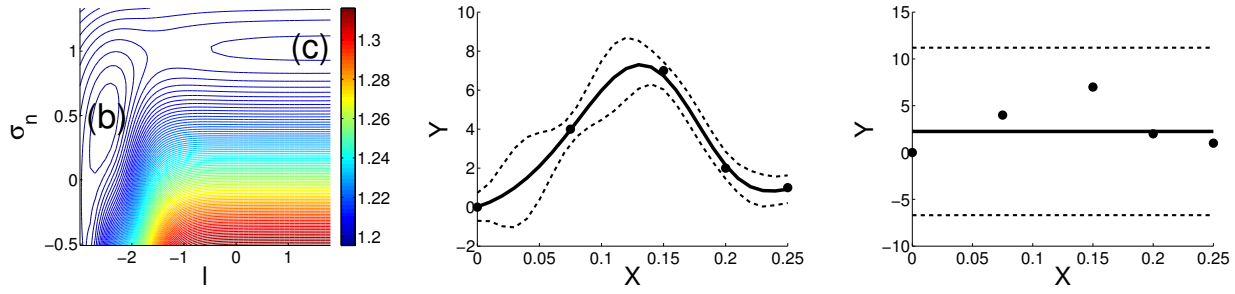
family are kept as a seed for the next fit. The family that previously was chosen as the best is given preference with more random restarts.

4.2.3 Fitting GPs to Stochastic Data

When collecting data from a stochastic source such as a physical system, it is important to take this stochasticity into account in the surrogate function serving as a model for the system. I take the view that objective function samples from the robot can be viewed as realizations of a random noise variable added to an underlying latent function, as described in §2.1.7.

However, this adds another hyperparameter to the GP, generalizing the model and increasing the chance of a poor fit to the true data[4]. Specifically, for small number of data points, the hyperparameter likelihood often has a large, flat minimum region or is multimodal, containing a local optimum describing the data with very little noise and a local

[4]If the system has a known sampling variance (known ϵ), this can be fixed during the hyperparameter likelihood maximization step, eliminating the additional hyperparameter.



(a) (b) (c)

Figure 4.5: **(a)** A slice from the 3 dimensional negative log likelihood surface for a squared exponential covariance function with a diagonal noise term (the signal parameter σ_f varies from 3.6 to 2.5 with the change in ℓ in order to capture both minima in this surface). For visualization purposes (to avoid washing out the minima due to the scale of the remainder of the surface), atan scaling has been applied. **(b)** The GP fit corresponding to the maximum that explains the data as relatively low noise. **(c)** The GP fit corresponding to the maximum that explains the data as pure noise.

optimum describing the data as almost pure noise (Fig. 4.5). This further motivates the use of the model selection process described in §4.2.2.

4.2.4 Demonstration of Algorithm Improvements

The suggested improvements in the previous sections are not algorithmic changes per se, but allow the hyperparameter selection process to occur with minimal human intervention and without any knowledge of the true function behavior. These improvements, many of which are standard techniques used for optimization problems, prevent overfitting and allow GP function regression to be reliably and easily used to generate surrogate objective functions that can be used in high-level algorithms such as EGO. This is important because the use of these surrogates entails the assumption that they are a reasonable representation of knowledge about f; if the regression is grossly inaccurate or cannot fit the data, then the surrounding algorithms will perform poorly (further motivation to include prior knowledge to guide the set of covariance functions and hyperparameter choices during model selection).

The specific changes are:

Algorithm 2 Robust Black-Box Optimization

```
1: function OPTIMIZE(f)
2:     x̃ ← space-filling design of k points.
3:     ỹ ← {}
4:     for i ← 1 to k do
5:         addToList(ỹ, f(x̃{i})
6:     end for
7:     savedHPs ← NULL
8:     for i ← k + 1 to n_r do
9:         n = GETNUMRANDRESTARTS(i)        ▷ Random restarts decrease as i increases
10:        f̂, savedHPs ← ROBUSTFITGP(x̃, ỹ, savedHPs, n_r)
11:        x_i := argmax_X metric(f̂(x))
12:        addToList(x̃, x_i)
13:        addToList(ỹ, f(x_i))
14:    end for
15: end function
```

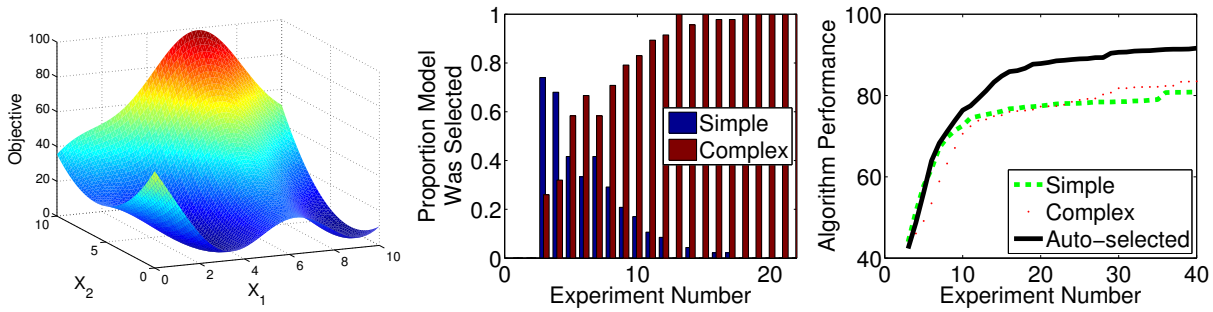
- Stochastic GP fit through addition of an independent noise term to the covariance function diagonal
- Save the previous MLE covariance function hyperparameters as a seed for a local search during the next iteration.
- Use a local search procedure with random restarts for MLE hyperparameter optimizations (this should also be used for optimization of your selection metric – such as EI – as well)
- Model selection over multiple candidate covariance functions through comparison of marginal likelihood.
- Use of LOO likelihood instead of marginal likelihood to prevent overfitting.
- Multi-resolution sampling procedure for random restarts, dependent on potential covariance function as well as iteration.

Algorithm 3 Robust GP Fitting

```
1: function RobustFitGP(x̃, ỹ, prevBestHPs, numRandomRestarts)
2:     covOptions ← set of potential covariance functions
3:     bestHPs ← {}
4:     bestLikelihoods ← {}
5:     for i ← 1 to Length(covOptions) do
6:         cov ← covOptions[i]
7:         if prevBestHPs ≠ NULL then                ▷ Consider the previous best HP
8:             prevBestHP ← prevBestHPs[i]
9:             bestHP, bestLikelihood ← LocalLikelihoodOpt(x̃, ỹ, cov, prevBestHP)
10:        else
11:            bestHP ← 0
12:            bestLikelihood ← −∞
13:        end if
14:        for j ← 1 to numRandomRestarts do          ▷ Try a number of random restarts
15:            randHP = Random(X)
16:            likelihood, HP ← LocalLikelihoodOpt(x̃, ỹ, cov, randHP)
17:            if likelihood > bestLikelihood then
18:                bestHP ← HP
19:                bestLikelihood ← likelihood
20:            end if
21:        end for
22:        Append(bestHPs, bestHP)
23:        Append(bestLikelihoods, bestLikelihood)
24:    end for
25:    bestIdx ← indexOfMax (bestLikelihoods)
26:    f̂ ← ConditionGP(x̃, ỹ, covOptions[bestIdx], bestHPs[bestIdx])
27:    return f̂, bestHPs
28: end function
```

The resulting algorithm (abbreviated version shown in Alg. 2) still contains the basic structure of the original black-box optimization algorithm (Alg. 1), but significantly improves the algorithm's results on physical systems through the improved GP fit summarized in Alg. 3. Following are empirical comparisons with and without some of these changes to demonstrate the effect they have on algorithm performance. The code to generate these results can be found in Appendix B.1.

To demonstrate the importance of an automatic model selection process, I show the



(a) (b) (c)

Figure 4.6: The result of model selection on optimization with EI on a simple noisy test function. Each algorithm was run 50 times. **(a)** The latent deterministic test function; Gaussian noise with a standard deviation of 4 was added. **(b)** For the auto-selection approach, the proportion of experiments in which each model was chosen after a certain number of data points. **(c)** The average optimization performance in terms of best sampled point for each single covariance function as well as the automatic model selection process.

result of optimizing a stochastic function with and without model selection between multiple covariance functions. The same optimization task is performed 50 times each with (a) only a noiseless covariance function, (b) only a noisy covariance function, and (c) automatic selection between these; see Fig. 4.6.

As this shows, the simple model is generally preferred initially, but with more datapoints the complex model with the noise term can better explain the collected data. By using LOO likelihood to select the model, overfitting with the complex model is greatly reduced. The algorithm performance also matches what is expected; initially, the simple model and auto-selection approaches are comparable, but as the data becomes more difficult to explain, the simple model provides poor regression leading to ill-informed experiment selection and worse overall performance.

Obtaining quality GP fits to sparse and growing data, while not specifically part of the expensive optimization metric, results in large performance improvements of the surrounding optimization algorithm and is therefore a crucial component of this thesis. The optimization of the selection metric (line 11 of Alg. 2) is also a crucial step, but I have found it to be more readily solved. In the supplied code, I have implemented a simple approach using random

restarts of MATLAB's implementation of active-set optimization; this has proven effective. Jones has also outlined an approach for this optimization in his EGO algorithm [50].

4.3 Extensions for Stochastic Objectives

The motivating system in this thesis, the snake robot, is stochastic; commanding the robot to execute the same motion multiple times will result in a random distribution of end positions. Due to this non-deterministic behavior, objectives based on this motion are also stochastic. In §4.2.3, I have discussed additional considerations pertaining to the regression process. Here I focus on the computation of the selection metric in the presence of noise; in particular, I develop several extensions to the EI selection metric to account for this stochasticity.

First, we define samples of the stochastic function as realizations of a random variable added to the latent objective,

$$f_{\text{noisy}}(x) = f_{\text{lat}}(x) + \epsilon, \tag{4.4}$$

where ϵ is drawn from $\mathcal{N}(0, \sigma_n^2)$. Recall the definition of EI (see Equation (2.22)): it is the expectation of *improvement* over the *best experiment* $\max(\tilde{\mathbf{y}})$, or

$$\text{EI}(x) = \int_{\max(\tilde{\mathbf{y}})}^{\infty} p_Y^x(y)(y - \max(\tilde{\mathbf{y}}))\, dy. \tag{4.5}$$

There is an implicit assumption in this equation that a particular function evaluation will always yield the same result; noisy function evaluations can invalidate this assumption even if the regression accounts for the noise. In the deterministic case, the value of the best experiment is $y_{\max} = \max(\tilde{\mathbf{y}})$. However, in the stochastic case, the meaning of the value of the best experiment is not clear (see Fig. 4.7).

A naïve approach would be to assume the result of the experiment was not noisy, and

simply take the deterministic approach. However, because there is an additive noise term, the underlying objective value $f_{\text{lat}}(x_{\max})$ will not always equal evaluations of $f_{\text{noisy}}(x_{\max})$, leading to an offset from the underlying function value which can negatively affect the result of optimization relying on this estimate. Instead, the choice of $y_{\max}$ should be based on estimates of the unobserved f_{lat} at the sampled $\tilde{\mathbf{x}}$ rather than the sampled $f_{\text{noisy}}(x)$ values.

As f_{lat} is not known exactly, this approach intuitively leads to a representation of $y_{\max}$ as a random variable. In this case, definition of the notion of *improvement* also requires consideration. This is normally given as the distance above the best previous evaluation, $\text{I}(y) = \max(y - y_{\max}, 0)$; if $y_{\max}$ is a random variable, then $\text{I}(y)$ becomes one as well.

In the remainder of this section, I develop simple heuristics to try to work around these concerns (c.f. [43, 77]), and also describe several alternative approaches. I then describe positive and negative attributes of each one (computational complexity, fidelity of approximation to a true Bayesian approach, etc), and provide an empirical comparison of the resulting performance.

4.3.1 Baseline Approaches

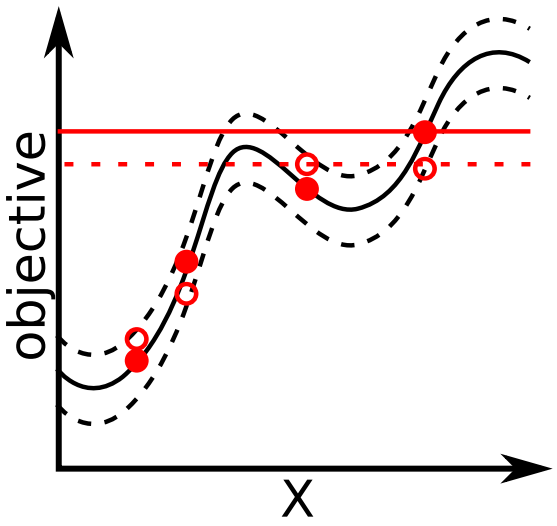
All of the baseline approaches described here use the standard EI equations, but alter the definition of $y_{\max}$, the best previous experiment term.

The first baseline is a naïve approach which ignores noise. This approach defines

$$y_{\max}^{\text{b1}} = \max(\tilde{\mathbf{y}}) \tag{4.6}$$

as in the noiseless case.

Next, we consider that $\tilde{\mathbf{y}}$ represents realizations of $f_{\text{lat}}(\tilde{\mathbf{x}}) + \epsilon$, and use $\hat{f}_{\mu}$ (the mean prediction of f_{lat}), as a surrogate to calculate $y_{\max}$:



(a) .

Figure 4.7: **(a)** If function evaluations (circles; underlying function values are dots) are taken from a noisy function (solid curve with 1-σ variance as dashed lines), the maximal sample (dashed horizontal line) does not necessarily correspond to the value or location of the underlying function's maximum at the sampled locations (solid horizontal line).

$$\widehat{\tilde{\mathbf{y}}} = \left\{ \hat{f}_{\mu}(x) \,\forall\, x \in \tilde{\mathbf{x}} \right\} \tag{4.7}$$

$$y_{\max}^{\text{b2}} = \max(\widehat{\tilde{\mathbf{y}}}) \tag{4.8}$$

This incorporates learned information about the level of observed noise into the prediction of $y_{\max}$ (through the use of the predicted latent function).

As the estimate $\hat{f}$ also contains uncertainty, the next baseline takes this into consideration. This is a "confidence bound" style approach, which contains a parameter β that can be tuned to alter the performance of the metric. When $\beta < 0$, this is a *lower confidence bound* approach, and when $\beta > 0$, an *upper confidence bound*.

$$\widehat{\tilde{\mathbf{y}}}_{CB(\beta)} = \left\{ \hat{f}_{\mu}(x) + \beta \, \hat{f}_{\sigma^2}(x) \;\forall\; x \in \tilde{\mathbf{x}} \right\} \tag{4.9}$$

$$y_{\max}^{\mathrm{b3}} = \max(\widehat{\tilde{\mathbf{y}}}_{CB(\beta)}) \tag{4.10}$$

Recall from §2.1.7 that a stochastic GP fit contains two notions of variance, the *process noise* added to the diagonal of the covariance representing the stochastic nature of the function (e.g., inconsistent repeatability of trajectories on a robot), and the *model uncertainty*, representing the uncertainty in the prediction of the underlying function. The $\hat{f}_{\sigma^2}$ in (4.9) represents the second of these two uncertainty terms.

4.3.2 Bayesian Approaches

The baselines described above take into consideration the predicted value and uncertainty of the best objective sample, and use various simplifying assumptions to represent their estimate as a single point. A Bayesian approach would instead represent the uncertainty in the value of the best objective sample as a random variable $Y_{\max}$ and a corresponding probability distribution over possible maximum values $p_{y_{\max}}(y)$.

The representation of $y_{\max}$ as a random variable has an effect on calculation of the EI. When $y_{\max}$ is a known deterministic quantity, the improvement (from Equation (2.20))

$$\mathrm{I}(x) = \max(\hat{f}^x - y_{\max}, 0) \tag{4.11}$$

is a random variable that is deterministic except for $\hat{f}^x$ (the random variable associated with p_Y^x, considering only the uncertainty in f_{lat} and not the predictive noise). When $Y_{\max}$ replaces $y_{\max}$,

$$\mathrm{I}(x) = \max(\hat{f}^x - Y_{\max}, 0), \tag{4.12}$$

the improvement is dependent on two non-independent random variables, $\hat{f}^x$ and $Y_{\max}$. The expectation $\mathrm{EI} = \mathbb{E}[I]$ should then be calculated over the joint distribution $p(\hat{f}^x, Y_{\max})$, which for many forms of the distribution of $Y_{\max}$ is not possible in a closed-form expression.

The following approaches describe several methods to estimate $Y_{\max}$ using different simplifying assumptions. The subsequent section compares the efficacy of these approximations when used to calculate EI for expensive stochastic optimization tasks with that of the baselines described above, as well as the relative required computation of the methods.

Predicted Distribution at Best Experiment: Although the parameter which generated the best sampled objective value may not be the parameter which has the best expected sample value or underlying latent function mean (recall Fig. 4.7), perhaps the simplest method to generate an expression for $Y_{\max}$ is to assume that it is. In other words, let x_{best} be the sampled $x_i \in \tilde{\mathbf{x}}$ which corresponds to $y_i = \max(\tilde{\mathbf{y}})$. Then define

$$p^{\text{B1}}_{y_{\max}} = p_Y^{x_{\text{best}}}. \tag{4.13}$$

Note that because $\hat{f}$ is a GP, the joint distribution of $\hat{f}^x$ and $Y_{\max} = \hat{f}^{x_{\text{best}}}$ (needed to evaluate the expectation of Equation (4.12)) is Gaussian, and easily obtained from the GP. In this and the following approaches, only the variance from the uncertainty in the latent function (and not the process noise) is considered for $p_Y^{x_{\text{best}}}$.

Predicted Distribution at the Maximum GP Mean: Rather than using the best sampled objective value as the basis for determining the distribution for $Y_{\max}$, another option is to use the GP's predicted distribution at the location of the highest predicted mean value at the sampled points, or

$$x_{\text{best}} = \operatorname*{argmax}_{x \in \tilde{\mathbf{x}}} \hat{f}_\mu(x), \tag{4.14}$$

$$p^{\text{B2}}_{y_{\max}} = p_Y^{x_{\text{best}}}. \tag{4.15}$$

Jointly Predicted Maximum at Sampled Points: Finally, instead of the Gaussian predictive distribution at a single point, one could use the more accurate distribution of the maximum at all of the points in $\tilde{\mathbf{x}}$[5]:

$$p^{B3}_{y_{\max}} = \max\{\hat{f}^x \,|\, x \in \tilde{\mathbf{x}}\}. \tag{4.16}$$

Because the GP defines the joint normal distribution of $\{\hat{f}_{x_1}, \hat{f}_{x_2}, \ldots, \hat{f}_{x_n}\}$, this results in computing the distribution of the maximum of dependent normal random variables. Unfortunately, this results in a non-Gaussian distribution for which an analytic expression for the PDF or even the expectation is not known (bounds on the expectation of this quantity are investigated in [87]). In the case of infinite series of i.i.d. random variables, the field of extreme value theory has found representative distributions of this quantity, but for correlated finite variables the results are not as well understood. Due to these challenges, this estimation method for $Y_{\max}$ is not considered below.

4.3.3 Stochastic Optimization Results

In this section, I empirically compare optimization performance of the various methods above. The optimizations here are run on several test functions in various dimensions with various levels of uniform Gaussian noise added to the function.

[5]Indeed, this could be done for all the points in the parameter space as well; the relative merits of this more extensive approach are not considered.

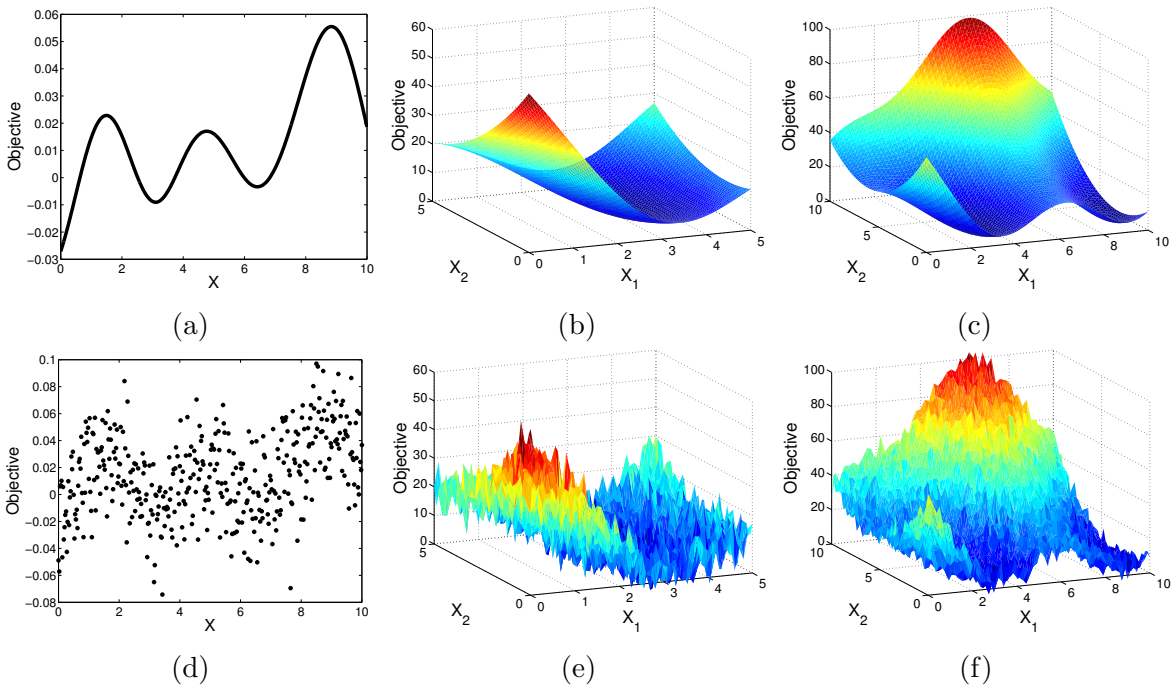


Figure 4.8: Test functions considered for an empirical comparison of various adaptations of the EI metric to stochastic functions. The top row shows the deterministic latent f_{lat}, and the bottom row a dense sampling of the stochastic f_{noisy}. From left to right, a 1-D function with a high level of noise, a 2-D function with a medium level of noise, and a second 2-D function.

The overall measure of performance used to compare these methods is the true function value at the GP's best predicted point, or $f(\text{argmax}_{x_i \in X}\, \hat{f}_\mu(x_i))$. A secondary consideration for any of these methods, especially those which require numerical integration, is computation time. This is also displayed with the results in this section.

The synthetic test functions used for this comparison are shown in Fig. 4.8. For each of these, zero mean Gaussian noise with different standard deviations (0.1 and 0.75 were considered) was added to the test function.

To draw conclusions for the improvement given by each of these approaches, the standard baselines of random experiment selection and EI with a GP without a noise covariance hyperparameter were first tested. The baselines described in §4.3.1, using EI with $y^{\text{b1}}_{\max}$, $y^{\text{b2}}_{\max}$, and $y^{\text{b3}}_{\max}$, were also tested. Finally, the Bayesian approaches of estimating the distribution of

$Y_{\max}$ (see §4.3.2) were tested. For this computation of the expectation of improvement over a joint bivariate distribution, two approaches were compared. First, assuming the variables are independently distributed, a simple 1-D numerical integral of EI was computed by taking the expectation over the closed form EI expression for values $y_{\max}$ sampled from $Y_{\max}$,

$$\mathrm{EI}^{\mathrm{1D}}(x) = \mathbb{E}[\mathrm{EI}^{y_{\max}}(x)] = \int_{-\infty}^{\infty} \mathrm{EI}^{y_{\max}}(x)\, p_{y_{\max}}(y_{\max})\, dy_{\max}, \tag{4.17}$$

where $\mathrm{EI}^{y}(x)$ is the EI of $\hat{f}(x)$ over y. Also considered was the two dimension expectation integral over the joint PDF of $\hat{f}(x)$ and $Y_{\max}$, p_{joint}:

$$\begin{aligned}
\mathrm{EI}^{\mathrm{2D}}(x) &= \mathbb{E}[\mathrm{I}(x)] && (4.18)\\
&= \mathbb{E}[\max(\hat{f}^{x} - Y_{\max}, 0)] && (4.19)\\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \max(y_t - y_{\max}, 0)\, p_{\mathrm{joint}}(y_{\max}, y_t)\, dy_{\max}\, dy_t && (4.20)
\end{aligned}$$

Note that Equation (4.17) and Equation (4.18) give different results because $\hat{f}(x)$ and $Y_{\max}$ are not independent (but as discussed above, for the Bayesian methods B1 and B2 this joint distribution is Gaussian and easily computed).

Fig. 4.9 shows the performance of each of these approaches for each test function, and Fig. 4.10 depicts the relative computation required by the different methods. The test scripts used to obtain these results are given in Appendix B.2.

One interesting conclusion that these results suggest is that optimization of stochastic functions can be effectively accomplished with relatively simple changes to the terms in the EI metric. Deterministic approximations for the unknown $y_{\max}$ consistently perform on par with or better than the more complex approximations for its distribution. Furthermore, these simple approximations are much simpler to compute, resulting in faster performance and less error-prone implementation details (such as numerical integration challenges).

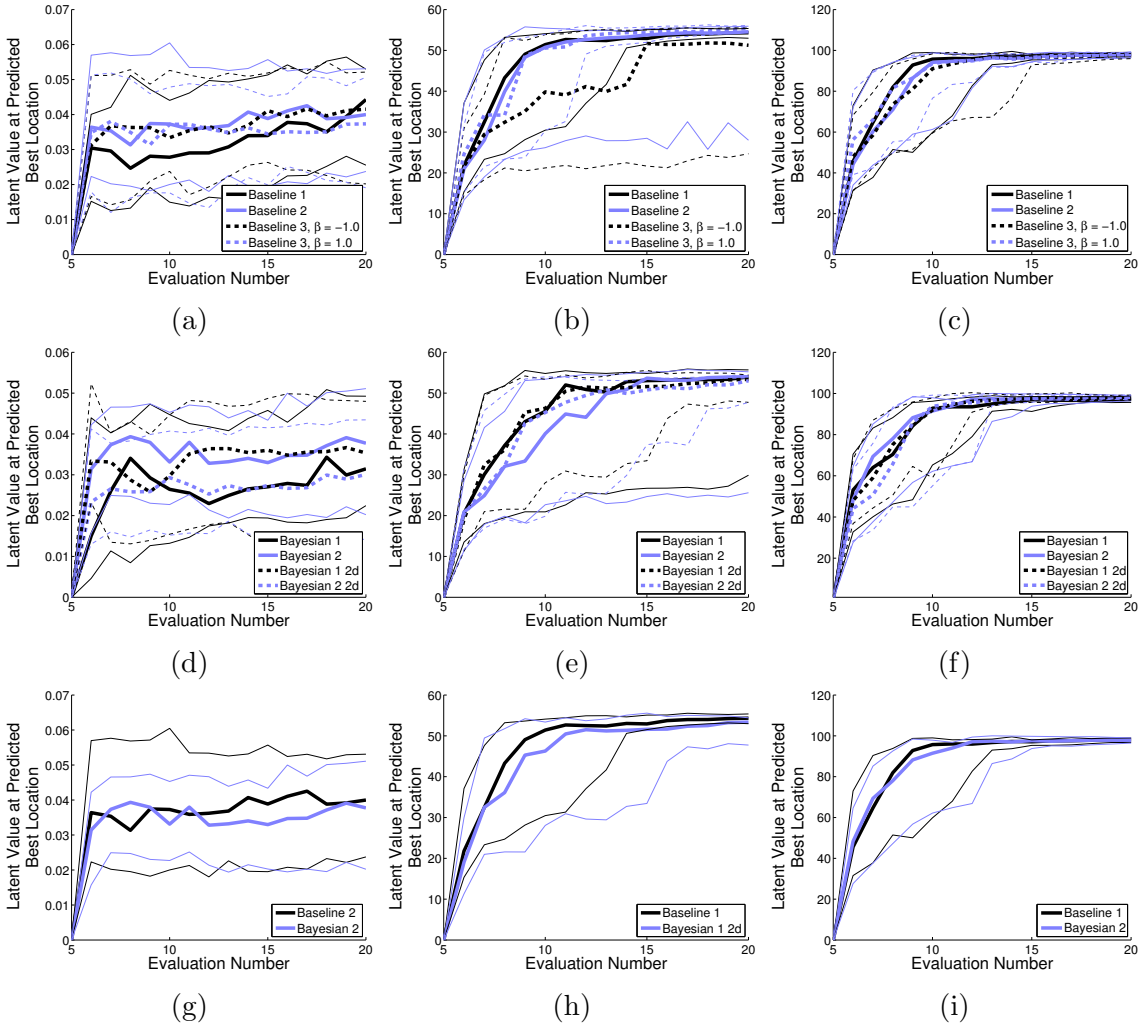


Figure 4.9: For each test function, 50 tests were run for each approach described in the text. Each column shows results from the test function in the corresponding column of Fig. 4.8. The first row compares the baseline approaches, the second row the described Bayesian approaches (using both 1-D and 2-D integrals), and the third row compares the best baseline to the best Bayesian approach. The thick lines are the median algorithm performance, defined as the quality of the recommendation, and the thin lines are 25% and 75% quantiles. Note that for each test functions, the baselines perform on par with or better than the best Bayesian approach.

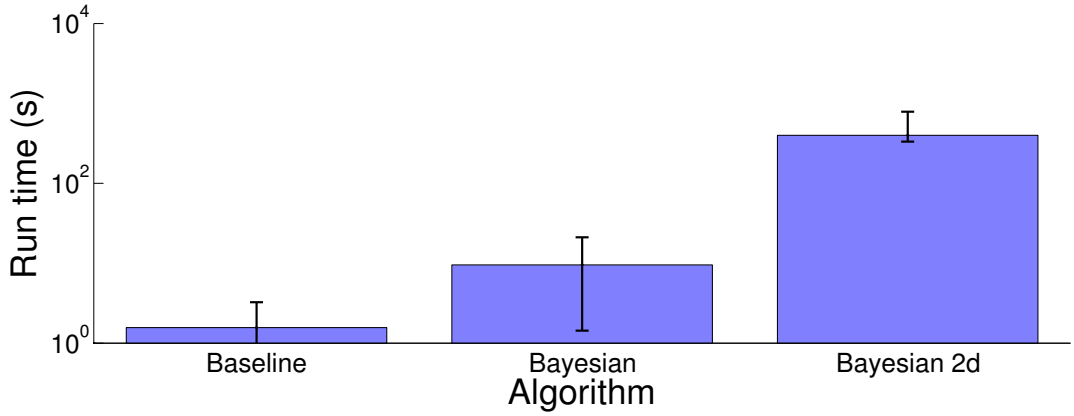
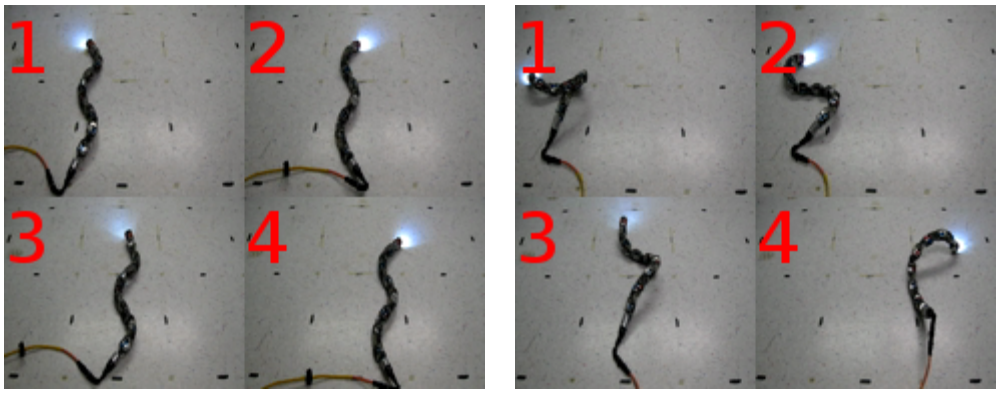


Figure 4.10: A comparison of the computational time required for the different algorithms (during optimization of test function 2). In each category of methods (baseline, Bayesian, and 2-D Bayesian) the results were comparable, so only one example of each is shown. The height of the bar represents the average computation on steps 11-20, and the error bars give the 25th and 75th quantile. Note that the data is on a semilog plot; the computation required for more complex methods is several orders of magnitude more than the simpler methods. These tests were run on a 2.6 GHz Intel Core 2 processor.

4.4 Optimization of Snake Robot Locomotion

The goal in developing this improved black-box optimization approach was to improve open-loop performance of a physical system, in particular the snake robot described in §2.3. The objective of interest for the robot was locomotion speed over various terrains (other objectives are discussed in future chapters). The chosen parameter space was the subset of the full gait equation parameters (c.f. [101]) which define the *sidewinding* gait. The terrains included flat ground, bumpy obstacles, and slopes with and without obstacles. As a comparison to the optimized performance, I used the performance of the hand-tuned default parameters for this sidewinding gait.

Initially, I optimized the speed of sidewinding locomotion across flat ground. The motion of the hand-tuned sidewinding gait is depicted in Fig. 4.11a, while Fig. 4.11b shows the motion resulting from the newly optimized parameters. This new motion not only achieves three times the speed of a gait hand-tuned for the same purpose, but is also a previously undiscovered method of snake robot locomotion; it is qualitatively different from the existing



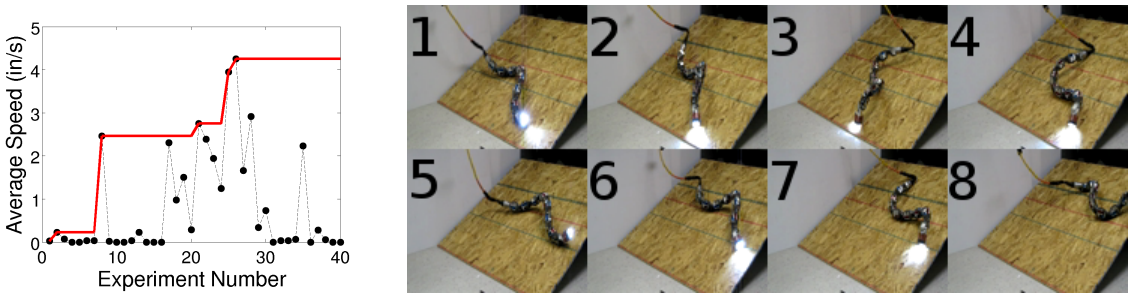
(a) (b)

Figure 4.11: **(a)** The previous hand-tuned sidewinding motion. **(b)** The optimized gait; although it used the same fixed parameter values as the sidewinding gait, it is qualitatively different than the existing sidewinding motion. Note that although this optimized gait has an increased amplitude, simply increasing amplitude from the previous hand-tuned gait will produce a less effective gait; this increase must be coordinated with a change in the other parameters.

sidewinding gait. Specifically, it is more dynamic, relying on momentum rather than solely kinematic forces in order to locomote.

The next task that the modified expensive black-box algorithm was used for was optimization of motion up a slope. One of the previous limitations of the snake robot was the inability to climb steep slopes. For example, using parameters for a sidewinding snake robot gait that were hand-tuned for flat ground locomotion, the robot would tip when attempting to climb up a 20 degree slope. However, using the algorithm described above, a number of improved solutions were quickly found. The optimized gait parameters, discovered after only 26 trials (Fig. 4.12a), resulted in a swinging motion that served to balance the snake on the slope during the climb (Fig. 4.12b).

Gait parameters for robot locomotion both on flat ground and on a slope were also optimized in the presence of obstacles (Fig. 4.13). For each combination of obstacle type and slope, 40 sequential gait parameters were tested (each subsequent test chosen as the result of maximizing EI over previous results). The resulting learning curves are shown in Fig. 4.14. In each case, a gait parameter choice was found which resulted in improved



(a) (b)

Figure 4.12: **(a)** The sampled speed from each experiment conducted during the optimization. The solid line represents the best value found so far, and is a useful visual comparison of results. The automatic tradeoff between exploration and exploitation can be seen; experiments 0-16 are more explorative; 17-27 are largely improving upon the best result so far. Determining a low chance for further improvement, the selection metric then chooses to explore the space (resulting in a number of poor choices) for the remaining experiments in the budget. **(b)** Still shots from the optimized climbing motion of the robot. In order to keep its balance, the robot "swings" from a horizontal position to one aligned with the slope of the incline.

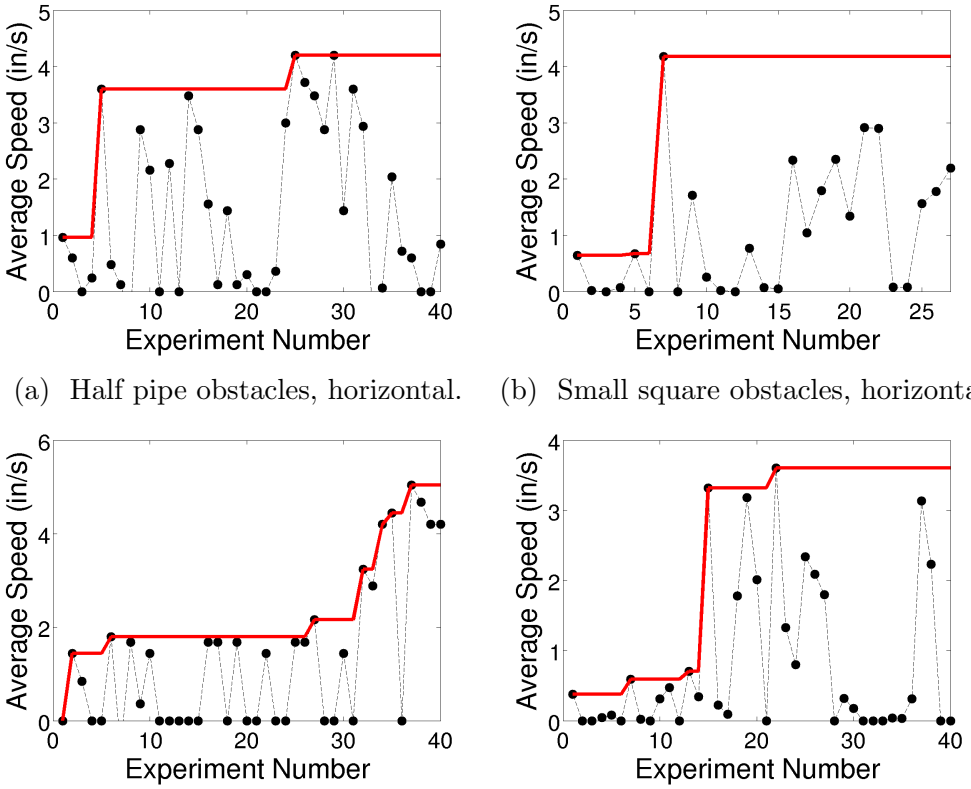


(a) (b) (c) (d)

Figure 4.13: Various obstacles were built on which to optimize open-loop gait performance. Note that each of these can either be inclined or flat or the ground. **(a)** A simple wood slope. **(b)** Irregularly spaced small square obstacles. **(c)** Three black pipes, split in half along the center. **(d)** Note the scale; the robot, even when lifting its body a moderate amount, cannot pass over the obstacles.

performance over the initial hand-tuned sidewinding gait parameters.

These results not only demonstrate the efficacy of the algorithm, but also extend the capabilities of the snake robot. More importantly, the new sets of parameters were found without expert knowledge regarding parameter selection; the only human interaction was resetting the snake and measuring the distance traveled. A benefit to the principled exploration of the parameter space rather than relying purely on human intuition is that global optima are often easier to discover. When hand tuning the sidewinding motion, the performance can decrease as the gait parameters are incrementally moved outside of a small range



(a) Half pipe obstacles, horizontal. (b) Small square obstacles, horizontal.

(c) Half pipe obstacles, sloped. (d) Small square obstacles, sloped.

Figure 4.14: Results from optimization tests on two sets of obstacles. Each obstacle was set flat on the ground for a first round of optimization tests; a second set of tests was conducted for each obstacle when set at an angle. For each test, the sampled objective function value is plotted for each experiment conducted, and a solid line indicates the best sampled value so far.

near the initial local optimum. By removing human inhibitions about leaving this "good" area of the parameter space, large unexplored regions of the parameter space are sampled, resulting in overall improved performance.

Chapter 5

Multiple Objectives

Many problems in robotics inherently require optimization of multiple conflicting criteria, such as the speed of a system and its energy efficiency. In these cases, the naïve approach is to consider a scalar combination of these criteria, e.g., to optimize a linear combination of speed and efficiency. However when no preferences are known between the criteria, such combinations limit the solution to a single point based on arbitrary preferences implied by the aggregate function. In multi-objective optimization (MOO) (see §2.5) these multiple objectives are treated explicitly as independent unless the user has a clear preference between them. Instead of a single optimum, this gives rise to a set of *Pareto optimal* solutions, termed the *Pareto set* (see Figure 5.1).

This full set of solutions is useful in real-world situations. For example, the snake robots which motivate the work in this thesis are equipped with an on-board camera to allow an operator to teleoperate the robot out of direct line-of-sight. This camera is fixed to the undulating body (often the head) of the robot, which causes difficulty in maintaining situational awareness during locomotion (Figure 5.2). To mitigate this effect, one can control the robot through small-amplitude low-frequency motions to stabilize the camera. However, such motions also reduce the speed of the system, creating a scenario with conflicting notions

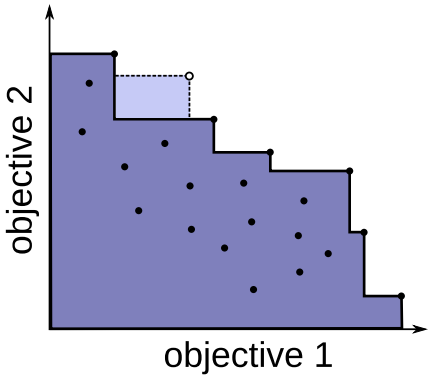


Figure 5.1: The set of Pareto optimal solutions are those which are not *Pareto dominated* by any other points in the full set of solutions. The Pareto optimal points in this figure are those which are on the black stair-stepped line; this set dominates the hypervolume delineated by the dark shaded region. The dominated hypervolume is a popular measure of the quality of a multi-objective solution set. The light shaded region is the potential improvement to the hypervolume if the point indicated by the circle were added to the set.

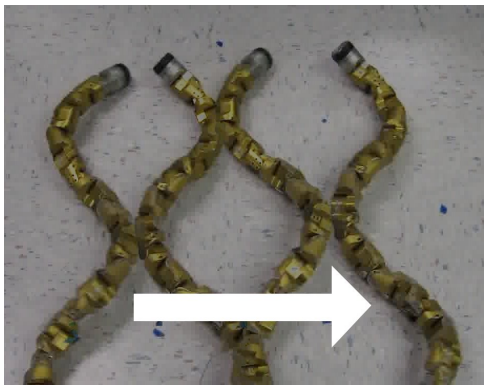


Figure 5.2: One problem when teleoperating our snake robots is that the large movement of the head, which houses the camera, can cause operators to become disorientated. We demonstrate multi-objective optimization methods in this chapter to help solve this problem, simultaneously optimizing for the conflicting objectives of head stability and speed.

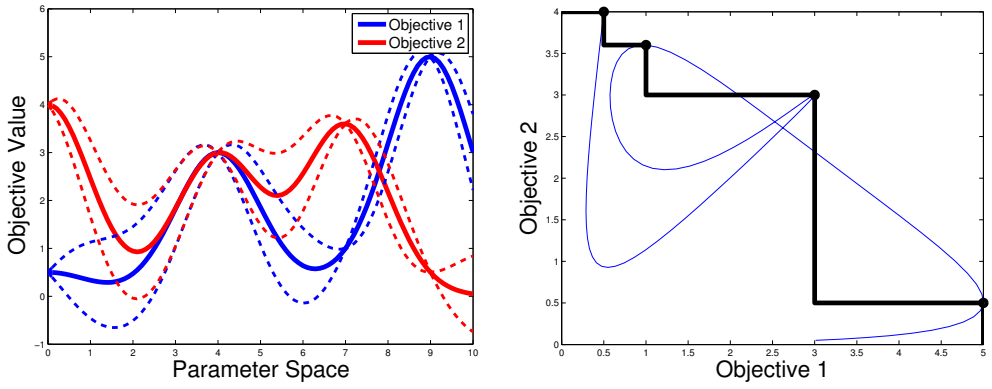
of good performance. In this case, speed may be important when the robot is teleoperated through a wide, easy to comprehend space, but as the robot enters a more complex passage, the stability of the head camera may become paramount. This clearly demonstrates how the relative importance of objectives can change during operation, motivating a search for the full Pareto set of solutions.

There is a large body of work on optimization of multiple objectives when a large number of objective evaluations can be made, or when the objectives are in a particular form (see §3.3). This thesis focuses on global optimization of *expensive* systems, and therefore the methods discussed in this chapter differ from much of the existing MOO work in the relatively small budget for objective evaluations (often around 40-50 for 2-3 dimension parameter and objective spaces). The goal of these algorithms is to find the best approximation to the Pareto set of solutions while restricted to such a budget.

In this chapter we seek to demonstrate the benefits of viewing robotic optimization problems in a MOO framework. I develop a new algorithm for obtaining a Pareto optimal set of solutions while budgeting the number of tests on the robot, based on Emmerich's derivation of expected improvement in hypervolume [28]. I also implement a leading existing approach for expensive MOO, Knowles's ParEGO [58] (described in §3.3). These are empirically compared on a set of test functions, and then a concrete real world MOO example objective for the snake robot – simultaneous improvement of head stability and speed during execution of sidewinding – is used to demonstrate the described algorithm.

5.1 Objective Space

Recall from §2.5 that MOO compares solutions in a multidimensional *objective space*. Whereas in single objective optimization, the objective is visualized as a curve or surface over some parameter space X (e.g., Fig. 4.8), in MOO the objective functions f_i are often shown



(a) (b)

Figure 5.3: Two objectives were sampled at $\tilde{\mathbf{x}} = \{0, 4, 7, 9\}$. **(a)** The resulting GP fits. **(b)** The objective space projection of the GP fits is shown as the thin line, while the thick line indicates the current Pareto front of the sampled points.

directly in the objective space as parameterized surfaces; the parameter space appears only implicitly. To obtain this parameterized surface, each point $x \in X$ is represented by $\mathbf{f}(x) = \{f_1(x), f_2(x), \ldots, f_n(x)\}$. For example, the result of this operation for GP surrogates of two simple 1-D functions is seen in Fig. 5.3.

Note that the parameter space often appears only implicitly in this work, and focus is on the $\mathbf{f}(x)$ that are associated with some x. Also, note that only part of the objective space is in the range of $\mathbf{f}$ and that $\mathbf{f}$ can be many-to-one; in other words, the mapping from X to Y is in general non-surjective and non-injective.

This focus solely on $\mathbf{f}(x)$ instead of pairs $(x, f_i(x))$ is useful for comparing solution quality. In Fig. 5.3a, is it not immediately clear which of the sampled points form the Pareto set. However, in Fig. 5.3b, the Pareto set of sampled points is immediately evident.

With surrogate-based algorithms, a separate GP regression is done for each objective independently. When fitting GPs to the objectives, the useful single-objective $\pm\sigma$ uncertainty-visualization technique for GPs (§2.1.5) does not carry over to the objective space view. Instead, each point $\mathbf{f}(x)$ results in an uncertainty ellipse.

5.2 Extending EI to Multiple Objectives

As the EI metric has been a success in expensive single-objective optimization, the natural question is whether it can be applied to the MOO case. Recall the notion of improvement for the single objective case (Equation (2.20)); if evaluating the objective at x (with a corresponding random variable $\hat{f}^x$ given by the surrogate), the predicted improvement over the set of sampled points $\tilde{\mathbf{y}}$ is simply the increase in the maximum value of the resulting set, or

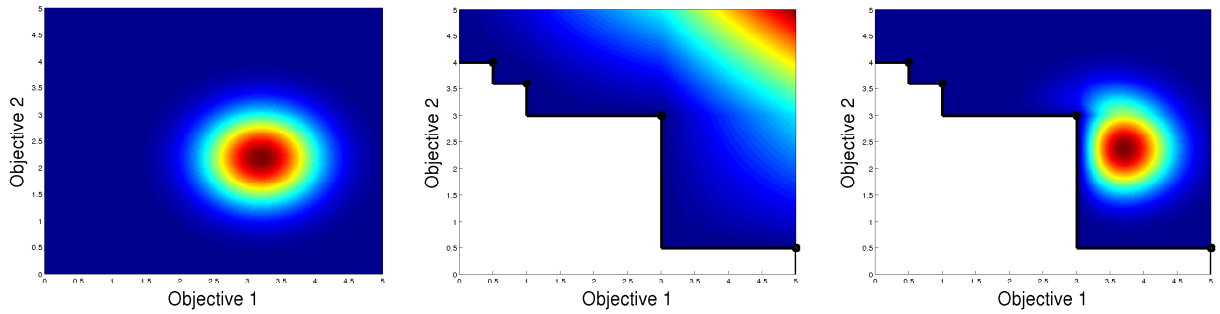
$$\mathrm{I}(x) = \max(\hat{f}^x - \max(\tilde{\mathbf{y}}), 0). \tag{5.1}$$

Note that as $\hat{f}^x$ is a random variable, $\mathrm{I}(x)$ is as well.

In MOO there is more than one objective, and the solution is not just a single point, but an entire *Pareto set* of points. To measure improvement in such a set, a valid metric must be devised to measure the quality of such a set. One such metric is the set's *hypervolume* [113]. This is the volume in objective space which is Pareto-dominated by at least one point in the Pareto set. To avoid infinite volumes, a reference point in objective space must be selected to define the lower bounds of this volume; this point should be chosen given prior knowledge of or an educated guess about the minimum possible value of the objective functions (e.g., the net displacement of a gait must always be greater than or equal to 0).

The concept of the hypervolume indicator is illustrated for a two-dimensional objective space in Fig. 5.1. This measure has desirable properties; for example it is not affected when a dominated point is added to a set of solutions, and the addition of a non-dominated point always increases a set's hypervolume. Given a method to compute the hypervolume HV of a solution set, improvement can be defined as

$$\mathrm{I}(x) = \mathrm{HV}(\tilde{\mathbf{y}} \cup \hat{\mathbf{f}}^x) - \mathrm{HV}(\tilde{\mathbf{y}}), \tag{5.2}$$



(a) (b) (c)

Figure 5.4: These objective space plots are computed using the example samples and function fits from Fig. 5.3. The expected improvement in hypervolume (EIHV) for a given parameter $x = 8$ is the expectation over the improvement in hypervolume for potential objective values $\hat{\mathbf{f}}^x$. Therefore, the integrand of Equation (5.3), given in **(c)**, is the **(a)** probability density of $\hat{\mathbf{f}}^x$ times the **(b)** improvement for each potential y in the objective space $Y_1 \times Y_2 \dots Y_n$. The resulting quantity is integrated over the objective space to obtain the final EIHV(x) value.

where $\hat{\mathbf{f}}^x$ is the multivariate random variable given by the surrogates' prediction for $\mathbf{f}(x)$. The expectation over this quantity can then be written as

$$\text{EIHV}(x) = \mathbb{E}[\text{I}(x)] = \int_{Y_1 \times Y_2 \times \cdots \times Y_n} \left(\text{HV}(\tilde{\mathbf{y}} \cup y) - \text{HV}(\tilde{\mathbf{y}})\right) p^x_{\mathbf{Y}}(y)\, dy \tag{5.3}$$

where this integral is taken over the objective space. In Fig. 5.4, the two components of this integral are shown. Recall that for the single objective case, EI is the distance which the center of mass of the probability of improvement distribution is above the best samples. Analogously, expected improvement in hypervolume (EIHV) is the distance which the center of mass of the probability of improvement distribution is above the set of best samples.

Although a simple closed-form solution has been derived for EI in the single objective case, $\mathbb{E}[\text{I}(x)]$ is not straightforward when the improvement is measured in terms of hypervolume; evaluating this for a given test point requires either a multidimensional numerical integral, or the development of an analytic form for this expectation. Fortunately, Emmerich et al. have provided the outline of a method to compute this quantity in closed form [28]. This method divides the objective space into a grid based on points on the Pareto front, computes a closed

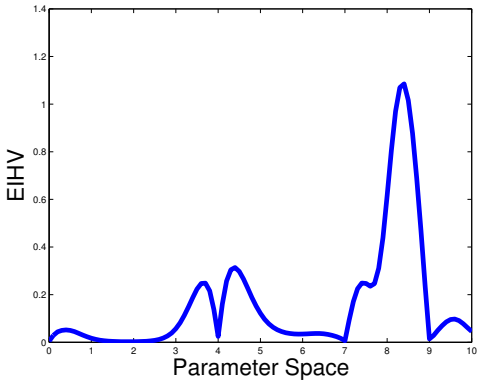


Figure 5.5: The expected improvement in hypervolume for each point in the parameter space (corresponding to the GP fits in Fig. 5.3).

form integral in each of these grid squares, and then adds the contributions to obtain the resulting EIHV for a given point in parameter space.

Working in objective space, EIHV considers the joint improvement in all objectives simultaneously, trading off the per-objective benefit of evaluating a point. Using the example functions from Fig. 5.3, the EIHV value for each point in the parameter space is shown in Fig. 5.5.

Using hypervolume as the indicator of solution set quality and finding an efficient computation of its expectation allows us to use machinery from the single objective case for optimization with multiple objectives. The resulting multi-objective optimization algorithm using EIHV metric can be summarized in a form parallel to the surrogate-based single objective algorithm defined in Alg. 1. The modifications are that line 7 is replaced by a fit for *each* objective, and the metric used in line 8 is EIHV.

As with single objective surrogate-function based algorithms, it is imperative that the function regression method provides a reasonable estimate of the objective and the uncertainty of that estimate. The hyperparameter tuning methods from §4.2 are useful in obtaining a realistic and non-trivial GP fit.

5.3 Synthetic Test Problem Results

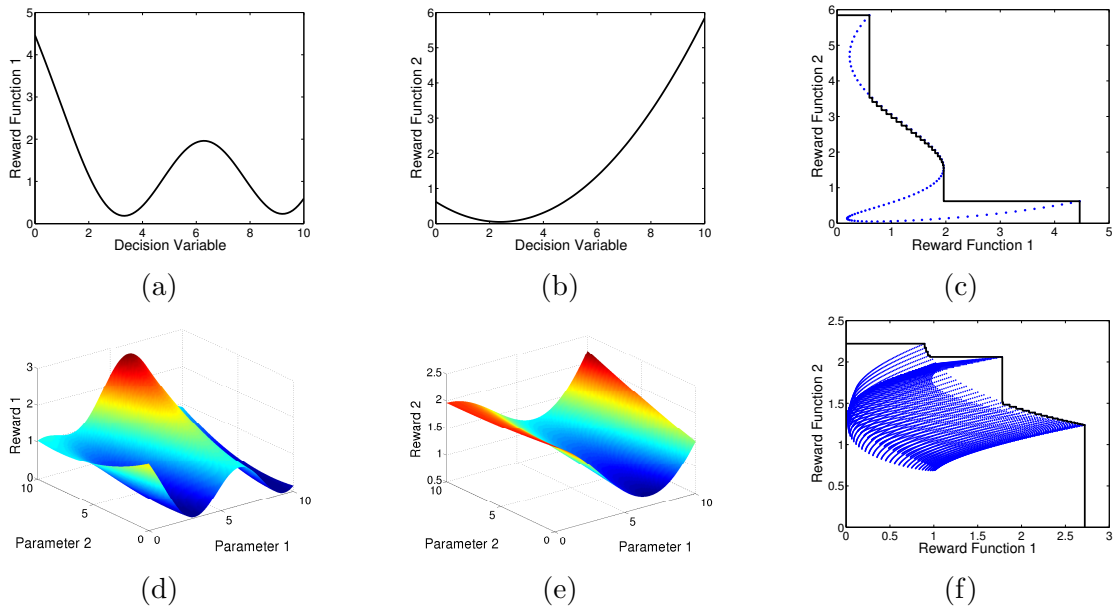


Figure 5.6: Competing test functions created from one dimensional slices (**(a)** and **(b)**) and scaled two dimensional regions (**(d)** and **(e)**) of the Branin test functions. The plots in **(c)** and **(f)** show the resulting objective space projection (blue dots) and Pareto frontiers (black lines).

The primary motivation for the careful experiment selection methods described herein is the expensive nature of tests on physical robotic systems. It is unreasonable to run an exhaustive set of tests on such a system to compare algorithm performance. Instead, a more extensive comparison on two simple (and cheap) analytic functions was run.

Many of the multi-objective test functions in the literature are particularly designed to confound existing multi-objective evolutionary algorithms (MOEAs), and therefore involve large high-dimensional parameter spaces with many disconnected Pareto set regions. The low-dimensional analogues, when they exist, are trivial surfaces that do not provide for a reasonable evaluation of the optimization algorithms under consideration.

Instead, I chose to use a region of the Branin test function, a common benchmark for global single objective optimization from the Dixon-Szegö test problem set [26]. I chose different regions of the Branin function for each objective, and found that the resulting

surfaces exhibit qualitatively similar properties to those observed for the performance of our robots. These regions, shown in Figure 5.6, are defined by the following equations, where $B\colon \mathbb{R}\times\mathbb{R}\to\mathbb{R}$ is the original Branin function and all inputs are between 0 and 10 inclusive:

$$f_1^1(x) = B(x,1)/10, \tag{5.4}$$

$$f_2^1(x) = B(3,x)/10, \tag{5.5}$$

$$f_1^2(x_1,x_2) = B(x_1,2+0.5x_2)/20 \tag{5.6}$$

$$f_2^2(x_1,x_2) = B(0.4x_1,5+0.1x_2)/10 \tag{5.7}$$

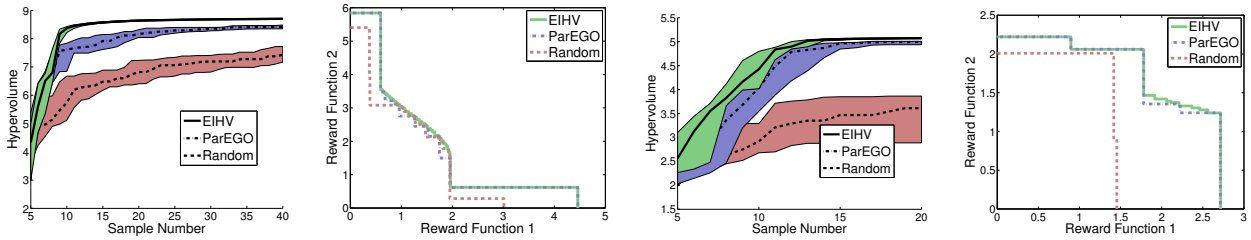
In addition to the optimization method described in §5.2, I have chosen a simple, popular, state-of-the-art optimization method for expensive multi-objective problems called ParEGO [58]. This algorithm takes the approach, at each iteration, of generating a single aggregate objective function[1]. It then reverts to a single-objective experiment selection method (the EI-based algorithm EGO, described in [50]) to choose the next sample. Finally, I also use random experiment selection to provide a baseline from which to measure the importance of any careful experiment selection.

For the simple one-dimensional test function, I ran each algorithm 20 times, each time independently selecting 40 locations at which to sequentially sample the objective functions. The repeated trials are necessary because initial sample location selection is random. For the two-dimensional test function, each algorithm was run 20 times, with 20 sampling locations selected each time.

The resulting algorithm performance is shown in Figure 5.7. In each case, both ParEGO and the EIHV were shown to significantly outperform random experiment selection, demon-

[1]This aggregate is created through a random polynomial combination of the individual objectives; the random weights of the combination change each iteration.

strating the potential savings when optimizing on expensive systems. The use of EIHV also outperformed ParEGO for both tested settings of ParEGO's s-value (10 and 1000), showing an algorithm with no tuning parameters and good performance. Due to these empirical results, I chose to use EIHV when optimizing multiple objectives on the physical snake robot.



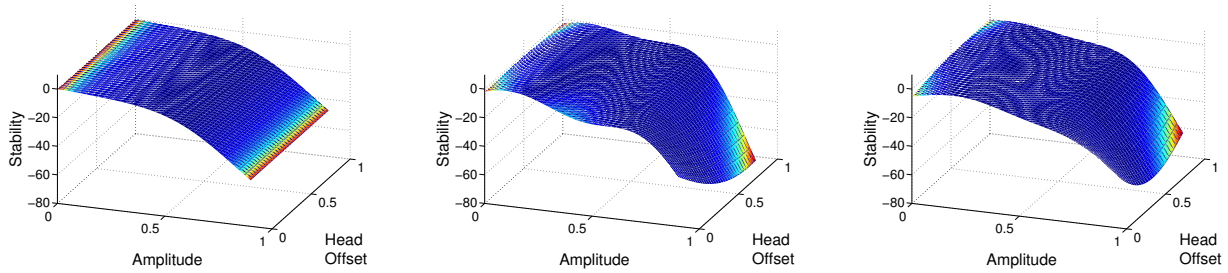
(a) (b) (c) (d)

Figure 5.7: Results of two different expensive MOO methods – optimization of the expected improvement in hypervolume (EIHV) and ParEGO – and a random experiment selection baseline on the 1-D **(a)** and 2-D **(c)** test functions shown in Figure 5.6. The line indicates the median hypervolume of the 20 trials, while the shaded regions represent the middle 50% of the trials. Also shown are the resulting Pareto frontiers from each method on each test function (**(b)** and **(d)**). On both functions EIHV outperforms ParEGO, and both methods are significantly better than random, demonstrating the importance of careful experiment selection in expensive optimization.

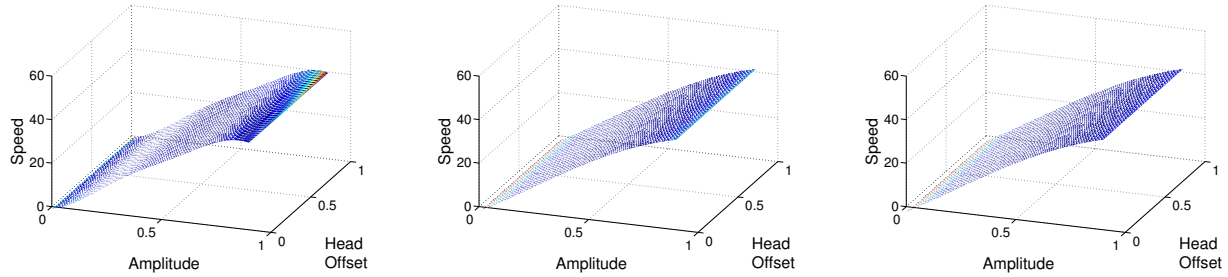
5.4 Robot Results

After the validation of EIHV in the previous experiments, I used this method to generate a set of Pareto optimal solutions for competing objectives of head stability and speed. In order to do so, I first needed to define the cost functions and the parameter space over which the optimization occurs.

The speed objective definition is straightforward – after running the snake for 10 seconds with the gait parameters as specified by the optimizer, the net displacement of the center of mass is measured. The head stability objective was more complicated. To obviate the need for a motion capture lab, I combined several tools that only relied on sensor data from on-board the robot. Intuitively, this captures a notion of how much the camera image



(a) 11th sample (b) 18th sample (c) 25th sample



(d) 11th sample (e) 18th sample (f) 25th sample

Figure 5.8: After the initial random experiments, the GP function regression method generates an estimate of the function after each subsequent sample. These surfaces are shown here for three selected iterations, where the top row is regression of the stability objective and the bottom is the regression of the speed objective. The color of the surface is the uncertainty in the estimate (blue regions in the center have low uncertainty, red regions near the edge have high uncertainty).



(a) Point A, 1.25Hz frame rate (b) Point B, 0.4Hz frame rate (c) Point C, 0.4Hz frame rate (d)

Figure 5.9: **(a)-(c)** Time lapse images showing the head and snake motion of three different points in the head stability/speed Pareto set. The snake in **(a)** moves quickly, but the head rotates back and forth. The snake in **(b)** moves at a slower rate, but with noticeably less head sway. Finally, **(c)** is a very slow but very head-stable motion. As can be seen in **(d)**, these points span the Pareto front, and none dominate the others. Shown as blue dots are the results of all experimental evaluations during the optimization.

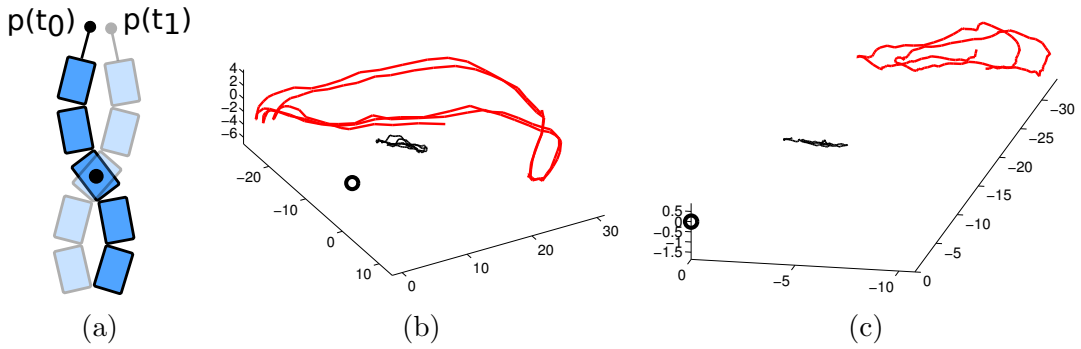


Figure 5.10: **(a)** The focal point $p(t)$ is a distance l from the head camera, normal to the image plane; it is shown here for two different timesteps. The motion of this point provides a measure of head stability. **(b)** The head stability objective used for the snake robot trials was the estimated swept area of the head camera's focal point. The data above from one of the trials shows the center of mass of the robot (black circle), the trace of the head (black line) and the trace of this focal point (red line) through multiple cycles of the commanded gait. The wide sweeps of the head result in a poor objective score of -68.6 ($V = 11.5$ and $H = 59.6$). **(c)** The same plot for a trial resulting in a more stable head trajectory; the resulting head stability objective is -2.70 ($V = 2.75$ and $H = 9.81$).

changed for an operator. Moreover, since the resulting image from a translating camera is less disorienting than a rotating camera, I chose to focus on the latter motion.

I choose a point p at a distance l of 18 inches normal to the center of the camera lens plane as the desired focal point. Using a Kalman filter based sensor-fusion state estimation technique [85], the motion of this point is estimated throughout the gait. However, because the primary consideration is orientation, I use a shape-stable body frame termed the *virtual chassis* [86], consider only the orientation component of the virtual chassis state estimate, and define the point p to be in the world frame assuming the virtual chassis is fixed in position but free to rotate in all three dimensions.

More formally, let the 4x4 homogeneous transform of the estimated orientation of the virtual chassis at the timestep t be defined by $R(t)^{VC}$. Then the position of the head frame in the virtual chassis frame at t is given by the transform matrix $T(t)^{0}_{VC}$ (where the 0 indicates the 0th snake module). The ray from the head camera module to p is of length l in the z direction, so let $p_0 = [0, 0, l, 1]^T$ be the location of p in the head module frame. Given these variables, p is defined in the fixed-position, free-rotation virtual chassis "world" frame at

time t (see Fig. 5.10a) as

$$p(t) = R(t)^{VC}T(t)^{0}_{VC}p_0. \tag{5.8}$$

To transform this focal point location into the stability cost, the minimum bounding box for this swept area is calculated. First, the vertical sweep is

$$V = \max_{0<t<T}(p_z(t)) - \min_{0<t<T}(p_z(t)), \tag{5.9}$$

where $p_z(t)$ is the z component of $p(t)$, and T is the total number of timesteps. The horizontal sweep first requires calculation of the total angular sweep from the origin to all $p(t)$ projected onto the $x-y$ plane. This should be the smallest angle for which the interior cone (from the origin) can contain all $p(t)$. This angle is H_θ, and is given in radians. The horizontal sweep is calculated as

$$d(t) = \sqrt{p_x(t)^2 + p_y(t)^2}, \tag{5.10}$$

$$H = H_\theta \sum_{0<t<T} \frac{d(t)}{T}, \tag{5.11}$$

where $p_x(t)$ and $p_y(t)$ are similarly the x and y components of $p(t)$, and $d(t)$ is the distance from the origin of $p(t)$ projected into the $x-y$ plane. Finally, the total cost – representing a bounding box of the swept position of the focal point – is given by $V \times H/10$ (see the examples in Fig. 5.10b and Fig. 5.10c). In practice I negate this value because these optimization methods maximize rather than minimize.

As the primary concerns are speed and head stability, I chose to optimize parameters of an augmented gait model that would add more position control of the head module while still keeping a low-dimensional parameter space. The basic form of the gait model $\alpha(n,t)$ in Equation (2.17) is restricted to the sidewinding parameter space as defined in [101], and an

additional offset ϕ is added for the head module ($n = 1$):

$$\alpha(1,t) = \beta_{odd} + A_{odd}\sin(\theta + \delta + \phi) \tag{5.12}$$

I then optimize over the amplitude and ϕ, with a fixed ratio between A_{odd} and A_{even}.

Finally, when running tests on physical systems, the non-deterministic nature can cause instability in many optimization methods. Although using a GP that can explicitly model this noise allows many surrogate function based optimizers to perform well in the presence of noise, I make two adjustments. First, I run each trial 5 times, averaging the results for each objective and removing outliers. Second, when determining the Pareto set for the purpose of the calculation of EIHV, I use the surrogate's estimate of the objective values for each sampled point. This improves the stability by ensuring that an artificially high sampled value will not too strongly discourage nearby samples.

The process and results of these optimization trials can be seen in Figures 5.8 and 5.9, respectively. The optimizer has sampled points which span the trade-off between both objectives, and in particular has found relatively fast motions that have improved head stability. The three montages of robot motion shown in Figures 5.9a, 5.9b and 5.9c show how this multi-objective optimization approach generates a range of solutions, whereas the typical approach of aggregating these objectives would only have found one of these solutions.

5.5 Conclusions and Future Work

Multi-objective optimization is often a reasonable alternative to creating a single aggregate objective in the case of competing system performance objectives. This is a case which comes up frequently in robotics as well as many other fields, such as design, decision theory, and economics. A Pareto optimal set should be found which contains solutions which are not *dominated*, or completely outperformed, by any other solution. The generation of

approximations to the true Pareto optimal solution set is difficult when system function evaluations are expensive, but once accomplished these solutions can be selected from to provide real-time trade-offs between objectives.

In this chapter, I have created and tested a MOO approach based on maximization of the expected improvement in hypervolume (EIHV) of the Pareto set. I have compared this to a leading MOO algorithm, ParEGO, on multiple test functions. Finally, I have applied this EI based approach algorithm to optimize snake robot gait parameters for fast and head-stable sidewinding. This application required care to reduce the effect of noisy evaluations on the optimization performance as well as the creation of a head-stability cost function from recent state-estimation techniques for the robot.

Future work involves testing these methods with higher-dimensional parameter spaces and use of these methods on other robotic systems. In addition, the explicit handling of noisy objective evaluations is an open problem. Another topic of interest is considering the EIHV metric (as well as other approaches) in the case of partial objective set evaluations, where you can choose not only which point in X to sample at but also which objectives to sample. Furthering this work can allow these methods to be used with more confidence for various applications.

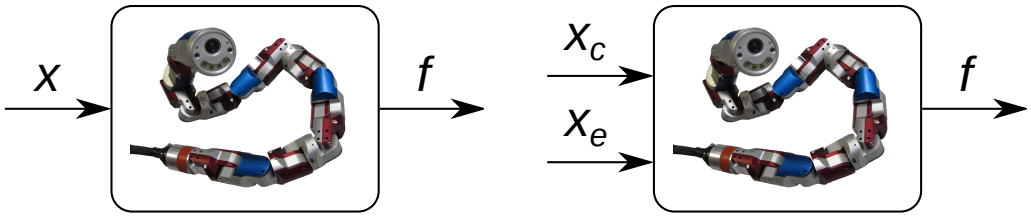
Chapter 6

Environmentally Adaptive Optimization

In the problems addressed in the previous chapters, an assumption is made that the parameters $x \in X$ of the objective function f are all controllable when an experiment is run, i.e., the objective is entirely defined by the control inputs (gait parameters, joint torques, etc.). Outside of a controlled laboratory environment, other variables (such as environmental factors) can have a significant impact on the outcome of an experiment or the behavior of a robotic system. For example, if the terrain has alternating flat and tilted or smooth and sandy sections, the robot performance (e.g. speed) will change in these different areas.

In this chapter, I explore the effects of relaxing this assumption of full control over parameters by splitting the parameter space into two subsets: first, the *control* parameters, which match the idea of the parameter space from the previous chapters, and the uncontrollable *environment* parameters, which describe the other inputs that may affect the objective, such as terrain smoothness or slope. This decomposition is depicted in Fig. 6.1.

Given this decomposition, the goal is to find *control policies* that adapt to changes in the environment by selecting the best controller parameters for each environment. The specific



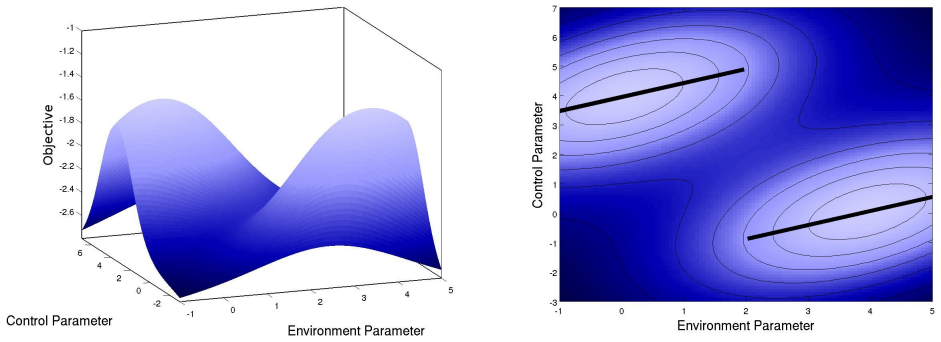
(a) (b)

Figure 6.1: **(a)** The previous expensive optimization work in this thesis fits the pattern on the left, where parameters are fed into the robotic system, which results in obtaining a value for f at that parameter. **(b)** In this chapter, the objective f is explicitly a function of both the control parameters as well as environment parameters; treating these differently allows the generation of adaptive control policies for the system.

problem I address is how to learn such a policy efficiently (a small number of tests on the robot) during an offline training phase, where the environment parameters are controllable. The resulting policy should then work effectively during testing, where environment parameters are uncontrolled; i.e., it should perform well in any environment that is encountered. This problem setup is similar to that of contextual bandits (see §3.2), where the context or side information corresponds to this notion of environment parameters. However, the focus in this chapter is on efficient learning during the training phase, while the general focus in the bandit community is on learning and minimization of regret only during the testing phase.

The optimal control policies define an optimal control parameter for each environment parameter under consideration. The optimal control parameters are often different for various environments; e.g., the best parameters for uphill motion are likely different than the best for downhill motion. For the expensive systems I focus on in this thesis, however, it is infeasible to test every possible controller in every possible environment. Even running an efficient control parameter optimization for each environment individually would be impossible.

To address this challenge, I make assumptions about the continuity of the objective. Although optimal control parameters differ for various environments, one would expect some degree of continuity in the performance of a robotic system – similar control parameters on



(a) (b)

Figure 6.2: **(a)** I am interested in problems for which the optimal control parameter changes significantly with changes in the environmental conditions. Ideally, similar environment/control combinations lead to similar system output; I therefore assume this function is continuous, although these methods can still operate with some discontinuities. **(b)** The optimal policy (dark lines that indicate mapping from environment to control) tends to be piecewise continuous; similar environments usually result in similar controllers, but there are likely to be some discontinuities. A good policy can be estimated from a low-cost model of the true expensive system output, requiring only a handful of carefully chosen points.

the same environment or similar environments with the same control parameters should lead to similar performance (Fig. 6.2). By jointly modeling the objective as a continuous function over both the environment and control parameters, *tabula rasa* learning is not performed for each environment; knowledge is shared across different environments. From my experience and the demonstrations in this chapter, this coupled optimization of controls for multiple environment drastically reduces the number of experiments required to optimize for a set of environments.

In the following sections, I show that this information-sharing approach is important, but does not provide a complete solution. The choice of experiments (points at which to evaluate the objective) can significantly affect the quality of the resulting policy. Following the theme of the previous chapters, I design *experiment selection* metrics to iteratively select parameters at which to evaluate in a manner that enables generation of the best possible policies.

In particular, I first outline the terms necessary to formally define the optimization

problem and the goals of our algorithms. I then propose two algorithms which extend the EI experiment selection metric to this new problem domain. I demonstrate the efficacy of these methods on a set of test functions, clearly showing the importance of careful experiment selection. Finally, I show results of several control policies created using these methods applied to both simulated systems and physical robots. Specifically, I demonstrate improved locomotion over changing terrain for the snake robots described in [101], as compared to a control policy generated through random sampling of the environment and control parameter spaces.

6.1 Problem Definition and Notation

This work aims to find a control policy – a map from environments to controls – that optimally adapts to external variables, while minimizing the number of experiments done to perform the optimization. The two subgoals then become:

1. **Experiment Selection:** Select subsequent parameters for evaluation, based on the previous objective evaluations, which maximize the score of the policy predicted by the policy generation algorithm.

2. **Policy Generation:** After the completion of a predetermined number of objective evaluations, predict an optimal control policy based on the samples.

In order to make these ideas more concrete, I first define several terms and then restate the subgoals more precisely.

Definition (Control Parameter)**.** The control parameter space X_c is a compact subset of $\mathbb{R}^{m_c}$. Each $x_c \in X_c$ represents a particular quantity that can be fully specified during normal operation. Some examples of control parameters include the value of a set of gains

in a PID controller, the relative concentration of two reactants in an industrial process, or the prescribed dosage of a drug during development.

Definition (Environment Parameter)**.** The environment parameter space X_e is a compact subset of $\mathbb{R}^{m_e}$. This space contrasts with the control space in that values $x_e \in X_e$ cannot be controlled under normal real-world operation, but can be specified in a laboratory setting. Furthermore, x_e can be measured during normal operation. Therefore, these parameters represent continuous valued external factors of the system, such as terrain steepness for a locomoting system, wind strength and direction for a UAV, particulate size in an industrial process, or disease strain during drug development.

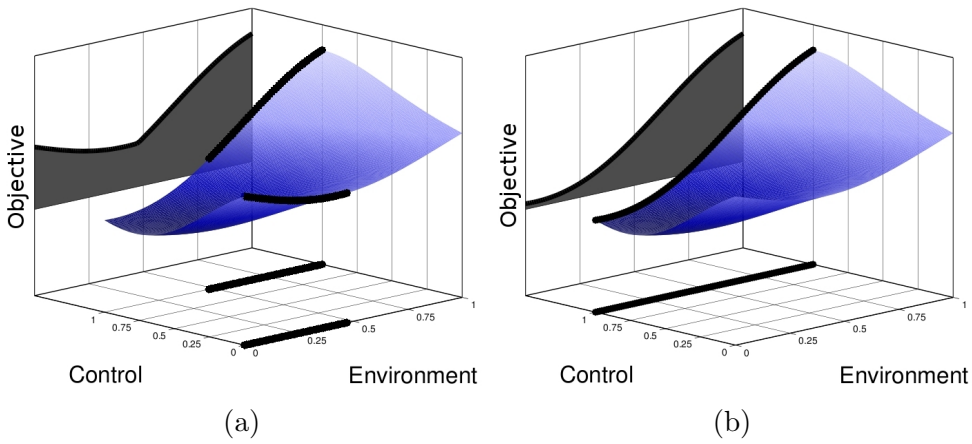
Definition (Objective)**.** The objective, denoted as $f\colon X_e \times X_c \to \mathbb{R}$, is a continuous, real-valued function of the environment and control parameters. As in the previous chapters, this function represents the performance of the system; here it depends on environmental conditions as well as the control parameter. Example objectives include the speed of a locomoting system over a terrain, the efficiency of a mechanical process, or the turbulence of a wing design calculated from a wind tunnel or computational fluid dynamics experiment. For the methods we propose here, we assume that sampling this objective is time intensive or computationally expensive, and therefore there is a limit to the number of times this function can be evaluated.

Definition (Control Policy, Score, Optimal Policy)**.** The control policy defined in this work is a mapping $\gamma\colon X_e \to X_c$, such that $\gamma(x_e)$ represents the control parameter set by γ in reaction to sensing environment parameter x_e.

The *score* $\mathcal{S}$ of a control policy,

$$\mathcal{S}(\gamma) = \int_{X_e} \omega(x_e) f(x_e, \gamma(x_e)) dx_e, \tag{6.1}$$

represents how well this policy adapts to varied environmental parameters. The $\omega\colon X_e \to$



(a) (b)

Figure 6.3: An example objective for one dimensional environment and control spaces. The policy γ is shown below the function and its performance is projected on the objective/environment plane. The policy score is the integral of this gray shaded region. An optimal policy is illustrated in **(a)**, presenting the best control parameter for every environment parameter. Note that the policy shown in **(b)**, which also maps to a control parameter that maximizes the objective at $x_e = 1$, results in a significantly lower overall score because of its poorer choices in other regions of the environment space.

$\mathbb{R}^+$ term is an optional weighting function that reflects the relative importance of learning controllers for various environments.

The optimal control policy for the system, γ^*, is defined as the policy which maximizes $f(x_e, \gamma(x_e)) \; \forall \; x_e \in X_e$. In other words,

$$\gamma^* = \underset{\gamma}{\operatorname{argmax}} \, \mathcal{S}(\gamma). \tag{6.2}$$

Note that γ^* is independent of ω, because $\gamma^*(x_e)$ can be independently determined for each $x_e \in X_e$. Although the weighting function can make a significant difference during experiment selection and ranking of sub-optimal policies, the optimal policy is unaffected by the relative importance of different environments.

To illustrate these ideas, I use one-dimensional environment and control spaces (Fig. 6.3). In Fig. 6.3a, the optimal policy γ^* is shown on the control-environment plane, and the score integral is visualized on the objective-environment plane; Fig. 6.3b shows a suboptimal policy. Note that the objective is assumed to be continuous, whereas the control policy does not

have this restriction.

The goal of this work is to find the highest scoring γ after a number of objective evaluations. As above, I break this problem into *policy generation* and *experiment selection* subproblems, stated here more formally:

1. **Experiment Selection:** Iteratively choose the sequence of points $X = \{x^1, x^2, \ldots x^n\}$, $x^i \in X_e \times X_c$, where the choice of x^{k+1} is informed by $\{f(x^i) \mid i \leq k\}$, which maximizes the score of the policy produced by the chosen policy generation algorithm.

2. **Policy Generation:** Given the results of n objective evaluations, choose the best prediction for the function γ^*.[1]

6.2 Proposed Methods

The difficulty of applying a standard optimization technique to this problem is the fact that we cannot evaluate the true function being maximized during policy generation, $\mathcal{S}$. This is because evaluating $\mathcal{S}(\gamma)$, the score of a single policy, involves an integral of the expensive objective $f(x_e, x_c)$. As the policy generation task admits no new evaluations of f, and experiment selection only allows n evaluations, it is clear that it is impossible to calculate $\mathcal{S}(\gamma)$ for any single policy γ, let alone apply a standard optimization technique to $\mathcal{S}$. Rather than directly searching the infinite-dimensional policy space, the problem setup requires selection of a series of $x^i \in X_e \times X_c$ which will yield the information necessary to build a good policy.

The first of these problems is addressed in this approach by using a surrogate function, $\hat{f}$, to model the objective and make decisions. This technique is widely used in global optimization of expensive functions, but the surrogate usually directly represents the function

[1] γ^* need not be representable in any given functional form; I store the result at a dense grid of points and retrieve the closest neighbor upon evaluation.

to be optimized (as in previous chapters). Here, I instead use the surrogate to represent a function that is an intermediary in the computation of our score function. However, this shares with the expensive global optimization community the key idea of replacing an expensive function with a surrogate, using the surrogate to make informed decisions, and then updating the surrogate when new information is available. Again, I use GPs for this surrogate $\hat{f}$.

The same caveats as were described in §4.2.4 apply here; the high level algorithms cannot be expected to perform well if the surrogate is not a reasonable representation of knowledge about the function. If the hypothesis space is not rich enough to capture behavior of the objective, or the hypothesis space is too rich, the model could under- or overfit the data, resulting in poor algorithm performance. The experimental results in this chapter use the squared exponential covariance as the GP's covariance function, but the methods described below do not rely on any particular choice of covariance; in general I recommend selecting among model complexities (for example, different covariance functions for the GP) as in §4.2.2. In addition, any prior knowledge about the objective should be encoded in the model prior.

Unsurprisingly, standard local and global optimization methods directly applied to optimizing the objective f perform poorly. This is because these methods are not optimizing the true quantity of interest – the score; rather they focus evaluations in environments with high objective results, and ignore environments with low objective results. The sparse evaluation of "difficult" environments (ones with low objective results) results in $\hat{f}$ having a high uncertainty in these regions. In turn, finding near-optimal control parameters (and hence high score contributions) from these environments is less reliable. This can result in a lower overall score. However, I have included results from using a straightforward EI search as a baseline approach.

To respond to the bias against "difficult" environments encountered by this straightfor-

ward optimization of f over $X_e \times X_c$, one might choose to incorporate a method such as information gain, uniform sampling, or maximum dispersion point selection to ensure that all types of environments are equally sampled. Unfortunately, these methods cause poor regions of the control space to be sampled at the same frequency as good regions of the control space, resulting in an ineffective use of the limited sample budget. I have included a uniform random point selection algorithm as another baseline to demonstrate this ineffective sampling behavior.

These baselines amount to heuristic approaches for this problem, and although heuristics can often perform well, I seek a more principled solution. The methods proposed in §6.2.2 and §6.2.3 attempt to provide a tractable solution which maximizes a statistical quantity related to the score function: approximate expectation of improvement above the current predicted policy score.

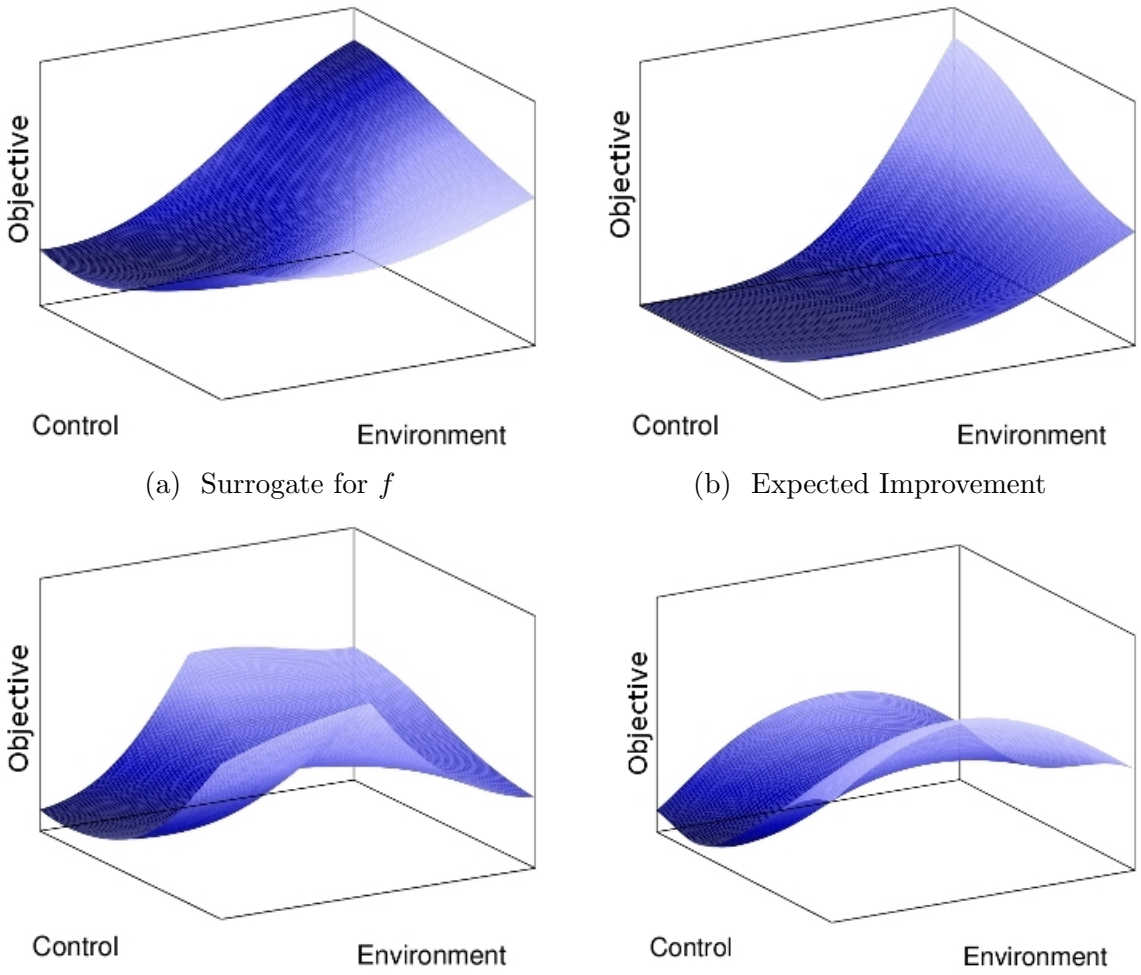
6.2.1 Policy Generation

Using the assumption that the posterior mean of the GP, $\hat{f}_\mu$, is the best estimate of the true function, the appropriate course of action is to select the policy which maximizes the estimated score,

$$\hat{\mathcal{S}}(\gamma) = \int_{X_e} \omega(x_e) \hat{f}_\mu(x_e, \gamma(x_e)) dx_e. \tag{6.3}$$

Therefore, I choose $\hat{\gamma}^* = \text{argmax}_\gamma \hat{\mathcal{S}}(\gamma)$.

Alternatively, one could also choose to take a low-risk approach, and use a lower confidence bound of the surrogate function to define $\hat{\mathcal{S}}$. We define $\beta \in \mathbb{R}^+$ as a risk-averseness parameter. Using $\hat{f}_\mu - \beta(\hat{f}_{\sigma^2})^{1/2}$ instead of $\hat{f}_\mu$ in Equation (6.3) biases the generated policy towards control/environment combinations that perform well (high expected mean value) but also which the surrogate is confident in (low predictive variance). This reduces the likelihood that the policy will choose a controller that will perform extremely poorly in an



(a) Surrogate for f

(b) Expected Improvement

(c) Unbiased Expected Improvement

(d) Expected Policy Score Improvement

Figure 6.4: **(a)** A surrogate for the objective f; the intensity represents the confidence (dark = high, light = low). **(b)** The value of a potential experiment using EI. The EI metric biases point selection towards environments with high objective values. **(c)** The value of a potential experiment using EI over the best predicted objective value for that environment. This reduces this bias toward easy environments and more directly optimizes the policy score. **(d)** The estimated policy score improvement as a result of sampling each point. This selection criterion is computationally intensive, but results in better performance than (c).

unexplored region of the parameter space, but the tradeoff is that this conservative approach avoids policies predicted to be high performing if it has low confidence in them.

As this work mainly focuses on experiment selection methods and typical expensive optimization problems are fairly low-dimensional, I maximize Equation (6.3) to estimate $\hat{\gamma}^*$ via a dense sampling of the relatively cheap $\hat{f}_\mu$. If needed, more efficient approximations could be applied for this optimization subproblem.

6.2.2 Experiment Selection: Unbiased Expected Improvement

Recall that EI has proven to be an effective experiment selection metric in global optimization problems (§2.4, §4.2). This metric seeks to provide a balanced method for trading off exploration of the search space and focus on the best regions discovered so far ("exploitation"). I restate Equation (2.22) here as

$$\mathrm{EI}(x) = \int_{\max(\tilde{\mathbf{y}})}^{\infty} p_Y^x(y)(y - \max(\tilde{\mathbf{y}}))\, dy. \tag{6.4}$$

Although the use of EI has been successful for many expensive optimization problems, a naïve application of this approach to select $x^k \in X_e \times X_c$ as $\mathrm{argmax}_X \mathrm{EI}(x)$ produces poor results for control policy generation. As the algorithm quickly finds the maximum regions of f, environments which have no control parameter that performs very well will not be explored at all. Instead, the algorithm biases its search towards environments that have high values for the objective (Fig. 6.4b). This results in a policy that performs well in "easy" environments, but suboptimally in "difficult" environments.

The approach proposed here adapts this basic idea to the explicit separation of the environment and control space. I consider the expected improvement of sampling at some $x^t = (x_e^t, x_c^t)$, but measure improvement over the maximum estimated value *for that environment* (when $x_e = x_e^t$), or $y^*_{x_e^t} = \max_{x_c \in X_c} \hat{f}_\mu(x_c, x_e^t)$, instead of over the best objective

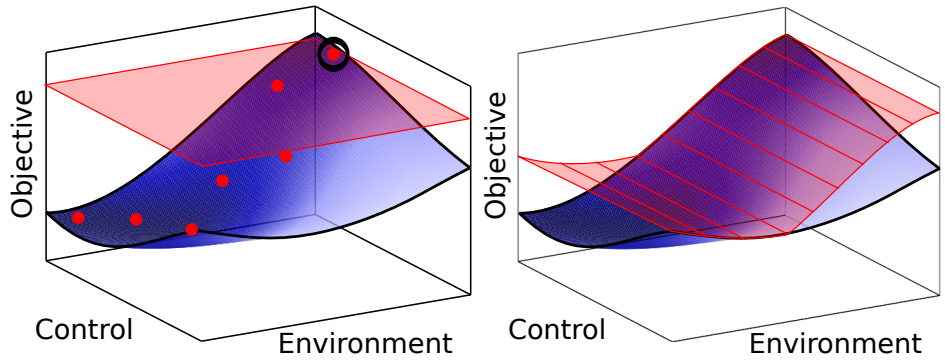


Figure 6.5: **(a)** Standard EI measures improvement over the best sampled experiment, the value of which is represented by the horizontal red plane. This biases against sampling in difficult environments, such as those towards the left of the environment axis. **(b)** Unbiased expected improvement reduces this bias by measuring improvement over the maximum of the estimated objective *for each environment*; the objective is shown below in blue, and the maximum is shown above in red.

evaluation so far, $\max(\tilde{\mathbf{y}})$. This gives the Unbiased Expected Improvement (UEI), and is illustrated in Fig. 6.5:

$$\mathrm{UEI}(x^t) = \omega(x_e^t) \int_{y^*_{x_e^t}}^{\infty} p_Y^{x^t}(y)(y - y^*_{x_e^t})\, dy. \tag{6.5}$$

The x^k chosen by this algorithm is then given by $\mathrm{argmax}_X \,\mathrm{UEI}(x)$.

By applying this method, the inherent bias towards environments containing high values for f (Fig. 6.4c) is removed. Since the calculation of $\hat{\mathcal{S}}$ involves integration of the performance of the controller $\gamma(x_e)$ over all environments, the UEI measures how a sample at (x_e^t, x_c^t) would contribute to improvements in the policy score $\mathcal{S}$ that we wish to maximize, rather than the intermediary objective f.

6.2.3 Experiment Selection: Expected Policy Score Improvement

Although the UEI method begins to approximate improvement of the true policy score function $\mathcal{S}$, there is one crucial limitation. The improvement to the score calculated by

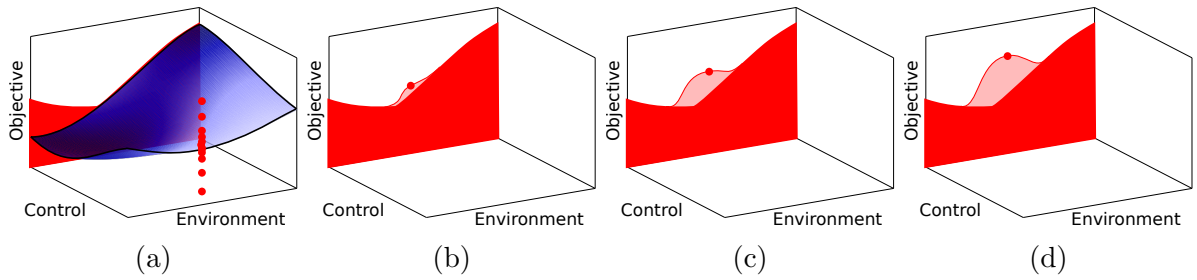


Figure 6.6: EPSI provides a measure of the potential improvement to the score of the estimated optimal policy $\hat{\gamma}^*$ resulting from sampling at some x^i. **(a)** Sampled values from the distribution at $\hat{f}(x^i)$ are shown; this is a sampling of the GP posterior at this point. The policy score given $\hat{f}$ is the shaded region on the objective/environment plane. **(b) - (d)** New estimates of the policy score are generated by updating $\hat{f}$ with these samples. The lightly shaded regions show the predicted improvement over $\mathcal{S}(\hat{\gamma}^*)$, the estimated optimal policy score. The EPSI is the weighted combination of these policy score improvement contributions.

$\text{UEI}(x^t)$ only considers improvement at one point in the environment space, independent of the other environments, and therefore is only an infinitesimal element of the true expected improvement of the policy score. However, sampling a point will add information to $\hat{f}$ about an entire region, not just a single point. To approximate the full expected improvement, we must find a way to extrapolate the effect of sampling a single point to an entire region. To measure this effect, I generate new GP surrogate functions for every possible objective value y in the posterior predictive distribution of a point, and integrate the improvement to the policy predicted by each new surrogate (see Fig. 6.6).

Using this method generates a more complete estimate of the effect of sampling a point on the policy score. Of course, computing a large numeric integral can also take significantly more time, and the quality of the solution can vary based on the resolution of the integral.

More formally, let us define the Expected Policy Score Improvement (EPSI) at a test point $x^t = (x_e^t, x_c^t)$ as:

$$\text{EPSI}(x^t) = \int_{-\infty}^{\infty} p_Y^{x^t}(y) \int_{X_e} \omega(x_e) \max\left(\hat{f}_{\mu_y}(x_e, \hat{\gamma}_y^*(x_e)) - \hat{f}_{\mu}(x_e, \hat{\gamma}^*(x_e)), 0\right) dx_e dy, \qquad (6.6)$$

where $\hat{f}_{\mu_y}$ is the mean of the surrogate function conditioned on the addition of a sample

at x^t with value y, and $\hat{\gamma}_y^*$ is the optimal policy generated using $\hat{f}_{\mu_y}$. The point chosen for evaluation by this algorithm is $\text{argmax}_X \text{EPSI}(x)$.

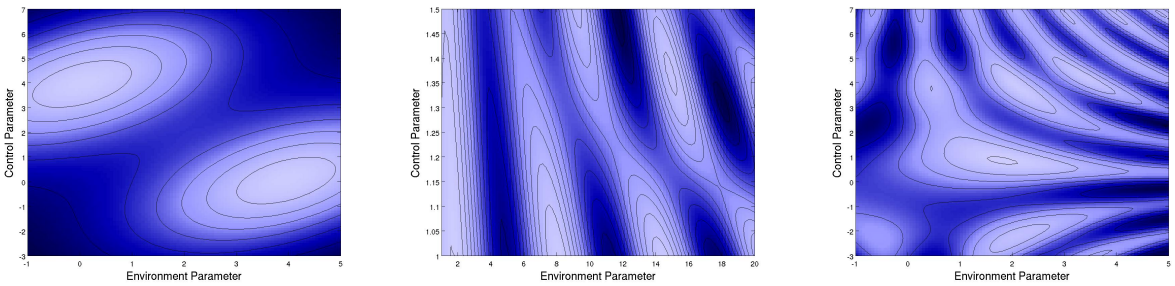
By maximizing Equation (6.6), we are choosing a point to evaluate which maximizes expectation of improvement in the policy score function, rather than choosing a point which maximizes improvement in one differential element of the policy score integral. An example of this criterion is shown in Fig. 6.4d.

6.2.4 Method Comparison and Discussion

As the UEI uses an analytic expression to calculate expected improvement, it is relatively quick to calculate but only considers one infinitesimal element of the policy score integral (albeit arguably the most significant element). EPSI provides a more complete approximation, but requires a numeric double integral, for which the integrand requires conditioning the GP.

This produces a resulting selection surface which is smoother with respect to the environment parameter as compared to UEI (Fig. 6.4). Qualitatively the locations of the maxima chosen by each algorithm are similar. However, the EPSI approach gives a more complete approximation of the true policy score improvement. This results in improved performance but suffers from a computationally burdensome numeric integral which may diminish its usefulness for some applications.

One notable difference in the computation of the EI used here and that typically used for global optimization is that here the improvement over maxima of our surrogate function (an approximation to the objective) is calculated, whereas other approaches calculate this improvement over the best previous objective evaluation. I calculate improvement in this manner because a randomly selected environment has zero probability of containing a previously evaluated point. The full implications of this difference are not explored in this thesis.



(a) (b) (c)

Figure 6.7: Three analytic test functions designed for the comparison of point selection algorithms. The exact formulae consist mainly of a collection of trigonometric functions, and are given in Appendix A.

6.3 Experimental Results

To evaluate the performance of these algorithms I first used synthetic "test functions" in lieu of an objective from a physical system, which allowed the completion of enough tests to enable one to draw reasonable conclusions about relative algorithm performance. The algorithms were also tested on objectives from real physical and simulated systems; they were used to learn policies $\hat{\gamma}^*$ (as previously described in §6.2.1) which were then evaluated on a course with changing environment parameters.

6.3.1 Synthetic Test Functions

I created three test functions of varying complexity to use for f, shown in Fig. 6.7, which could be used to compare point selection algorithms. Specifically, these functions were created such that a static control policy would not be effective over the entire space. They also mimic qualitative properties I have noted in objectives from the physical robot, such as periodicity and multiple local optima.

To measure the performance of each algorithm, an initial set of k points is non-deterministically generated through a high-dispersion sampling method such as a Latin hypercube [66], and then use the algorithm to sequentially select $n-k$ more points, evaluating the objective after

each choice. The predicted $\hat{f}$ is used after each objective evaluation to generate a policy, which is scored via a numeric integral on the objective. This entire process was repeated 10 times for each algorithm, the results averaged, and standard errors calculated to provide a rigorous comparison of methods.

Although the algorithms described herein do not have any intrinsic parameters to tune, there are several implementation details which must be considered. Specifically, the choice of covariance function for the GP, the sampling strategy, the sampling density, and the density of the numeric integrals all can have an effect on overall performance. Therefore, to keep the comparison as fair as possible, I kept these choices constant when comparing the algorithms. In particular:

- The squared exponential covariance function was used for the Gaussian process.
- A grid of points was used as the set of all points at which to evaluate the metric. The location (offset from the origin) of this grid was defined by independent random offsets in each parameter, which were recalculated for the selection of each subsequent point x^i.
- Varying densities of this grid, from 10 to 40 points in the environment parameter space and 10-20 points in the control parameter space, were compared. Each algorithm was run for 10 trials at each grid density. The inner numeric integral of the EPSI method was a summation over the environmental components of this grid.
- The outer integral of the EPSI method was run at various resolutions, and with varying limits: integration limits of ± 3 and ± 4 standard deviations were both used, each with sampling density of 31 and 61 points. These values were selected after conducting a small study of the effect of limits and sampling density on the accuracy of the numeric integration of the expected improvement metric.

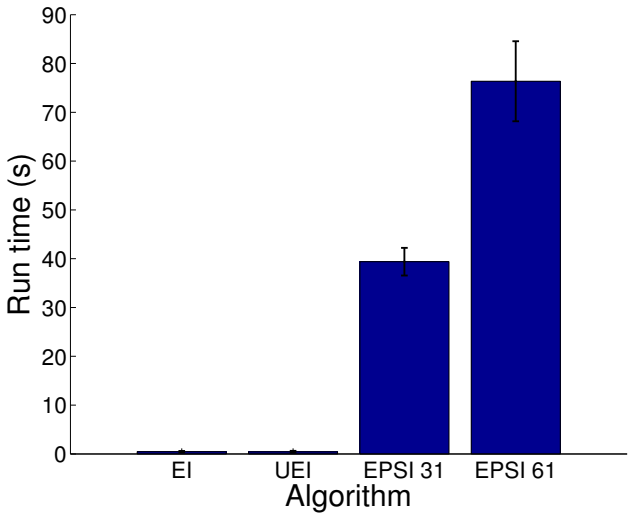
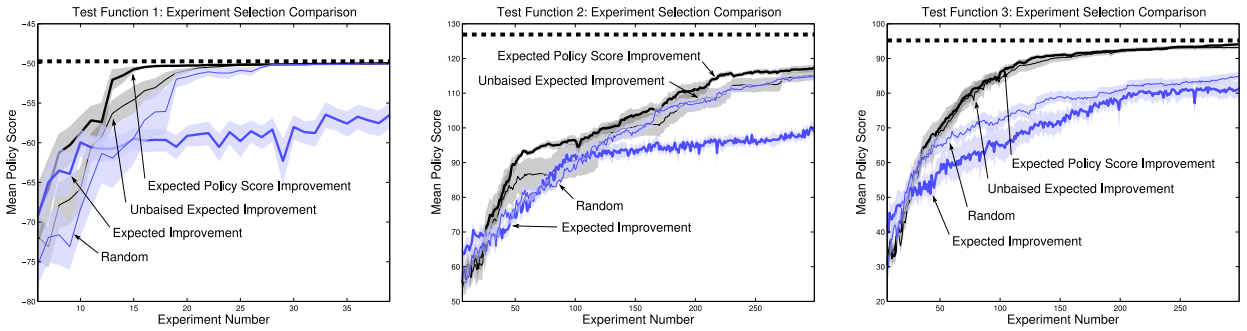


Figure 6.8: The average run time for iterations 31-40 of three separate runs of standard EI, UEI, and EPSI (with numerical integral resolutions of 31 and 61), using the MATLAB code provided with this thesis run on a 2.8 GHz Core i7-2640M processor. For the simple implementations used, the run time of UEI is inconsequential compared with EPSI, and doubling the numerical integral resolution of EPSI doubles the computational time (as would be expected). The error bars provide a 95% confidence interval for each algorithm.

We reiterate here that the EPSI method had greater computational requirements given the same implementation choices, however in many applications with real systems the time required by the algorithm is insignificant compared to objective evaluation times. In Fig. 6.8, typical computational run times are shown for comparison; these tests were run on a Core i7 processor.

A summary of the results of these methods is shown in Fig. 6.9. As expected, random point selection performs suboptimally, showing that it is important to carefully select experiments. However, as uniform coverage is probabilistically guaranteed, random selection results in continual policy improvement. Standard EI also shows an unsurprising trend, as it is biased towards "easy" environments: initially, performance is comparable to the other algorithms, as it seeks out the best area of $X_e \times X_c$. However, the overall expected improvement is low in regions of X_e that do not have high values for f, and so policy score tends to stagnate quickly, ignoring potential score function improvements from sampling in these



(a) (b) (c)

Figure 6.9: Comparison of algorithms on the three analytic test functions shown in Fig. 6.7. Each algorithm was run 10 times, and the scores averaged. The dotted black line represents the best possible policy for that function. EPSI performed best, followed by UEI. Expected improvement selected initial experiments intelligently, but its inherent bias lead to poor overall performance. Random point selection suffered from no such bias, and resulted in steady policy score improvement with increasing experiments. Each line represents the mean of 20 trials, and the shaded regions indicate ± 1 standard error.

regions.

The UEI performs much better, eliminating this bias of standard EI. This suggests that it is a reasonable, simple choice to use for tackling such point selection problems. Finally, the EPSI algorithm improves upon UEI, but only slightly. In all of the results, it was always shown to be at least as good, but often the margin of improvement was very slight. This indicates that although EPSI potentially gives a better approximation to the expected improvement of the policy score, UEI provides a much simpler and quicker method which produces similarly high quality results. The final choice between these methods involves several factors, and is largely application dependent.

While not strictly algorithm parameters, implementation decisions can have significant effects on performance. Fig. 6.10a illustrates that a more dense sampling of candidate points improves the average generated policy's performance, relative to the start of the test, by as much as 50% (near experiment 75) for the UEI algorithm. During computation of the numeric integrals of the EPSI algorithm, this grid size was the resolution of the integral over the environment space (see Equation (6.6)), and was also shown to have a notable effect (see

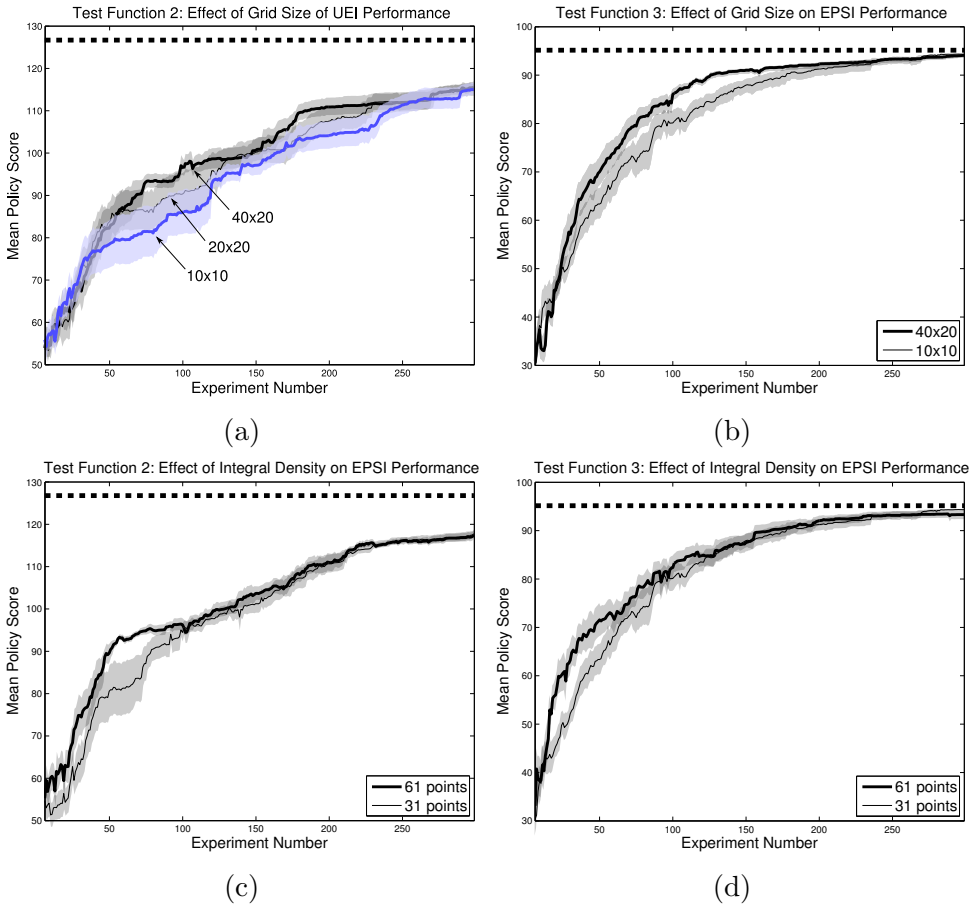


Figure 6.10: A comparison of the effect of grid resolution (top row) and numeric integral density (bottom row) on algorithm performance. In general, higher density candidate point grids and numeric integrals resulted in improvements in the quality of results. All EPSI results shown use ± 3 standard deviations of $p(y)$ as the limits to the numeric integral. Each line represents the mean of 20 trials, and the shaded regions indicate ± 1 standard error.

Fig. 6.10b). The resolution of the integral over the predictive distribution for EPSI can be a limiting factor as well, as seen in Fig. 6.10c and 6.10d. The dependence of EPSI on high density numeric integrals is one of the most significant limitations of this algorithm.

6.3.2 Simulation and Physical System Results

As such a complete analysis could not be run on physical systems due to the expensive nature of objective evaluations and the inability to compute a true policy score, I instead set up a

range of environmental conditions in a "test course," and then used a subset of the above algorithms (the best proposed algorithm, EPSI, as well as a random point selection baseline) for experiment selection. Policies were then generated and scored on this test course. These policies map environment parameters (slope and crevice width) into a 2-D gait parameter control space (see [101]).

Two test courses were defined, one for a simulation of a snake robot and one for the physical mechanism in Choset's lab [108]. The tests involved crawling through a crevice and crawling up slopes of varying steepness. The effects of transitions between environments were not considered when generating the control policies; therefore during evaluation on the test course the terrain and parameter changes were instantaneous (when using the physical snake this was accomplished by pausing the test in order to change these parameters).

Finally, the additional difficulty of noisy function evaluations is encountered when working both with the robot and with simulations. Although there is no explicit mention of noise in the algorithms, an appropriate choice of GP covariance function attempts to characterize this by fitting a noise parameter as well (c.f. §4.2.3). This allows the algorithms to remain effective even in the presence of stochastic objective evaluations.

Results of evaluation of the policies generated during testing are shown in Fig. 6.11. Overall, EPSI caused superior policies to be generated, as compared to uniform random point selection. This comparison is not as unfair as it might seem; in the tests on synthetic functions, random performed second only to the proposed algorithms, because standard approaches are not appropriate for this problem. The difference can be noted even after only 10 samples of the space (the first 5 of which are the randomly generated initial sampling). These results also show that using surrogate functions still carries risk, as a bad surrogate function fit can result in a bad policy, such as that seen after 20 experiments from the simulated snake. This result suggests the use of the low-risk policy generation method described in §6.2.1 when few sampled points are available, and a less conservative method

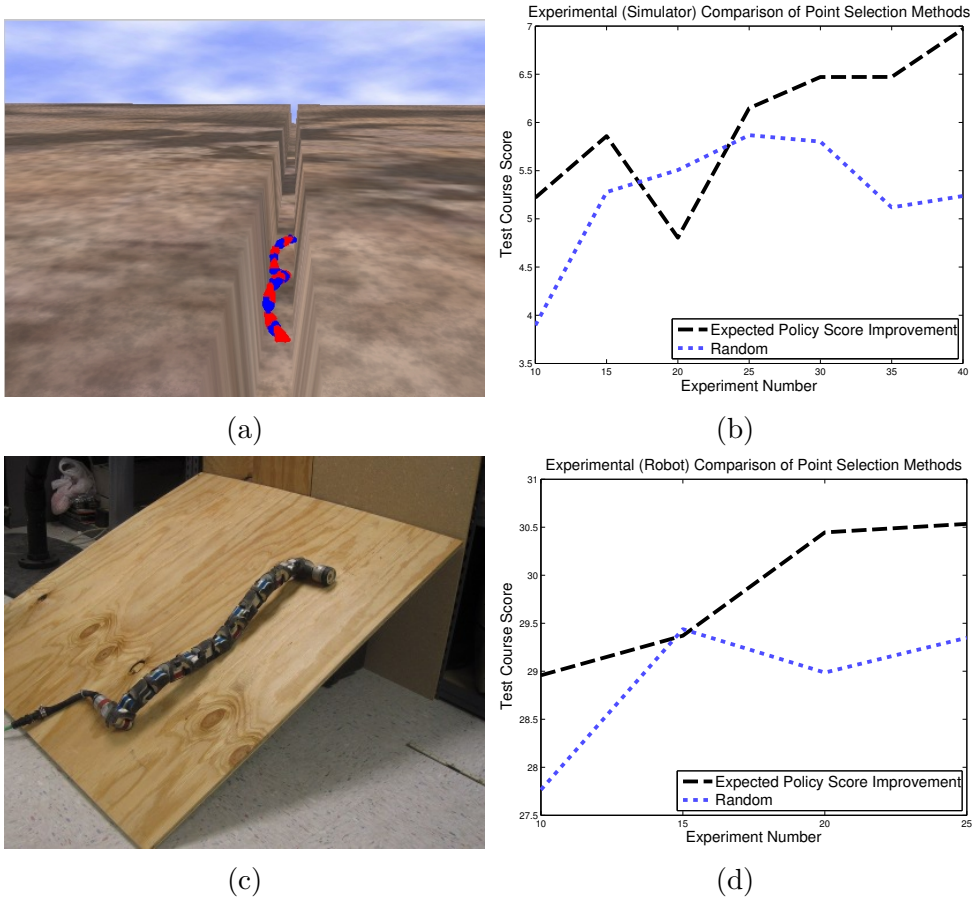


Figure 6.11: Performance of policies generated from points selected randomly versus using the EPSI method. In the top row, a simulated snake robot crawls through a crevice of varying width using a helical rolling motion. The system performance is a measure of locomotive energy efficiency; high amplitude controllers do well in wide cracks but waste energy in small cracks. The bottom row shows a physical robot climbing up an incline; higher amplitudes work well for flat ground, but smaller amplitudes allow the robot to climb steeper inclines without slipping backwards. In both cases, 5 initial experiments were selected randomly before use of the point selection method. Candidate points were chosen from a grid 40 environment points by 20 control points. Only one trial was conducted for each setup.

when more data is available.

6.3.3 Application for Real-Time Control

In the above experimental results, the parameters of the test environment are explicitly given, so the system does not need to estimate the environment parameters. Also, transitions between environments with different parameters are ignored; the experiments consist of several independent tests on environments with different parameters. In the following experiments[2], I have also considered estimation and transitions by running the robot over a continuous test course consisting of segments with different environment parameters; no knowledge of these parameters is given to the system and so it must determine the parameters and then react appropriately through a learned policy. The policy learning is completed using the UEI approach described above.

For the experiments in this section, the test course consists of four panels of plywood measuring 4' by 4', hinged together in series (see Fig. 6.12). The joined ends are raised to different heights to produce four different slopes; the angle between adjacent panels is limited to ± 30 degrees. The robot is placed at one end, with the goal of moving over the four panels to the other end in the shortest amount of time.

Learning the Adaptive Policy

The training setup used to evaluate choices of x_e and x_c can be seen in Figure 6.13. The parameter ranges included a slope from -25 to $+25$ degrees for x_e, and a sidewinding horizontal amplitude of 0 to 0.9 for x_c. The sidewinding gait was used as described in [101]. The ratio of horizontal to vertical amplitude was restricted to 8/3, and the spatial period was fixed at one and a half wavelengths over the length of the robot (16 modules).

[2]These experiments were joint work with Chaohui Gong, and the slope estimation procedure described below was developed by Chaohui; it is included here to provide a full description of the test setup.

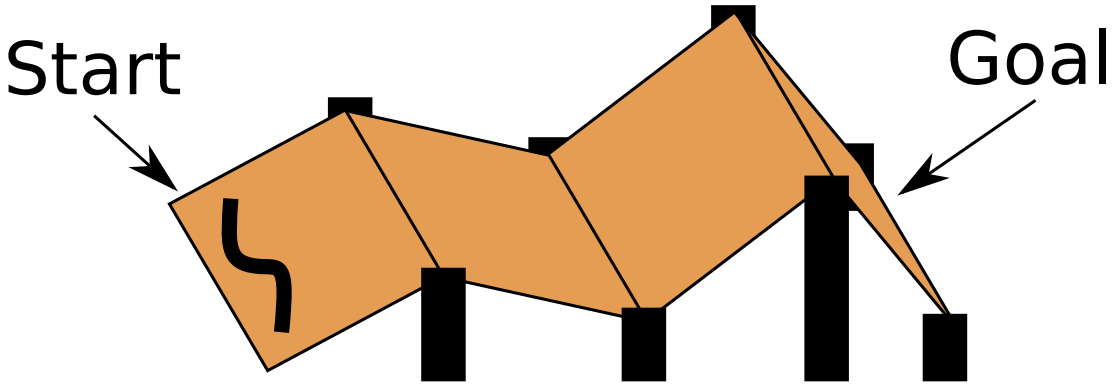


Figure 6.12: Diagram of test course. The robot was placed at the start location to begin, with the task of reaching the goal location in the minimum amount of time. The slope of each segment can be set independently, and is estimated in real time as the robot moves along the segment.

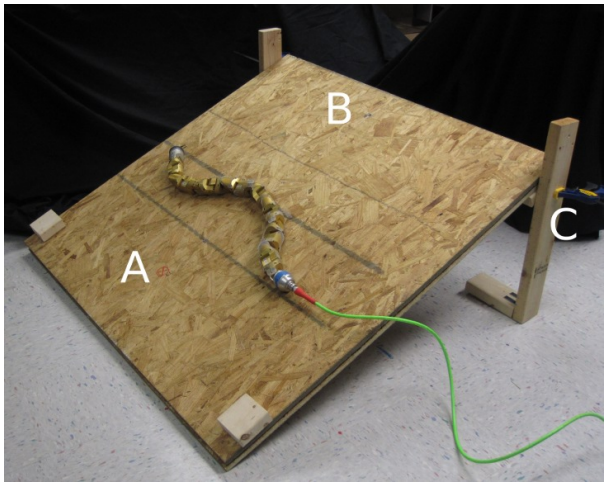
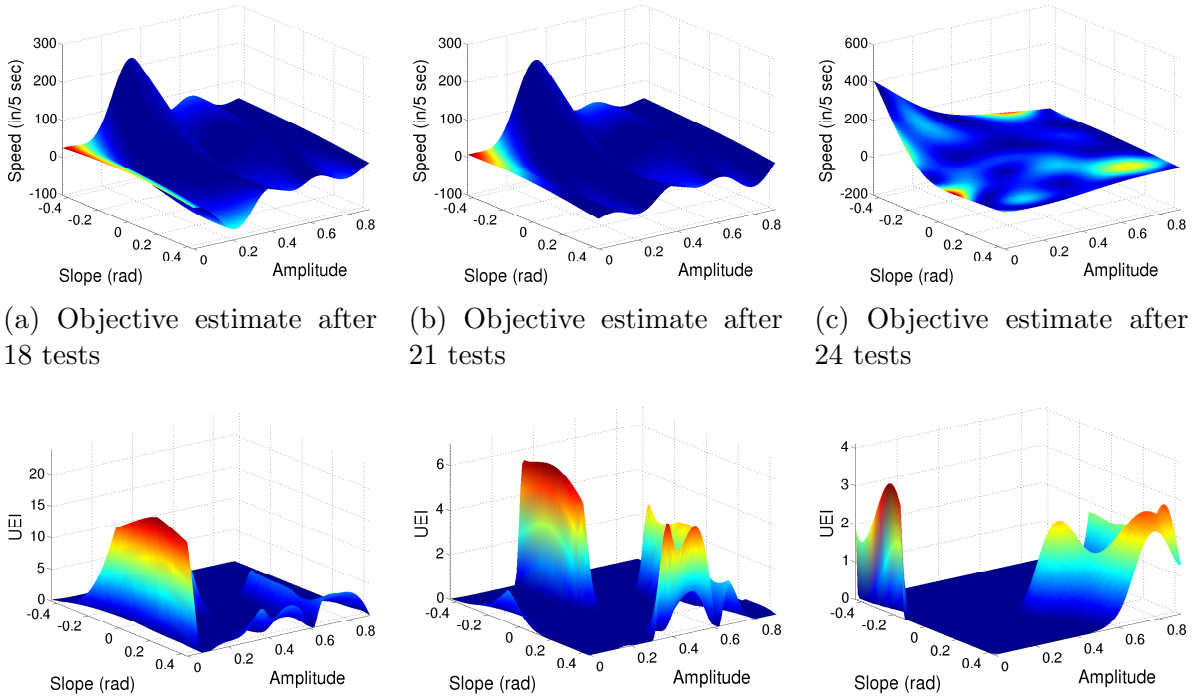


Figure 6.13: The training setup consisted of a 4 foot by 4 foot plywood sheet. The robot began at a small circle at point A for uphill trials, and one at point B for downhill slopes. The height of the test setup was changed when requested by the optimization algorithm by adjusting one quick clamp on each side of the plywood sheet, and clamping on to small blocks of wood attached to the underside of the sheet. The distance travelled was measured by comparison of the snake's position at the end of ten seconds to ruled markings drawn on the wood.

For this task, I define $f(x_e, x_c)$ as the dot product between the desired direction of travel for slope x_e (either up or down in the direction of the gradient) and the vector from the start point of the robot's center of mass to the end point after 10 seconds of gait execution with parameters x_c. Note that negative values are reported if a robot travelled a negative distance (such as rolling back down the slope after tipping over while travelling upwards). When the robot exceeded the allowable range (such as fast rolling motions downhill), the robot was timed and a constant velocity assumption was used to extrapolate downhill distance. When collecting data, I chose to reduce the inherent noise by running each sample 5-8 times (outliers were removed, and extra samples were not taken when data was extremely consistent from the first samples).

In the experiment selection process, 25 slope/control parameter combinations were tested; after an initial 10 random points, the UEI experiment selection method selected the next 15. After each evaluation, Equation (6.3) was used to generate a policy $\hat{\gamma}^*$ (as previously described in §6.2.1). As an illustration of the optimization process, Figure 6.14 shows the predicted reward function and resulting experiment selection metric during a sub-sequence of the samples that were taken. The entire distribution of the sample locations is shown in Figure 6.15a. Note that random selection would not define the boundaries of the policy as well, and that simple optimization of the reward over all slopes would focus towards the overall peak, resulting in a policy performing well only for slopes near that peak. The adaptive optimization approach seeks and finds a policy which performs well on all slopes.

Finally, the resulting optimized policy is shown in 6.15b. Note that it can be discontinuous, as its value (the control for a given slope) at each slope is determined by running any standard optimization method over the (cheap to evaluate) "slice" of the GP's predicted surface for that slope; this is independently done for each slope, and therefore allows for jumps in the optimal control between one slope and another. In my implementation, I ran a set of such optimizations over a dense sampling of the slopes using the GP (this only took



(a) Objective estimate after 18 tests

(b) Objective estimate after 21 tests

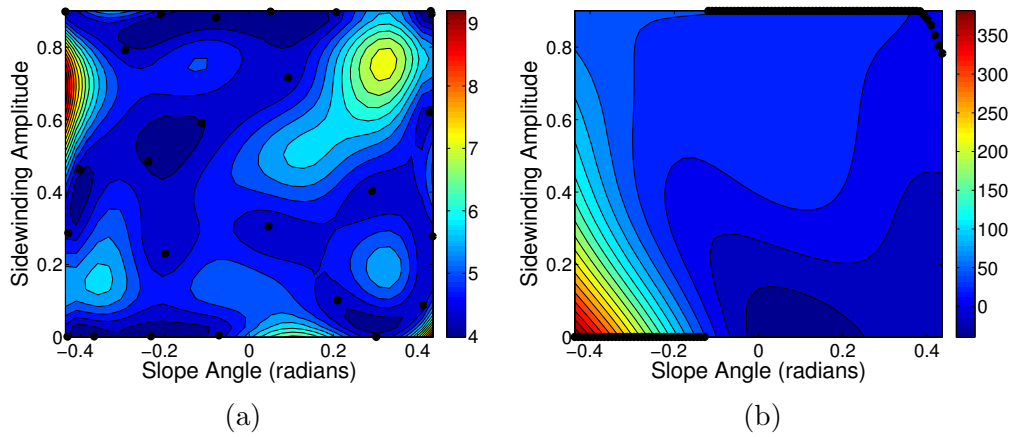
(c) Objective estimate after 24 tests

(d) UEI selection metric after 18 tests

(e) UEI selection metric after 21 tests

(f) UEI selection metric after 24 tests

Figure 6.14: The top row shows the changes in the estimated objective $\hat{f}$ as the number of samples increases. The height of the surface indicates the predicted value, whereas the color indicates the uncertainty. The left corner (negative slope, low amplitude) has not been sampled and has a high uncertainty after 18 and 21 experiments; after 24, however, the function has been sampled at this location causing a significant change in the surface prediction. The bottom row shows the value of the selection metric, the unbiased expected improvement. As opposed to a simple optimization of the space, samples are still collected for slope angles which have no high predicted speed; this allows the algorithm to learn a policy that performs well for all values of the environment parameter rather than only the "easy" environments.



(a) (b)

Figure 6.15: GP posterior pointwise variance and mean after sampling budget has been exhausted. **(a)** Uncertainty in the final objective function estimate. The black dots in the figure are the 25 sampled points. The uncertainty is highest furthest from the sampled points. **(b)** The mean predicted objective. The optimal policy is shown by the large black dots; at steep downhill slopes, the optimization determines that a zero-amplitude sidewinding gait is most effective (this gait rolls down the hill very quickly). At a critical slope, the optimal control switches to a high amplitude sidewinding gait. At very steep slopes the optimal control parameter decreases slightly; perhaps this decreased amplitude increases the stability or repeatability of the motion.

around 3 seconds on an Intel Core i7 processor), and stored these values in a simple lookup table so they could be retrieved later during testing.

Real-time Slope Estimation

In order to react to changing environmental parameters, the robot must be able to sense those parameters. For this task, the accurate estimation of the slope angle is critical. This is accomplished by first estimating the pitch angle of the *virtual chassis* relative to a spatial reference frame, and then inferring the slope angle from this estimated pitch.

The virtual chassis [86] is a *shape-stable body frame* for mechanisms with many degrees of freedom. Rather than rigidly attach a body frame onto one of the links of the robot, this body frame captures a more intuitive notion of the robot's position. The origin of the virtual chassis is located at the center of mass of the robot, and the orientation is aligned with the principal moments of inertia. This is of particular use for estimation of the slope, because

one can make assumptions that the largest two principal components of the virtual chassis form a plane parallel to the ground (following the intuition that this is the "flat" part of the robot).

Using the state estimation approach from [85], I obtain a filtered, averaged inertial measurement signal from the gyroscopes, accelerometers, and joint angle encoders throughout the robot. This produces a robust orientation estimate for the virtual chassis,

$$R = \left[\vec{r_x}, \vec{r_y}, \vec{r_z}\right] = \begin{bmatrix} r_{x1} & r_{y1} & r_{z1} \\ r_{x2} & r_{y2} & r_{z2} \\ r_{x3} & r_{y3} & r_{z3} \end{bmatrix}, \tag{6.7}$$

where $\vec{r_x}$, $\vec{r_y}$ and $\vec{r_z}$ denote the $\vec{x}$, $\vec{y}$ and $\vec{z}$ axes of the virtual chassis relative to a world frame.

To retrieve the pitch of the snake robot from the estimated orientation of the virtual chassis R, we find the rotation about the $\vec{z}$ axis (longitudinal) that would result in a coordinate frame where the $\vec{x}$ axis (forward direction of sidewinding motion) is in the horizontal plane. This resulting coordinate frame can be given as

$$RR_z(-\theta_{\mathrm{VC}}) = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = \begin{bmatrix} r_{x1} & r_{y1} & r_{z1} \\ r_{x2} & r_{y2} & r_{z2} \\ r_{x3} & r_{y3} & r_{z3} \end{bmatrix} \begin{bmatrix} \cos(\theta_{\mathrm{VC}}) & \sin(-\theta_{\mathrm{VC}}) & 0 \\ \sin(\theta_{\mathrm{VC}}) & \cos(\theta_{\mathrm{VC}}) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.8}$$

where θ_{VC} is the desired pitch angle of the virtual chassis.

The condition we are interested in is finding the θ_{VC} which results in the x axis placed in the horizontal $x - y$ plane; in this case, the z component of the resulting x-axis must be 0, or $c_{31} = 0$. Therefore, we can write

$$r_{x3}\cos(\theta_{\mathrm{VC}}) - r_{y3}\sin(\theta_{\mathrm{VC}}) = 0, \tag{6.9}$$

which gives the pitch angle of the virtual chassis as $\theta_{\mathrm{VC}} = \mathrm{atan2}(|r_{x3}|, r_{y3})$

As the snake robot is moving, the $x - z$ plane of the virtual chassis is not guaranteed to be parallel to the slope. To correct for this effect, a relation between the estimated virtual chassis pitch θ_{VC} and the angle of the slope θ_{SL} is empirically determined:

$$\theta_{\mathrm{SL}} = \frac{1}{1.20}(\theta_{\mathrm{VC}} - 7.85 \times 10^{-2}). \tag{6.10}$$

This empirical relation is based on the measurements from experiments in which a snake robot executes sidewinding with different gait parameters on different slopes. Fig. 6.16 shows a summary of the experimental data along with the best fit relationship between θ_{VC} and θ_{SL}.

To further improve real-time estimation, the estimated slope angle is then fed into a simple low-pass filter to smooth out noisy measurements,

$$\tilde{\theta}_{\mathrm{SL}}[k] = \tilde{\theta}_{\mathrm{SL}}[k-1] + \frac{\Delta t}{T}(\theta_{\mathrm{SL}}[k] - \tilde{\theta}_{\mathrm{SL}}[k-1]), \tag{6.11}$$

where Δt, T , $\theta_{\mathrm{SL}}[k]$, $\tilde{\theta}_{\mathrm{SL}}[k]$ are respectively the time interval, gait period, estimated slope angle, and smoothed slope angle at time step k.

Experimental Results

To test the performance of the adaptive policy, I constructed two test environments. The first test environment serves as a simple demonstration of the adaptive controller. The slopes of the four boards relative to the ground are 12.0°, 24.6°, −18.8°, and −17.6°. As the snake robot climbs up, a large amplitude is selected according to the predicted optimal policy; when the robot is on a negative slope, the policy chooses a zero amplitude wave which results in the snake robot quickly rolling down the hill (as shown in Fig. 6.17). The time consumed for each section of the course is given in Fig. 6.18a.

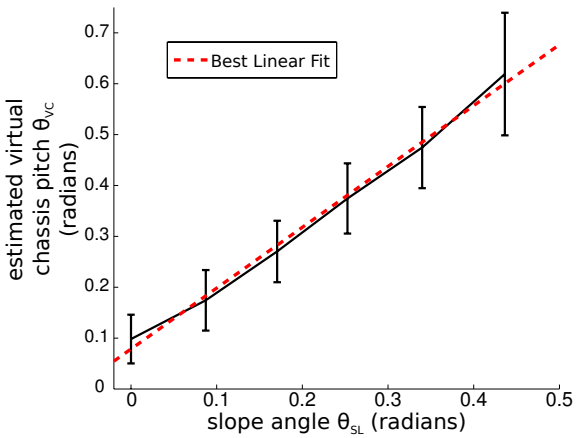
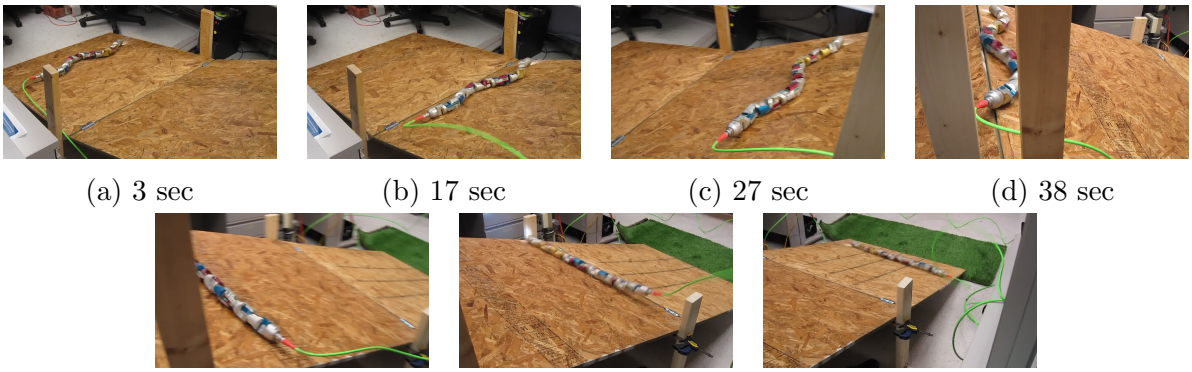


Figure 6.16: The bars in the figure denote the variance of estimated virtual chassis pitch θ_{VC} with different gait parameters and on different slopes. A linear relation between the slope angle θ_{SL} and virtual chassis pitch θ_{VC} is revealed.



(a) 3 sec (b) 17 sec (c) 27 sec (d) 38 sec

(e) 40 sec (f) 42 sec (g) 43 sec

Figure 6.17: For this setup of our test course, the optimal control policy chose a sidewinding parameterization with a wide base for the uphill segments, and quickly transitioned to a zero-amplitude roll for the downhill segments. Note the time each frame was taken, and the difference in speed when moving uphill versus downhill.

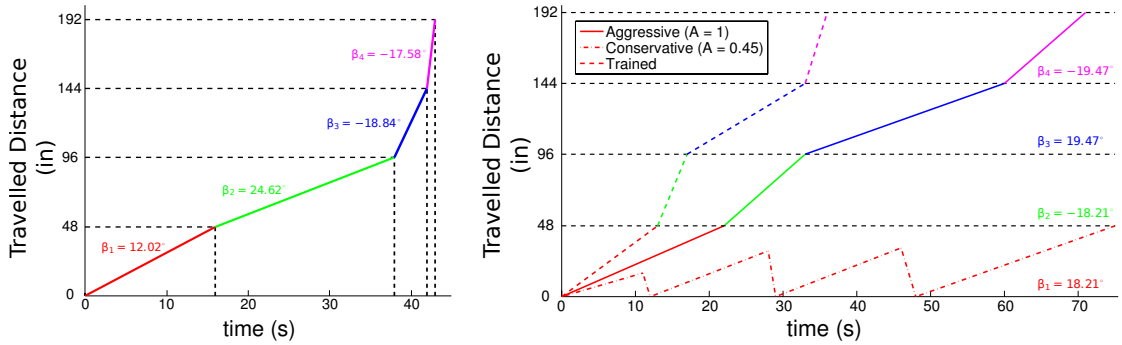


Figure 6.18: **(a)** Performance of adaptive sidewinding on a simple test course. The slope of each line denotes the travelling speed, and different colors represent panels at different slopes. **(b)** A comparison among three strategies (shown in different line types). Sidewinding with a small amplitude failed to finish the task, and the conservative approach of sidewinding with a large amplitude leads to slow completion. The adaptive strategy outperforms both static policies.

The second test environment was defined by the slope angles 18.2°, −18.2°, 19.5°, and −19.5°, and formed an M-shaped terrain. In this test environment, I compared the results of using the adaptive policy to two fixed-parameter strategies: (a) a conservative choice of $A = 1$ as an amplitude which provides good performance on ascending slopes, and (b) a small amplitude $A = .45$ which is most effective on descending slopes. The comparison of these strategies is shown in Fig. 6.18b. During execution, sidewinding with a low amplitude caused the robot to tumble down 3 times before climbing up the first ascending slope, and the robot was unable to complete the course. Sidewinding with a large amplitude succeeded in traversing the entire test course, but the time to completion (71 seconds) was longer than that of the adaptive policy (36 seconds).

6.4 Conclusion

This chapter has formulated an optimization problem that applies to expensive systems operating in varying environmental conditions. Through careful selection of training experiments, I have demonstrated the learning of effective control policies with only a small

number of objective evaluations.

This framework is applicable to a rich set of problems. Locomoting systems, industrial processes, and prescription drugs all operate in changing environmental conditions, are expensive to test, and could benefit from optimal adaptive control policies. I have described two potential approaches for this experiment selection, both inspired by the statistical notion of EI. One approach provides a fast, efficient computation that performs reasonably well, while the other is more complex and computationally intensive, but produces better results overall. I have also proposed a simple method for policy generation, given the experiments chosen by such an algorithm. I have demonstrated the efficacy of these algorithms, and presented a summary of results on analytic test functions as well as on a physical snake robot system.

In addition to producing superior control policies as compared to naïve approaches, these methods have the advantage of requiring no algorithm parameters to tune for particular applications. However, there are still several implementation details that must be considered, including reasonable covariance functions for fitting GPs to the objective and numeric integral resolution.

Improving the algorithms so that these implementation details are less important choices is one direction for future work; by improving the efficiency of the computation in the numeric integral, the limiting factor for high-resolution integrals is removed. This could be approached by applying methods to quickly update a GP distribution given one additional sample, or deriving analytic approximations that could be used in place of numeric expressions.

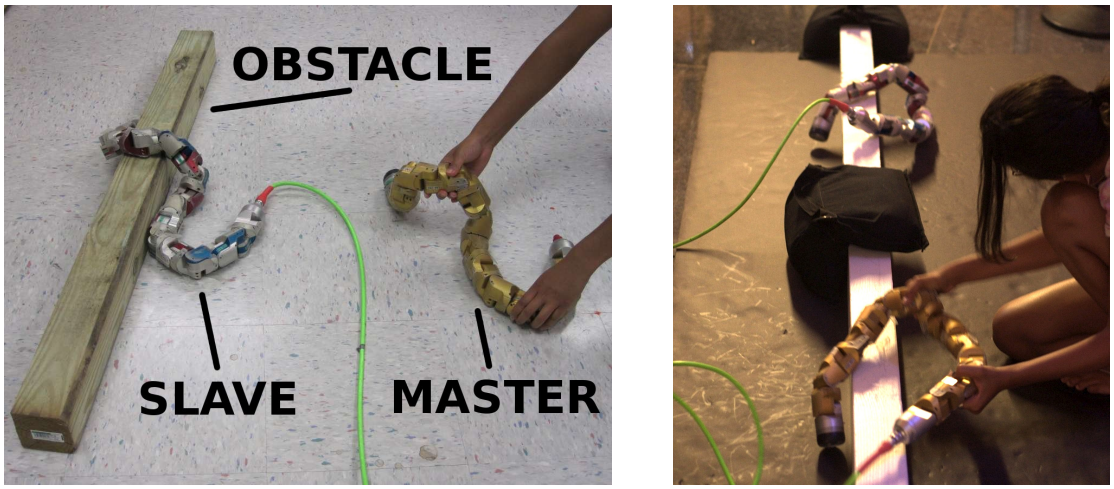
Other plans for extending this work include demonstrating the efficacy of the algorithms on higher-dimensional problems, more extensive use for improving adaptability of real world systems, and comparing the proposed methods against a wider range of possible approaches to this problem.

Chapter 7

Generating Parameterized Non-gait Motion from Demonstrated Input

The previous chapters have explored efficient optimization techniques for expensive parameterized functions in a number of settings. The snake robot demonstrations in these chapters have been limited to optimization of existing snake robot gaits (c.f. §2.3). Although these cyclic motions are useful for steady-state terrain such as flat ground, static slopes, or uniformly distributed small obstacles, they are not as appropriate for overcoming unstructured terrain on a scale comparable with the robot. Moving over large rocks, onto a ledge, over beams, or past a junction in a pipe all require special purpose scripted, non-cyclic motions. This chapter focuses on the discovery and encoding of these special purpose motions.

For complex non-planar tasks, generating such motions by prescribing individual joint trajectories can be unintuitive. Although the robot is controlled at the joint level, it is more natural to consider motions at a higher level. Inspired by the idea of master-slave, or *kinesthetic*, control for a number of complex systems (e.g., large, treaded articulated robots [5] and remote welding equipment [100]), a similar control method was implemented on the snake robots (Fig. 7.1). This approach to overcoming obstacles with a snake robot



(a) (b)

Figure 7.1: **(a)** In the kinesthetic control approach, a user moves a compliant input snake (often termed the "master" or a "dolly"), and the controlled "slave" robot mirrors the commanded shape. This can be an effective way to locomote the snake in challenging situations. **(b)** This demonstration process was used for large scale data collection at Chicago's Museum of Science and Industry; over the course of three days at the museum, nearly 10% of the 276 attempts that were made by visitors succeeded.

allows the easy, straightforward recording of open loop, acyclic commanded angle trajectories (non-gait motions) which move the robot over these obstacles and can later be replayed when these obstacles are encountered in the field. Trajectory discovery through demonstration is discussed in §7.1.

The representation of the recorded trajectory is important. Simply recording the joint angles at the full feedback rate may allow for easy playback, but does not allow for meaningful modification or improvement. As the optimization techniques I develop in this thesis are designed for parameterized functions, I seek a method to encode recorded trajectories in a relatively low-dimensional parameterization, where the parameters encode the useful variations in these trajectories. This parameterization also enables easier storage and implementation and more intuitive understanding (and therefore tuning) of the trajectories. The encoding process is discussed in §7.2, while the optimization of these trajectories to more robustly overcome obstacles (or complete other tasks) is described in Chapter 8.

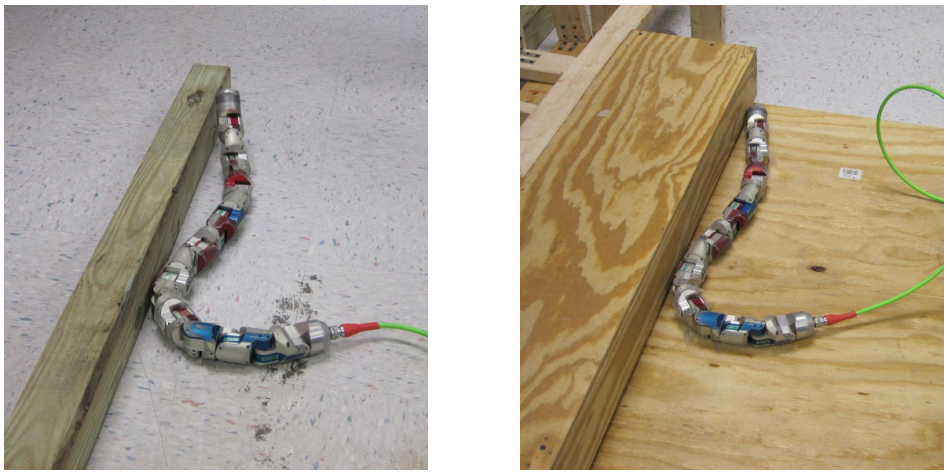
7.1 Kinesthetic Input for Recording Demonstrated Input

The kinesthetic input approach to data collection allows for dynamic collection of data through direct input and playback. This is enabled by the controllable motor current limits on our snake robots. When these limits are turned to zero, the snake is easily backdriveable. Because the modules are still powered on, the magnetic encoder continues to provide feedback.

This mode of input can be simultaneous - using one snake as a master, sending commanded angles to a slave snake - or pre-recorded - using one snake as both master and slave by recording a motion, resetting the snake to the original position of input, and playing back the most recent recording. An example of the simultaneous approach is shown in Figure 7.1a, where an operator is steering the robot over an obstacle by moving a different robot through the desired shapes.

Although in many examples of kinesthetic "teach-in" (such as motions of a robot arm) the same robot is used to first record and then replay the data, locomotion poses particular challenges due to the presence of position and orientation variables that are not directly actuated. When moving a robot arm, the complete state is recorded; when moving a locomoting system, the internal shape changes are recorded but the position of the system in the world is not. When two robots are used in a simultaneous playback mode, the operator is better able to intuitively understand the effect of his or her actions, and easily correct minor errors. This results in higher quality solutions than a two-step record and replay process.

In the following sections, I describe the process of collecting data from experts (§7.1.1) and novices (§7.1.2), summarize the collected data, and discuss the benefits and drawbacks of each approach.



(a) 4×4 Beam (3.5×3.5 inches) (b) Ledge (15 cm tall)

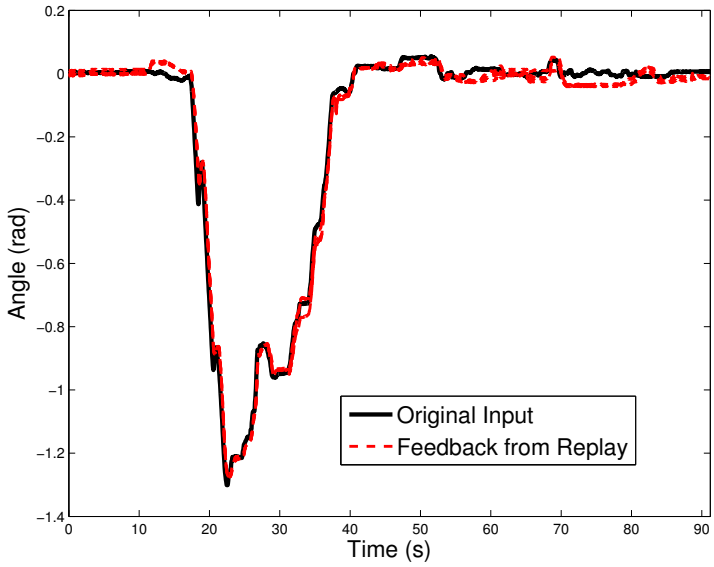
Figure 7.2: The two challenges used for collecting expert input. Both represent obstacles that were encountered during tests at TEEX Disaster City®, and which we were unable to overcome with existing gait-based control methods. Also shown in these images are the starting positions used for the trials. The goals were to **(a)** move the robot completely over the 4×4 and **(b)** to move the robot so it is supported only by the ledge, without touching the ground.

7.1.1 Expert Input

Two obstacles were chosen based on ones encountered in previous field experiences that we were unable to overcome: a 4×4 solid wood beam (Fig. 7.2a) and a 6 inch tall ledge (Fig. 7.2b). To generate a corpus of data, five different approaches for climbing each of these two obstacles were created[1]. Using the kinesthetic input approach, we saved a reference trajectory of the joint angles for each approach on each obstacle. The robot then attempted 25 trials on each obstacle, replaying each recorded reference trajectory five times to generate unique outputs. A typical example of the consistency of replaying the reference trajectory is shown in Figure 7.3.

At each obstacle the robot started in a common position, illustrated in Figure 7.2. A trial was considered a success if the robot moved completely over the beam or into a position where it was only supported by the ledge. Although these replays were run at real time for

[1]The generation, recording, and testing of these reference trajectories was done by Alex O'Neill.



(a) Module 7, "Head to Tail" approach to a 4×4

Figure 7.3: The reference trajectory for a single module in one trial is shown as the solid black line; the dotted red lines show the joint angle feedback captured when playing back this reference trajectory on a robot attempting to move over an obstacle. This illustrates typical deviation from commanded angle trajectories.

the purpose of this data collection, they can be replayed faster; a speedup factor of 10x was attempted a number of times and presented no issues.

Data from these trials were recorded, including of the type of obstacle, the method (as described below) used to overcome the obstacle, the trial number, the success of the trial, and any notes concerning the trial. A summary of this data is shown in Table 7.1.

The basic approaches taken were the following:

- **Flop:** Lift the snake's head up high, and then "flop" it down onto the obstacle.
- **Roll:** Roll the snake onto the obstacle.
- **Head to Tail:** Progress a kink through the snake from the head to the tail to get over the obstacle.
- **Tail to Head:** Progress a kink through the snake from the tail to the head to get over the obstacle.

Table 7.1: Results of Replaying Recorded Trajectories

Obstacle	Approach	Successes/Trials
4×4	Flop	4/5
	Roll	4/4
	Head to Tail Progression	5/5
	Tail to Head Progression	5/5
	Pinch	5/5
15 cm Ledge	Flop	5/5
	Roll	4/5
	Head to Tail Progression	4/5
	Tail to Head Progression	5/5
	Pinch	5/5

- **Pinch:** Lift up the middle first then the sides.

This repeatability of these experiments suggests that recording expert trajectories as input leads to useful recorded data. However, there is still room for improvement, as the success rate was not 100%, and small variations still led to failures. In §7.2 I discuss the parameterization and optimization of these trajectories to further improve robustness.

7.1.2 Novice Input

The demonstrated expert trajectories were effective, but I hypothesize that spending enough time to generate a diverse set of solutions for the vast set of starting configurations, obstacles, and other parameters is infeasible for a small group of researchers. I believe that data from a few selected, experienced individuals is unlikely to cover the landscape of potential strategies for overcoming such obstacles; to ensure that the motions can be effectively generalized across varied tasks, a broader source of ideas should be surveyed.

To obtain data from a much larger set of users, our lab began a collaboration with the Museum of Science and Industry in Chicago, Illinois; the museum generously allowed us to

set up a temporary station in its main rotunda for three days (August 24th-26th, 2012)[2]. This provided a constant line of volunteers with fresh ideas and approaches, from children with few preconceived notions of how the system should work to adults eager to try and solve the puzzle we presented.

Task Setup

The goal of the exhibit was to collect data that could be used to manipulate a snake robot over an obstacle. To simplify the task, we selected a single obstacle, the 4×4 dimensional lumber that was used as one of the challenges for the expert in §7.1.1. For the exhibit (see Figure 7.1b), we placed two of these beams end-to-end, with sandbags to hold them down and visually separate them. At the start of each trial we placed a snake robot in front of each of these obstacles, ensuring a consistent default starting configuration.

Each visitor was given the goal of moving the robot on the right completely over the obstacle, with the instruction only to touch or move the robot on the left. They were also told that the robot on the right would try to duplicate the motions of the robot that they were moving, although there were physical limits – the robot would not be able to levitate, for example, and the snake's modules had limited torque and speed. For some visitors (especially younger children), we gave brief demonstrations to illustrate these concepts. Each visitor was then given two minutes to attempt this challenge.

Interestingly, not even experienced snake robot researchers could necessarily complete this task easily. Some members of our lab had consistent success, whereas others could not complete this challenge within the allotted time.

[2]Dave Rollinson and Florian Enner assisted with the exhibit and data collection process.

Data Collection

Sensor data was recorded from each module in both snakes at approximately 20 Hz that included the joint angle, 3-axis accelerometers, 3-axis gyros[3], motor current, and internal temperature (at 1 Hz). In addition to the robots' internal sensors, we also recorded other data about each two-minute trial (i.e., a single visitor's attempt at the challenge), including

- Whether the attempt resulting in a success (the robot entirely moving over the obstacle, so no part remained in the space above it) or a failure
- The total time spent (two full minutes (usually) for failures, and the time of completion for successes)
- Video footage from two HD cameras
- Additional information about the user that enabled us to more easily match photographic or videographic data with sensor logs
- Any other notable observations

This information was included as annotations to the log files, allowing it to be easily parsed and displayed with the sensor data. The video files were logged separately, and used during post processing for qualitative assessments, debugging, and clarifying issues with the logged data.

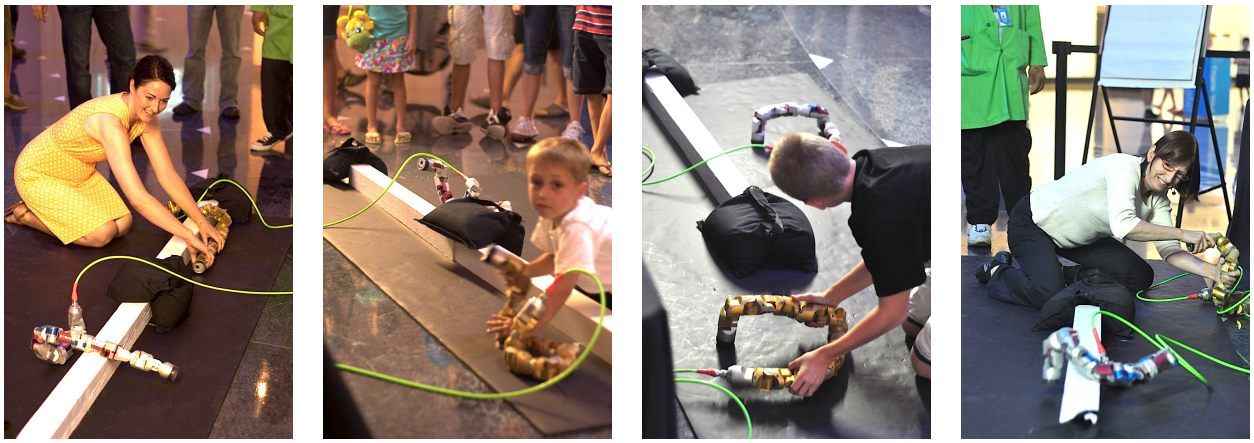
Over three days, 276 attempts at the obstacle were recorded, from users spanning various demographic categories (Fig. 7.4). Overall, the aggregate success rate was 9.4%, or 26/276. In Table 7.2, we provide success rate broken down by rough age group and gender.

[3]Some older modules only have 2 axis-gyros.

Table 7.2: Aggregate Success Percentage by Gender and Approximate Age

	Friday	Saturday	Sunday	Total
Male	9.5% (7/74)	10% (6/59.5)†	6.5% (3/46)	9.4%
Female	8.7% (4/46)	10% (4/39.5)†	9.5% (2/21)	9.4%
3-15 years	8.6% (6/70)	8.0% (4/50)	5.3% (2.5/47.5)†	7.4%
16-25 years	15% (5/34)	16% (4/25)	17% (2/12)	15%
26+ years	0.0% (0/16)	14% (2/14)	6.7% (0.5/7.5)†	6.7%
Overall	9.2% (11/120)	11% (10/89)	7.4% (5/67)	9.4%

† Some trials involved a team spanning gender or age groups, and were thus counted proportionally for each group.



(a) (b) (c) (d)

Figure 7.4: A sampling of the people who interacted with our exhibit. The participants ranged from young children to teenagers to adults.

Observed Challenges and Proposed Improvements

As expected, the collected data from the novice museum visitors was much broader than the demonstrated expert input, even within the approaches taken in the 26 recorded successes. When attempting to draw conclusions from the data, however, several challenges arose.

1. Although trials were scored with only success or failure, this does not accurately represent the quality of many of the attempts. For example, the first 30 seconds of a trial may have made significant progress over the beam, and then an accidental move reverses that progress within a few seconds. If the collected corpus of data is to be later used as the basis for learning a robust motion, it is important to indicate the value of segments representing partial trials; however, the method to segment and rate this data is not clear.

2. Many attempts effectively resulted in multiple shorter trials from differing starting configurations. If after 30 seconds of moderate progress, a mistake returns the robot to the starting side of the beam in perhaps a slightly changed orientation and shape, this amounts to the beginning of a new trial. Similar to the idea of ranking partial segments above, each of these segments should be attributed a success or failure in addition to recording the starting configuration.

3. The ground truth was limited. Although sensor data was recorded from the robots, no motion capture data was collected, and therefore the location of the robot was unknown (except to the extent that 'success' was marked on a trial the instant it fully passed over an obstacle).

4. The signal to noise ratio was high. Learning from novices, this is to be expected; however, even within most of the successful trials, there were significant exploratory motions that did not contribute to the progress of the robot.

5. One particular cause of poor quality trials is hypothesized to be "de-registration," or misalignment in relative orientation between the master and slave systems or position relative to the obstacle (c.f. the photos in Fig. 7.4). As the master is moved, the slave repeats the actions in the controlled degrees of freedom (internal shape), but the unactuated portions of the state (position and orientation in the world) cannot be replicated. Many operators appeared confused by this, the resulting de-registration decreasing the quality of input they gave, while other users immediately attempted to re-register before continuing.

In response to these data quality issues, I make several suggestions for collecting more useful data. First, motion capture data should be recorded for each trial, resulting in world position and orientation (relative to the obstacle) for both the master and slave system. This would address items 1, 2, and 3 above, providing information that could be used to segment trials, determine new initial conditions, and allow interesting conclusions to be drawn from an interaction perspective.

Second, the addition of a test station at the exhibit where visitors could use the robot for a short period to build intuition (perhaps requiring the user to pass a simple test) before attempting the timed challenge. This could be limited to control of a computer simulation to reduce system cost and complexity, but would address item 4 above by turning the novice users into trained novices.

Finally, to address item 5, avenues for active reorientation could be explored. If the master snake was aware of the slave snake's orientation, major problems (such as inversions) could be eliminated by reversing the commanded angles. Notably, the data from Sunday was taken when no copy of the obstacle was given on the master snake side; this allowed us to test whether removing this source of potential deregistration confusion would increase performance. Although there were too few trials too support any definite conclusions, the removal of the obstacle actually appeared to decrease average performance.

Overall, this massive collection of data highlighted many challenges while providing a corpus of data that could be used for preliminary analysis. The initial hypothesis that novices would generate a broader array of recorded trajectories was validated, but the usefulness of this breadth has not yet been proven. In §7.3 and §7.4, I will discuss attempts and ideas for obtaining useful low-dimensional, generalizable, parameterized trajectories from this data.

7.2 Identifying Keyframes For Single Recorded Expert Trajectory

The data collection methods in §7.1.1 enable repeatable locomotion of the snake over selected obstacles. However, in real applications such as search and rescue scenarios, variations exist in the terrain and obstacles, leading to the need to extend and improve the recorded trajectories. I also seek to increase their robustness (percentage of successful trials) in the presence of noise and small variations, and to generalize these joint angle trajectories so that they can adapt to a wider range of obstacles – ledges at different heights, for example.

This thesis focuses on optimization of parameterized functions of robot performance, but one difficulty when improving these demonstrated trajectories is the large number of variables required to represent the controller. The recorded input from the slave snake in these trials is simply a list of joint angles, received at around 20Hz over a period of 60 to 180 seconds. This leads to a matrix of size $k \times n$, where k is the number of joints (16 for the tests conducted here), and n is the number of timesteps (usually 2000 to 3000).

The goal in this section is to find a low-dimensional representation of the data. This process need not perfectly reconstruct the input, but should capture the intent of the original so that the simplified trajectory can still be used, without loss of effectiveness, to move the robot over an obstacle. Ideally, this processing would also remove unnecessary noise and jitter introduced while recording the input, potentially even improving the effectiveness of

the paths. Below, I formulate the process of finding a low-dimensional representation as a optimization problem, specifically obtaining a series of keyframes.

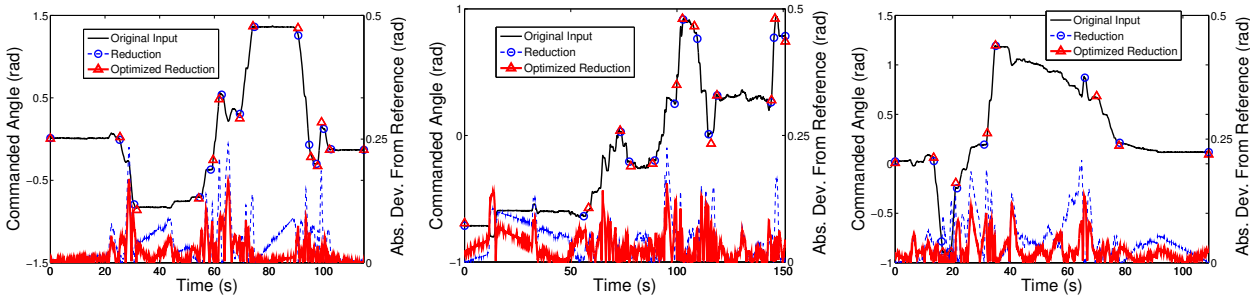
The choice of this keyframe-based approach was motivated by a number of factors. First, keyframes are easy and intuitive to vary. Second, they can often be further simplified from a vector of joint angles to the linear combination of a small number of shape modes. Finally, identified useful keyframes are easy to later sequence together when attempting to build more complex motions. Combining stored trajectories instead would require matching initial and final conditions, resulting in limited sequencing options.

Although dynamic motion primitives [90] have been used for recording and generalizing demonstration trajectories, they usually require more parameters than the keyframe approach does. Furthermore, the stated advantages for generalization are most appropriate when the robot-to-workspace transform is known (such as a fixed base manipulator); a challenge with snake robots is that they have no such known transform.

7.2.1 Method

I develop the following method to produce such a parameterized reduction of the joint angle trajectories generated in the previous section. First, I define the form of the reduction as ordered endpoints for a piecewise linear function, one set for each joint on the robot. When obtaining this reduction, the number of endpoints needed should be minimized, while maintaining some notion of representational error below a given threshold (ensuring a reasonable level of representational fidelity to the original data).

To find the parameters of the reduction for a given joint, I initially sample points that densely interpolate that joint's trajectory and use these as the endpoints for the piecewise linear function. Points are removed incrementally, selecting at each step to remove the point resulting in the smallest increase in error. This is repeated until a specified error threshold has been reached.



(a) Module 5, "Pinch" approach to a 4×4

(b) Module 12, "Roll" approach to a 15 cm ledge

(c) Module 3, "Flop" approach to a 4×4

Figure 7.5: These plots show the fidelity of a piecewise linear approximation to the original kinesthetic input for three randomly selected joint angle trajectories. The original trajectory is shown by the solid black line. Simply reducing from a dense interpolation of the input (here, the mean-squared error threshold was $0.004\,\text{rad}^2$) gives a fair approximation to the input; the circular markers indicate the linear segment endpoints and the dotted blue line indicates the deviation of this approximation from the original (right axis scale). The secondary gradient descent optimization over the linear segment endpoints provides a noticeable improvement (triangular segment endpoints and red solid line for deviation). Note that by moving to a piecewise linear parameterization, the amount of information that needs to be stored for these trajectories reduce from 2774, 3652, and 2636 points in $\mathbb{R}^2$ to 14, 13, and 10, respectively.

The fidelity of the resulting piecewise linear function is then improved by a secondary optimization step. Each remaining point is allowed to vary in time and angle, relaxing the initial assumption that these points exactly interpolate the data. This can be done through gradient descent on the representational error, cycling through all points one at a time until convergence is reached. Figure 7.5 shows the result of both the reduction and the optional optimization step for three different joint trajectories, using a threshold of $.004\,\text{rad}^2$ for each.

More formally, let the original input trajectory be defined by a vector of timestamps $T = [t_1, t_2, ...t_n]^T$, and the joint angles corresponding to these timestamps as

$$\Theta = \begin{bmatrix} \theta_1^1 & \dots & \theta_1^k \\ \vdots & \ddots & \vdots \\ \theta_n^1 & \dots & \theta_n^k \end{bmatrix}$$

We use the convention of subscripts for timestep index, and superscripts for joint number index.

Our lower-dimensional reduced trajectory can be defined as $\hat{\Theta} = \{(\hat{t}^i, \hat{\theta}^i) \,|\, 1 \leq i \leq k\}$, where $\hat{t}^i$ is a column vector of the times for the endpoints of the piecewise segments for joint i and $\hat{\theta}^i$ is a column vector with the corresponding joint angle at each of these times.

A slight overloading of notation allows us to use $\hat{\Theta}$ as a function as well; in this case, $\hat{\Theta}(t, i)$ returns the angle produced by this reduced trajectory for time t and joint i. This can be computed via linear interpolation between the surrounding points.

Given this notation, we can define the mean-squared error of our the reduced representation as the normalized sum of squared distances from the original input, or

$$\text{error} = \frac{1}{kn} \sum_{i=1}^{k} \sum_{t=1}^{n} \left(\hat{\Theta}(T(t), i) - \Theta(t, i) \right)^2, \tag{7.1}$$

where parenthesis are overloaded to index into the matrices T and Θ in row-major order. Given the more rigorous definition of the error function, the reduction and tuning/optimization step can now be completed as described above.

7.2.2 Results

Overall, this reduction was run on each of the 10 recorded trajectories (five approaches each to overcoming two obstacles), and has resulted in typical reductions from initial T of around 3000 points, and Θ that are 3000×16, to $\hat{\Theta}$ reductions that are typically defined by around $10 - 14$ time/angle pairs for each of the 16 modules. This amounts to a typical reduction factor around 120, resulting in around 360 parameters (around 24 per joint).

While typical reductions of $3000 \times 16 = 48000$ values to around 360 is significant, this obviously still results in a fairly high-dimensional space. However, in many cases the actual values that one might modify for optimization would be much lower. One might assume timestamps to be fixed, reducing to 180 parameters. If only certain modules are of interest, then this might further reduce to 45. Additionally, one can couple keyframes across multiple

Table 7.3: Results of Simple Parameterized Trajectory Controllers

Obstacle	Approach	Successes/Trials		
		Original	Reduced	Optimized
4×4	Flop	4/5	5/5	5/5
15 cm Ledge	Roll	4/5	5/5	5/5
15 cm Ledge	Head to Tail	4/5	4/5	4/5

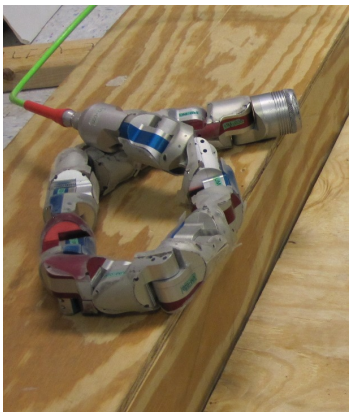
modules, tolerate increased error, use wavelet decomposition, form a grammar from common symbols, or apply other analysis with expert knowledge to decrease the number of parameters for optimization into a reasonable range ($10-30$), depending on the task. The exact processes which this further reduction would involve are likely to be highly task dependent.

Testing Reduced Trajectories

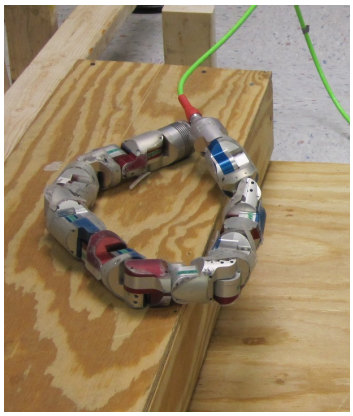
To validate the usefulness of the reduction methods described above, we must test the resulting joint trajectories to verify that they can actually accomplish the task with a similar rate of success as compared to the originals.

For a optimization threshold of $0.004\,\mathrm{rad}^2$, Table 7.3 shows the performance of the reductions as compared to the original commanded trajectories for three selected obstacle/approach combinations. In no case is there a decrease in rate of success; in fact the rate of success increases for two obstacles (although there is not enough data for any improvement to be statistically significant). I hypothesize that any increase in success rates is due to a reduction in the high frequency signal (mostly noise during the original data collection), resulting in more repeatable motions.

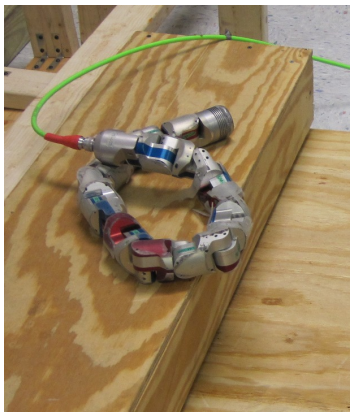
Interestingly, these results do not show a noticeable improvement between the simple reduction and the extra optimization step (potentially due to the high quality of the reduction). However, while running the tests there was indication that this optimization significantly increased the faithfulness of the representation. In Figure 7.6, the ending position for the



(a) Original Controller



(b) Initial Reduction



(c) Optimized Reduction

Figure 7.6: The ending position for the "Head to Tail" approach to climbing on top of a six inch ledge. The optimized reduction is noticeably more faithful to the original controller, as indicated by the fewer modules overhanging the ledge resulting in a more stable ending configuration and the better match between ending shapes.

"Head to Tail" approach to moving onto a ledge is compared for the original trajectory, the reduction, and the optimized reduction. In both the original and optimized reduction, there is very little of the robot hanging over the ledge, whereas the simple reduction leaves considerable more of the robot over the ledge.

Although more conclusive tests will be done in future work, these results give strong supporting evidence that the given reduction and parameterization methods result in simple, effective controllers for overcoming obstacles. In particular, these results indicate that there is no noticeable degradation in performance between replaying the original trajectories and the piecewise linear optimized reductions.

7.3 Finding Parameterized Trajectories Using Large Collections of Novice Trajectories

The hypothesis that led to collecting data from large numbers of novice users was that the resulting data would contain a breadth of different solution classes as well as variations within

similar classes of solutions. The desired end goal when parameterizing these trajectories is a set of low-dimensional parameterized trajectories where the parameters describe much of the variation among the demonstrations.

The parameterization approach described in 7.2 is not appropriate for generating such a representation, as it is designed for single trajectories. However, this approach demonstrated the success of identifying keyframe shapes, which can be similarly applied to collects of novice trajectories.

To learn a set of solutions, a clustering approach was used to segment the entire collection of recorded data into keyframes based on Euclidean distance in joint angle space. If the joint angles from demonstration D_i are given as the matrix

$$\Theta_i = \begin{bmatrix} \theta_1^1 & \dots & \theta_1^k \\ \vdots & \ddots & \vdots \\ \theta_n^1 & \dots & \theta_n^k \end{bmatrix},$$

where each column represents a timestep and each row a joint of the robot, then the matrix $\tilde{\Theta} = [\Theta_1 \, \Theta_2 \, \dots \, \Theta_m]$ represents the joint angles from all of the trials. The columns were fed into a clustering algorithm (both k-means and Gaussian Mixture Models (GMMs) were used), resulting in a cluster center $c_i = [\theta_1 \, \theta_2 \, \dots \, \theta_n]^T$ associated with each column of $\tilde{\Theta}$.

To identify classes of solutions, each demonstration D_i was described as a transition map between states corresponding to cluster centers. Self transitions were ignored; if the cluster centers for Θ_i were $c_1, c_1, c_2, c_2, c_4, c_1$, the identified transitions would be $1 \to 2$, $2 \to 4$, and $4 \to 1$. The number of each of these transitions were then used as edge weights on a graph G with vertices for each cluster center. This graph served as a visualization on which different solution classes could be easily identified.

Unfortunately, a reasonable number of cluster centers and other clustering algorithm parameters could not be found that resulted in a balance between representational fidelity

(improving with number of clusters) and clarity of solution classes (decreasing with number of clusters). Even with thousands of cluster centers, few of the original solutions could be played back just using the associated keyframes. The number of cluster centers had to be reduced to around 50 for patterns to appear in the transition graph (with more clusters, most edges corresponded to only a single transition from the data).

I believe that this approach could be used as the basis of a useful reparameterization algorithm, but the data collection issues mentioned in §7.1.2 resulted in challenges when attempting to analyze the data. More sophisticated approaches, such as those discussed below, could lead to further improvements on improved data.

7.4 Conclusions and Future Work

This chapter has demonstrated that kinesthetic input is effective to learn motions not just for manipulators (e.g., [11, 91, 92]), but also for locomoting snake robots. An approach for obtaining low-dimensional parameterized representations of demonstrated input has also been described, and shown to be effective for simplifying expert trajectories.

Attempting to generate a broad set of low-dimensional parameterized trajectories from a collection of novice input was not as successful. Although better results might have been obtained by gathering a larger quantity of data, I believe the underlying problem was the quality of the data. I have discussed several considerations for future collection of such data.

In particular, additional information such as the orientation and position of the system would likely have resulted in more intelligent cluster centers, improving the identified patterns. This information could also assist in identifying useful subtrajectories from failures which seemed to be were partially effective. Given an improved transition graph, one could use unsupervised learning techniques to subsample the data and obtain small potentially useful segments, sequencing these segments to plan trajectories over new obstacles.

An observation made during the clustering process was that Euclidean distance in the space of joint angles is not the correct distance metric. As the robot is using its whole body to locomote by pushing off of the environment, the overall shape is more important. This implies that a distance metric using module positions in $\mathbb{R}^3$ could improve clustering performance, especially when paired with additional state information from an improved data collection process (such as distance from obstacle). Such a metric would also be more adaptable to changes in the number of segments in the robot. However, consideration must be given during reconstruction of trajectories to playback on the robot, as the mapping from shapes to joint angles is nontrivial.

Finally, other methods such as using kernel principal component analysis (PCA) with a dynamic time warping (DTW) kernel seem promising, and should be considered. Overall, I believe learning from multiple noisy demonstrations holds promise for obtaining robust solutions, and contains many interesting research challenges.

Chapter 8

Stochastic Binary Optimization

In robotics, demonstrations from humans can be effective in situations where a task must be completed – some examples are suture tying [91, 92], attempting the "ball-in-cup" manipulation challenge [11], or overcoming obstacles with a snake robot. These tasks often define some natural binary notion of success – is the knot tied, is the ball in the cup, or has the robot passed over/through the obstacle. The definition of a more continuous objective, however, is not intuitive. How does one measure whether any particular failed attempt to tie a knot is better than any other?

In the previous chapter, I discussed methods for finding low-dimensional parameterized representations of demonstrated trajectories. These representations can then be optimized relative to some simple objective (Chapter 4), multiple objectives (Chapter 5), or across a range of environments (Chapter 6). The optimization approaches described in these chapters take into consideration the expensive nature of robotic systems, but all share a significant limitation: the objective is assumed to be a continuous, real-valued function.

In this chapter, I discuss the problem setting where the objective is not a deterministic continuous-valued function, but a stochastic binary valued function. The binary property of the objective is straightforward to define in terms of success. Allowing for stochasticity is

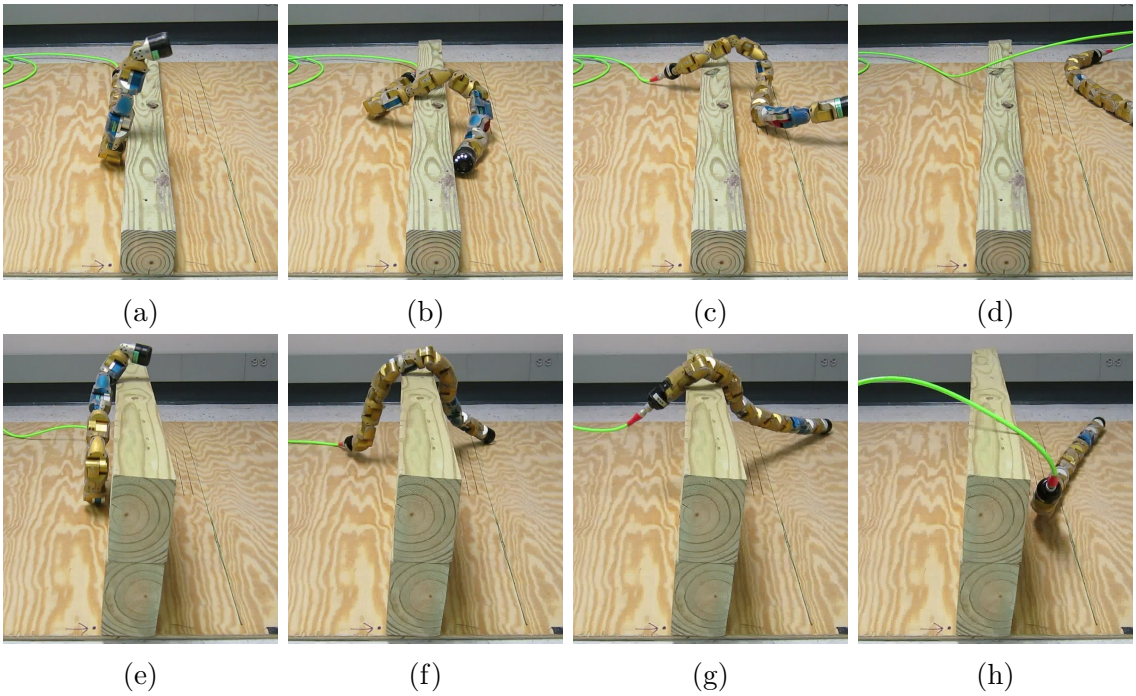


Figure 8.1: Efficient optimization is possible even with limited function evaluations which only return a noisy success or failure. **Top:** Moving over a 3.5 inch beam with the predicted best motion after 20 evaluations, using no prior information. **Bottom:** Building a strong prior by sharing results from previous optimizations on different obstacles allows the robot to move over an 11 inch beam on the first attempt.

important because the non-deterministic nature of physical systems (due to environmental factors or noise in the system) can cause subsequent attempts using identical commanded trajectories to have differing results. I describe the use of a similar framework to that described in Chapter 4 for the stochastic binary setting.

Unfortunately, attempting this optimization in parameter spaces where regions with significant probability of success are relatively sparse (e.g., a snake robot attempting to overcome a tall obstacle) will effectively amount to a blind search. Using the concept of environment (task) parameters and control parameters discussed in Chapter 6, I describe how to exploit task structure to solve simpler problems first (smaller obstacles), and then use the learned knowledge as a principled prior for the difficult task. This enables efficient optimization of tasks which would otherwise require an exhaustive search.

This chapter begins with a definition of the stochastic binary optimization problem. I describe several existing algorithms which could be applied to this problem as baselines. Next, I derive a selection metric for stochastic binary functions based on the idea of EI for binary outputs. I present a summary of results from a comparison of these methods on a set of synthetic test functions, and apply the EI-based method to learn robust motions for a snake robot to overcome obstacles (top of Fig. 8.1). Finally, I demonstrate the sharing of knowledge among tasks in order to efficiently learn to overcome more difficult obstacles (bottom of Fig. 8.1).

The primary contributions in this chapter are the definition of the stochastic binary optimization problem, the application of Gaussian processes for classification (GPC) to adapt expensive black box optimization methods to this problem setting, and the definition of the EI for stochastic binary outputs. Secondary contributions include the comparison of methods on synthetic test functions and the optimization of a new locomotive capability on a real snake robot.

8.1 Binary Stochastic Problem Definition

The problem I attempt to solve is analogous to minimizing simple regret for a continuous-armed bandit that receives a 1/0 binomial reward, with a budget of n function evaluations.

More formally, I state it as follows: given an input (parameter) space $X \subset \mathbb{R}$ and an unknown function $\pi \colon X \to [0, 1]$ which represents the underlying binomial probability of success of an experiment, the learner sequentially chooses a series of points $\tilde{\mathbf{x}} = \{x_1, x_2, \ldots x_n \,|\, x_i \in X\}$ to evaluate. After choosing each x_i, the learner receives feedback y_i where $y_i = 1$ with probability $\pi(x_i)$ and $y_i = 0$ with probability $1 - \pi(x_i)$. The choice of x_i is made with knowledge of $\{y_1, y_2, \ldots y_i - 1\}$. The goal of the learner is to recommend, after n experiments, a point x_r which minimizes the (typically unknown) error, or *simple regret*,

$\max_{x \in X} \pi(x) - \pi(x_r)$; this is equivalent to maximizing performance $\pi(x_r)$.

8.2 Approach

As mentioned above, the core method used in this chapter is an adaptation of the expensive optimization framework described in Chapter 4. A key insight is that this previous approach is applicable to the binary stochastic problem domain if the surrogate function regression method is replaced with one more appropriate.

This becomes clear in the problem definition above. In fact, although the output obtained from sampling the system in this chapter is always 0 or 1, we are still searching for the optimum of a continuous function as before; this is now π (our probability of success) rather than the objective f.

To address this, I model the surrogate function with a formulation of GPs for classification problems, as described in §2.2. This use of GPC serves as a way to better model our assumptions about the problem. Specifically, GPC assumes an underlying π and only allows conditioning on binary 0 or 1 data, matching our problem definition. For the GPC response function σ, I use the standard normal CDF.

Given the change in regression method, the expressions for the selection metric must also be modified. In the remainder of this section, I explore several baseline approaches to this problem (only some of which use GPC as the regression method), as well as a novel principled extension of EI to the binary stochastic base.

8.2.1 Baseline Algorithms

Using GPC to model this problem allows us to infer a posterior probability distribution $\hat{\pi}$ over the unknown true function π from observing several (x_i, y_i), and also to obtain a posterior over a latent function $\hat{f}$. Although this latent function could technically be used

for experiment selection, it does not have a direct probabilistic interpretation except through the response function σ.

As baselines to compare against the binary EI metric I present in §8.2.2, I use a uniform random experiment selection method along with the following approaches.

First, as UCB methods are often used in bandit and expensive optimization problems (e.g., [3]), I compare against UCB on the latent function $\hat{f}$, with β as a tuneable metric parameter:

$$\mathrm{UCB}_f^\beta(x) = \hat{f}_\mu(x) + \beta \hat{f}_\sigma(x) \tag{8.1}$$

Note that this method cannot be applied directly using $\hat{\pi}$, because the $\hat{\pi}_\mu(x)$ and $\hat{\pi}_\sigma(x)$ are not well defined. One may choose to use the variance of $\hat{\pi}(x)$ (which in general must be computed numerically) for $\hat{\pi}_\sigma(x)$, or one may interpret the original quantity statistically and choose to define

$$\mathrm{UCB}_\pi^\beta(x) = y \text{ where } \int_0^y p_\pi^x(t)dt = \beta, \tag{8.2}$$

using β directly as the confidence level rather than a multiplier on the standard deviation; again, the integral of the PDF p_π^x would in general require a numerical integral.

For comparison, I also test the standard EI metric in the latent space, EI_f, and on a standard GP directly fit to the binary data. For the former, because we are not directly observing samples from $\hat{f}$ we must redefine the $y_{\max}$ term in the improvement quantity from Equation (2.20) as

$$y_{\max} = \max_{\tilde{\mathbf{x}}} \{\sigma^{-1}(\bar{\pi}(x))\}, \tag{8.3}$$

where $\tilde{\mathbf{x}}$ is all sampled x_i. This represents the latent space projection of the maximizer of $\bar{\pi}(x)$ at the previously sampled points.

Finally, I compare against the continuous-armed bandit algorithm UCBC (Upper Confidence Bound for Continuous-armed bandits) proposed in [4]. This algorithm divides X into a set of n equal-sized intervals, and runs the multi-armed bandit UCB algorithm to select the interval from which to sample. The point to sample is then chosen uniformly at random from this interval. Recommendations for how to choose the algorithm parameter n are given in [4].

8.2.2 Expected Improvement for Binary Responses

In the case of stochastic binomial feedback, the notion of improvement that underlies the definition of EI must change. Because the only potential values for y_i are 1 and 0, after the first 1 is sampled $y_{\max}$ would be set to 1. As there is no possibility for a returned value higher than 1, improvement (and therefore EI) would be identically zero for each $x \in X$.

Instead, note that these are noisy observations of an underlying success probability and query the GP posterior[1] at each point in $\tilde{\mathbf{x}}$. Let

$$\hat{\pi}_{max} = \max_{\tilde{\mathbf{x}}} \bar{\pi}(x). \tag{8.4}$$

As the 0 and 1 responses are samples from a Bernoulli distribution with mean $\pi(x)$, I define the improvement as if one could truly sample the underlying mean. Choosing this rather than conditioning our improvement on 0/1 is consistent with the fact that our $\hat{\pi}_{max}$ represents a probability, not a single sample of 0/1. In this case,

[1] I have evidence that this is a reasonable approach, as the stochastic experiments in §4.3 did not find significantly different performance between this and more sophisticated approaches.

$$I_\pi(x) = \max(\hat{\pi}^x - \hat{\pi}_{max}, 0), \tag{8.5}$$

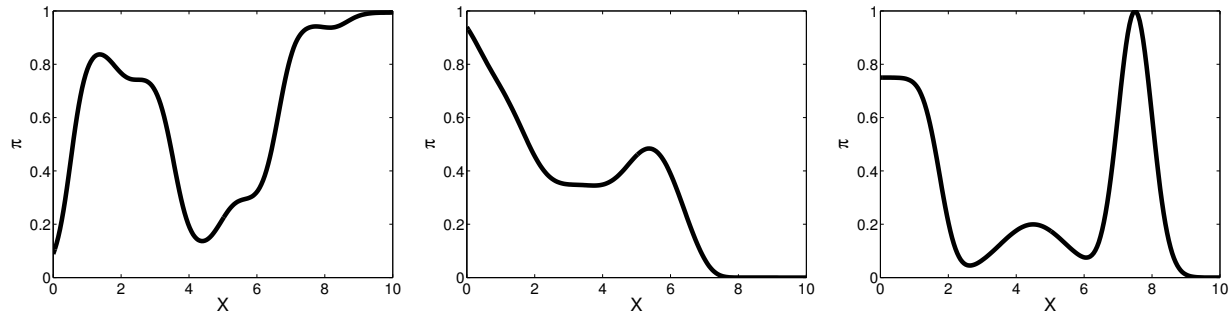
where $\hat{\pi}^x$ is the random variable associated with p_π^x, the GPC pointwise posterior distribution at x. This is analogous to $\hat{f}^x$ in Equation (2.20). Note that $I_\pi(x)$ is a random variable.

To calculate the binary EI, I follow a similar procedure to that in §2.2 (in particular, the definition of expectation along with a change-of-variables substitution for $\pi = \sigma \circ f$ and $\bar{y} = \sigma(z)$) to calculate the expectation of $I_\pi(x)$.

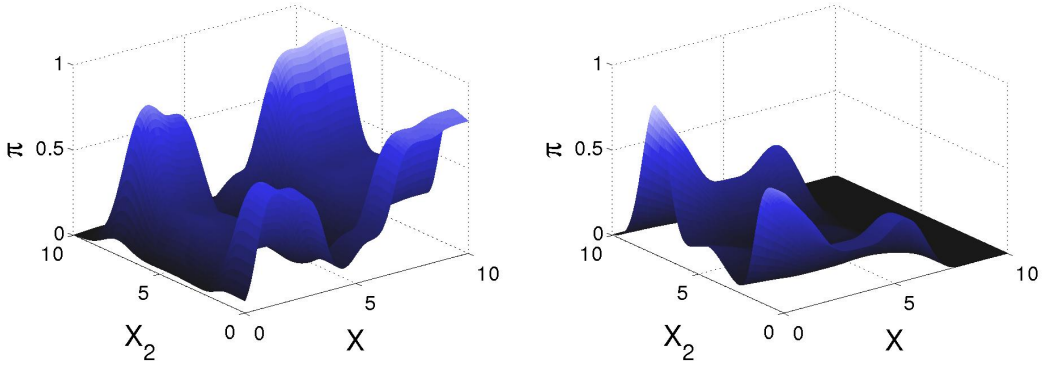
$$\begin{aligned}
\mathrm{EI}_\pi(x) &= \int_{\hat{\pi}_{max}}^{1} (\bar{y} - \hat{\pi}_{max}) p_\pi^x(\bar{y}) d\bar{y} \\
&= \int_{\hat{\pi}_{max}}^{1} (\bar{y} - \hat{\pi}_{max}) p_Y^x(\sigma^{-1}(\bar{y})) \frac{\delta \sigma^{-1}}{\delta \bar{y}}(\bar{y}) d\bar{y} \\
&= \int_{\sigma^{-1}(\hat{\pi}_{max})}^{\sigma^{-1}(1)} (\sigma(z) - \hat{\pi}_{max}) p_Y^x(z) \frac{\delta \sigma^{-1}}{\delta \bar{y}}(\sigma(z)) \frac{\delta \sigma}{\delta z}(z) dz \\
&= \int_{\sigma^{-1}(\hat{\pi}_{max})}^{\infty} (\sigma(z) - \hat{\pi}_{max}) p_Y^x(z) dz
\end{aligned} \tag{8.6}$$

The marginalization trick that allowed us to evaluate this integral and obtain a solution only requiring the Gaussian CDF in the case of $\bar{\pi}$ (Equation (2.16)) does not work because here the integral is not from $-\infty$ to ∞; fortunately it is one-dimensional regardless of the dimension of X and is easy to numerically evaluate in practice (e.g., using adaptive quadrature techniques).

After finding this expression for EI_π, it can be used as a selection metric in the modified black box optimization algorithm discussed above (using GPC as the function regression method).



(a) Test Function 1 (b) Test Function 2 (c) Test Function 3



(d) Test Function 4 (e) Test Function 5

Figure 8.2: A number of synthetic test functions were created for algorithm comparison and validation. The equations for the functions used as benchmark tests in this chapter are given in Appendix A.

8.3 Empirical Results on Synthetic Functions

To validate the performance of the adapted EI metric for stochastic binary outputs, I created several synthetic test functions for $\pi(x)$ on which a large number of optimizations could be run. Shown in Fig. 8.2 are several of these functions, which exhibit properties such as multiple local optima and a narrow global optimum to challenge optimization algorithms; moreover, they are stochastic ($\pi(x) \notin \{0, 1\}$) over much of X.

8.3.1 Experimental Setup

To compare the various algorithms, I allowed each algorithm to sequentially choose a series of 50 points, $\tilde{\mathbf{x}} = \{x_1, x_2 \ldots x_{50}\}$, with feedback of y_i generated from a Bernoulli distribution

with mean $\pi(x)$ (according to the test function) after each choice of x_i. This was completed 100 times for each test function.

For our random selection baseline, at each step i, a random point was chosen and evaluated. For the baseline EI_f and UCB_f metrics (which used the latent GP) as well as the proposed EI_π metric, the standard black box optimization framework described in Alg. 1 was used with an initial Latin hypercube sampling of 5 points. The UCB_f baseline tests were run with various values of the β parameter from 0.5 to 10; 1 was found to work as well as or better than other values and is shown in the comparison here. The maximization of the metric was done by evaluating the metric on a dense grid over the space; in practice and in higher dimensions one would typically apply another global optimization method to obtain the maximizer.

One of the contributions in Chapter 4 is the formal model selection process leading to the robust black box optimization algorithm shown in Alg. 2. Although this chapter advocates for selection of the covariance function and hyperparameters of the GP at each iteration through likelihood maximization, for the comparisons in this chapter I chose to use a simple squared exponential covariance with fixed hyperparameters (length scale of $e^{0.75}$ and signal variance of e^5) to reduce the variance in algorithm performance due to optimization of this likelihood function.

For comparison with the UCBC algorithm, the algorithm parameter of n was chosen as recommended in [4] for unknown functions, $n = (T/ln(T))^{1/4} = 2$, assuming the number of samples $T = 50$. I also ran UCBC with $n = 10$, but did not get appreciably different performance.

All of the algorithms described in this chapter are implemented in the code provided with this thesis. The example MATLAB scripts used to run these experiments are given in §B.3.

8.3.2 Measuring Performance

To obtain a measure of the algorithm's performance at step i, I use the natural Bayesian recommendation strategy of choosing the point which has the highest expected probability of success $\mathbb{E}[p_\pi^x]$ given knowledge only of the evaluated points $\{x_1, x_2 \dots x_i\}$ and values $\{y_1, y_2 \dots y_i\}$. In practice, one may wish to optimize a utility function that also considers risk (e.g., the uncertainty in that probability).

After choosing $x_r = \text{argmax}_X \, \mathbb{E}[p_\pi^x]$, this point is evaluated on the underlying true success probability function π, and the resulting value $\pi(x_r)$ is given as the performance of the algorithm at step i. For the random selection and UCBC algorithms which do not have a notion of $\hat{\pi}$, a GP was fit to the data collected by the algorithm to obtain this $\hat{\pi}$ using the same parameters as the other algorithms[2].

8.3.3 Comparison of Results

In Fig. 8.3, I plot the average performance over 100 runs of the proposed stochastic binary EI EI_π as well as the random baseline and the UCBC algorithm. As expected, the knowledge of the underlying function grew slowly but steadily as random sampling characterized the entire function. The focus of EI_π on areas of the function with the highest expectation for improvement led to a more efficient strategy which still chose to explore but focused experimental evaluations on more promising areas of the search space. Notably, EI_π matched or outperformed tuned versions of all other algorithms tested, *without* requiring a tuning parameter.

The UCBC algorithm worked well for simple cases (test function 1 had a significant region with high probability of success) but faltered as the functions became more difficult to optimize. An inefficiency of this algorithm is that there is no shared knowledge between

[2]UCBC does not define a recommendation strategy; the natural choice of a point uniformly at random from the interval with the highest mean performed very poorly and was therefore omitted from the results.

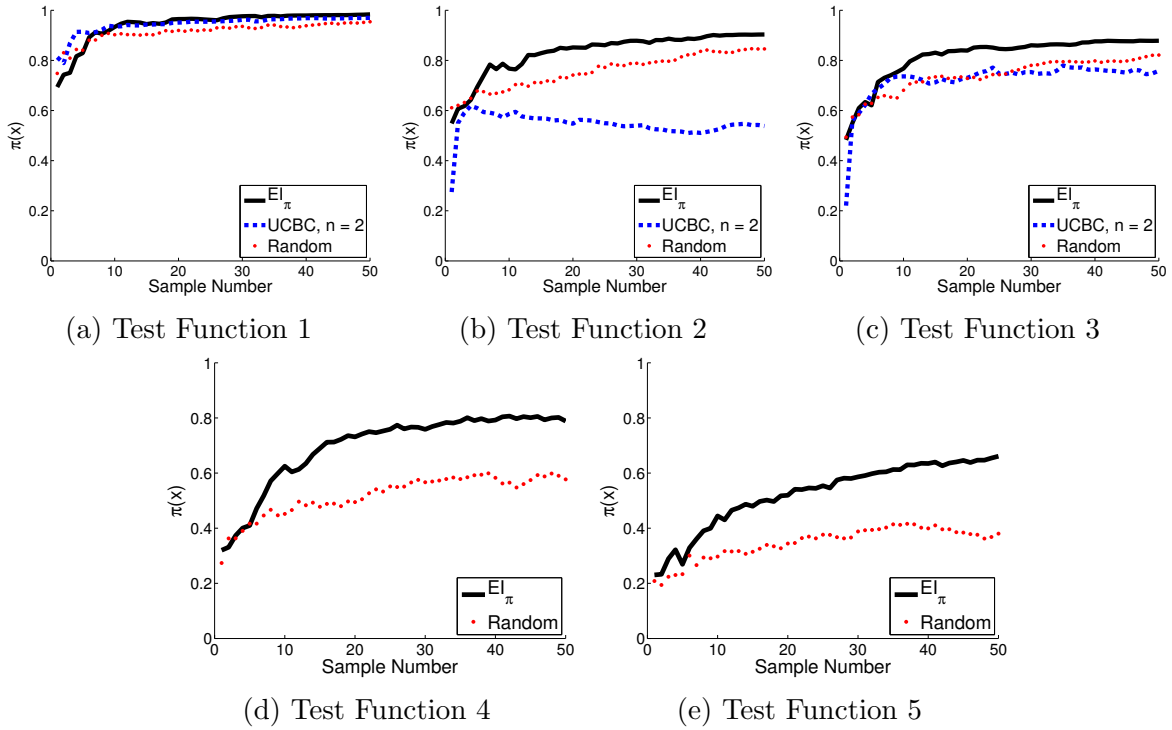
nearby intervals – one would expect that if a function is continuous, the performance at interval k is likely to be similar to that of $k-1$ for a large enough number of intervals. A potential difficulty is the dependence on a tuning parameter – the number of intervals n. It is likely that different values for this parameter would significantly affect the results; I chose the parameter recommended by [4] ($n = 2$), but also varied this parameter (to $n = 10$) and obtained comparable performance on test functions 1 and 3 and slight improvements on test function 2. This reinforces my observations that bandit algorithm parameters which produce the best theoretical bounds do not always translate to efficient algorithm performance.

Another challenge is that UCBC is not defined for higher dimensions; the natural extension would be to use a grid of area elements instead of a set of intervals, but the choice of n for each dimension isn't clear; for this reason I limit the comparison with UCBC to the one-dimensional test functions (1-3).

I also note that EI_π outperforms the naïve use of black-box optimization techniques on the latent GP $\hat{f}$, as shown in Fig. 8.4. This is largely true because the interpretation of variances on the latent function when used in the classification framework is unintuitive – the variance $\hat{f}_\sigma$ is not based solely on the sampled points as in the regression case; instead larger values of $\hat{f}_\mu$ tend to have larger variances due to the nonlinear mapping into the space of probabilities $\hat{\pi}$.

This problem is especially apparent in test function 3, where the local maxima cover a fairly wide area likely to be sampled during the initial space-filling design, whereas the global maximum is narrower; because the variance of the latent function continues to be high at high values of the mean, and drop off very slowly, both EI_f and UCB_f tend to focus remaining evaluations in this localized area.

Because EI_π instead uses the posterior in the underlying success probability space, the variance decreases near the local maxima as expected, and the algorithm explores other areas of X with potential for improvement.



(a) Test Function 1 (b) Test Function 2 (c) Test Function 3

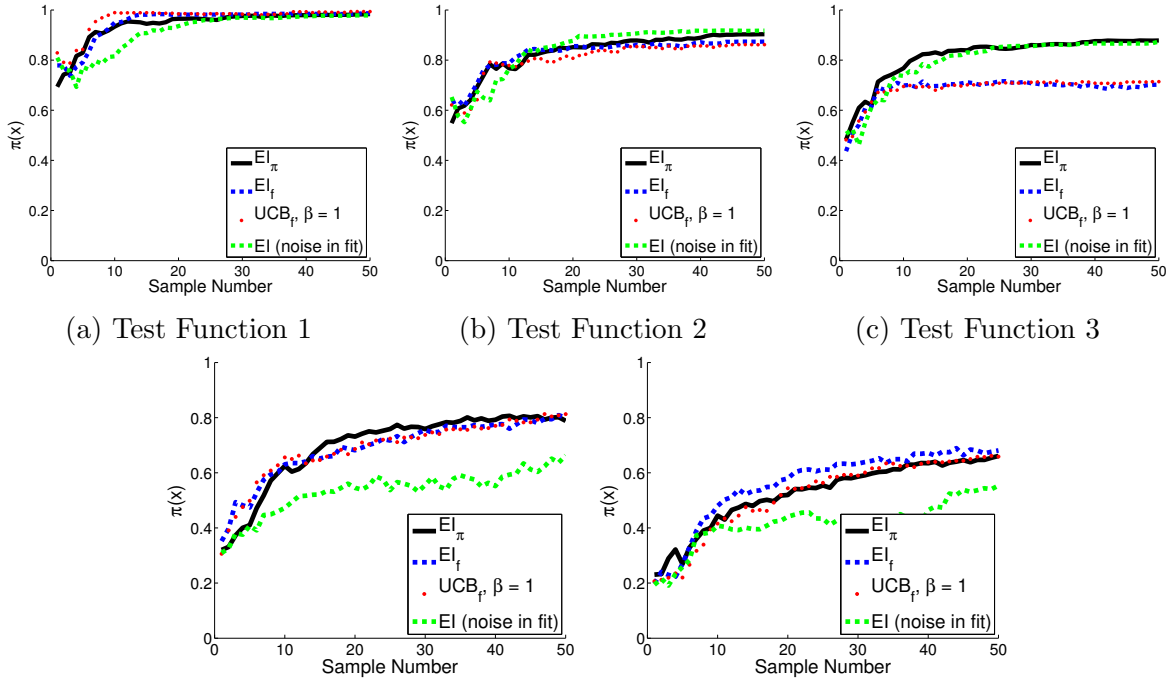
(d) Test Function 4 (e) Test Function 5

Figure 8.3: After each sample, each algorithm was queried as to its recommendation for a point x that would have a maximum expectation of success $\pi(x)$; these results show the underlying probability value of that point averaged over 100 runs of each algorithm. Here I compare the stochastic binary EI (EI_π) to the continuous-armed bandit algorithm UCBC suggested in [4] as well as uniform random selection. The cause of the unusual drop in performance of UCBC with more samples on test function 2 has not been determined.

Finally, standard EI fit directly to the binary data performs remarkably well on the 1-D functions, while not performing as well on the higher dimensional test problems. This method also required additional model selection; the results shown are the best obtained after carefully fitting a noise term in the diagonal of the covariance; poor selection or omittance of this term resulted in performance far below any baseline shown.

8.4 Physical Robot Experiments

The motivating system for this thesis, Choset's snake robot, is able to use gaits to move quickly across flat ground, forward through narrow openings, and up the inside and outside



(a) Test Function 1 (b) Test Function 2 (c) Test Function 3

(d) Test Function 4 (e) Test Function 5

Figure 8.4: As in Fig. 8.3, the average expected probability of success for each algorithm's recommendation at each step is shown above. These results, averaged over 100 runs of each algorithm, compare the proposed EI_π algorithm with use of EI and upper confidence bounds on the latent function obtained while fitting a GP to binary data, and EI on a GP fit directly to the 0/1 data with a tuned noise parameter.

of irregular horizontal and vertical pipes. However, moving over cluttered, obstacle-laden surfaces (such as a rubble pile) provides a challenge for the system. The task in this chapter, moving over a beam of dimensional lumber, was inspired by encountering this obstacle (in particular, a 4×4) in field deployments during disaster response training exercises.

As described in §7.1.1, a master-slave system was set up to record an expert's demonstration of moving the robot over an obstacle. Using a sparse function approximation of the expert's input, I created a 7-parameter model that was able to overcome obstacles of various sizes, albeit unreliably – the same parameters would only sometimes result in success. Parameters of this model (offset, widths, and amplitudes of travelling waves) were difficult to optimize by hand to produce reliable results.

Using the EI_π metric in the optimization framework described above, 2- and 3-dimensional

subspaces of the model were searched to identify regions of the parameter space that resulted in a robust motion over the obstacle that was used to record the original unreliable motion. In each of these cases, running 40 experiments at 20 points[3] resulted in the recommendation of a parameter setting which produced robust, successful motions (Fig. 8.5).

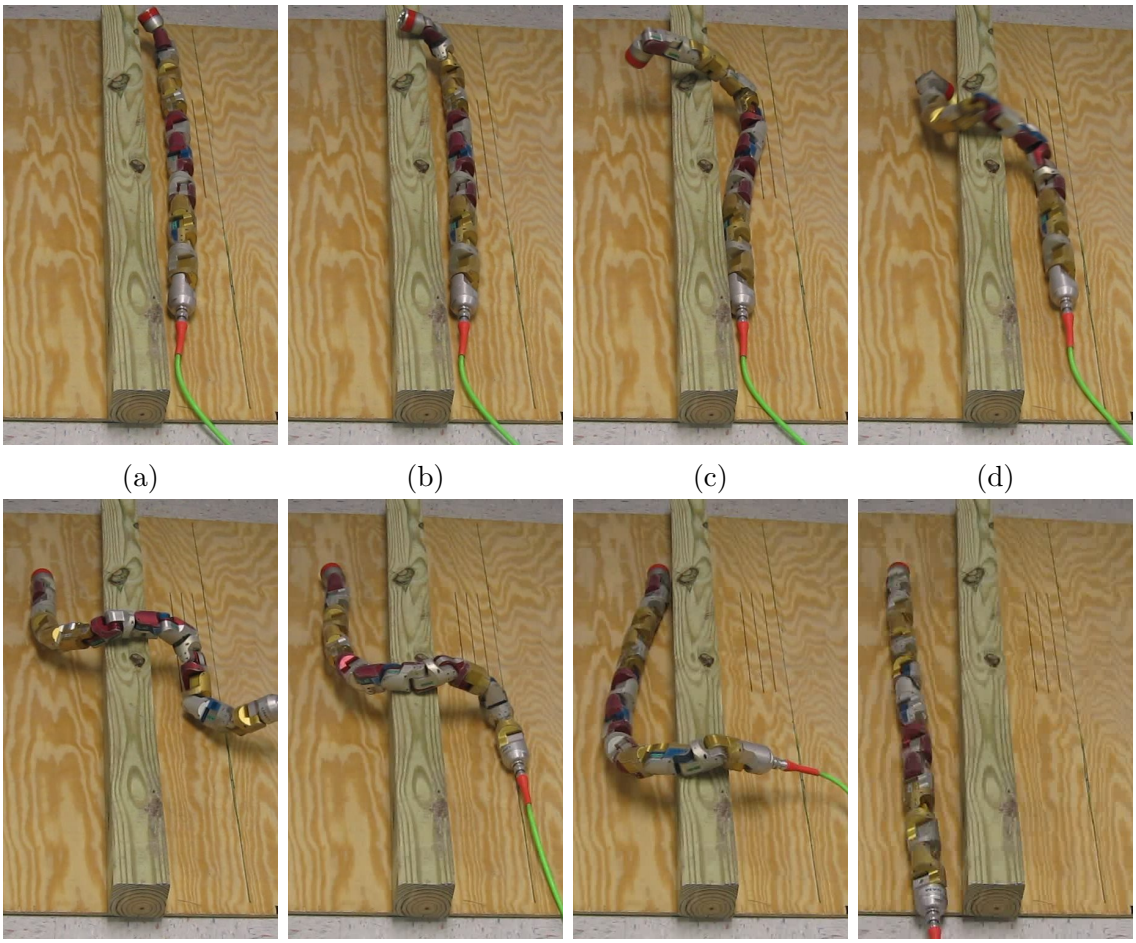


Figure 8.5: After completion of the optimization, the predicted best parameters result in a robust motion which successfully moves the snake robot over the obstacle.

After completing the first 2D optimization, it was noted that after the optimization found a successful solution it would not sample other areas of the space. This is because as $\hat{\pi}_\mu$ approaches 1, the maximum possible improvement approaches 0, as does EI_π; this

[3]The model and test setup resulted in two experiments per selected parameter.

discourages selection of points that are not near the current maximum. This effect does not occur in the non-binary setting, because improvement (and therefore EI) is unbounded no matter the value of $y_{\max}$. For a given desired EI value, a normal distribution (potentially generated from the GP) exists which can exceed this threshold.

While continuing to sample this successful solution in the binary setting technically meets the optimization goal of finding a robust solution, there is utility in the use of the remaining experiments to find other robust solutions in the parameter space. To accomplish this exploration, I modified the selection metric to not select any point which had a high confidence in its estimate of the true probability. To measure this confidence in being within ϵ of the mean at a point x, take the integral

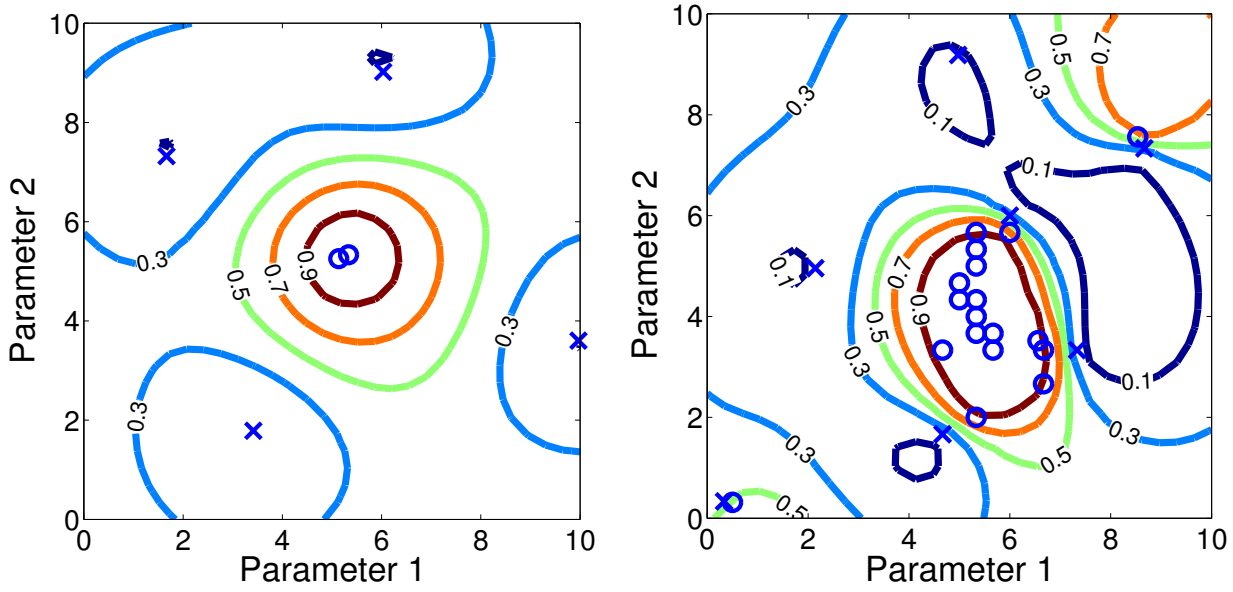
$$C(x) = \int_{max(\bar{\pi}-\epsilon,0)}^{min(\bar{\pi}+\epsilon,1)} p_{\pi}^{x}(\bar{y}) d\bar{y}. \tag{8.7}$$

I reran the optimization over the same 2D parameter space, not considering points where, for $\epsilon = .1$, $C(x) >= 90\%$. As seen in Fig. 8.6, this generated a more diverse solution set which provided a richer set of motions for the robot.

8.5 Exploiting Task Structure to Solve Difficult Problems

Attempting this same optimization on a taller (9 inch) obstacle resulted in no successes within the first 20 trials; a solution with a non-zero probability of success was sparse enough that this was essentially a blind search of the parameter space. It is preferable to avoid an exhaustive search, even for such problems where the regions with high success probability are sparse within the space.

To address this problem, I used the observation that in many cases, tasks can be param-



(a) Experiment selection using EI_π

(b) Experiment selection using EI_π and ignoring areas of high certainty

Figure 8.6: Selected points and predicted success probability for optimization of robot motion over an obstacle using the EI_π experiment selection metric. The 20 parameters chosen for the 2D optimization are shown as an "O" if they resulted in a success, and an "X" if they resulted in a failure. In (a), the optimization only using the EI_π metric results in pure exploitation after confidently finding a good solution. In (b), avoiding selection of points with a high confidence generates more robust solutions.

eterized so that a spectrum of tasks is defined which contains the difficult task as well as simpler tasks. The insight in this section is that if such a task parameterization is chosen, one can learn the general behavior and location of optima of the objective from one or more simpler optimization problems, and use these as a principled prior for optimization of the difficult task. This idea is similar to the problem definition from Chapter 6, as well as those in [10].

To learn parameters of the expert trajectory for more difficult obstacles, such as the 9 inch beam that provided a challenge, a fourth parameter (obstacle height) was added to the GP function approximation. This generated a success probability posterior for all parameters and all obstacle heights, creating a prior that incorporated previous data for subsequent optimizations. A series of optimizations were then attempted for beams of heights 5.5, 7, and 9 inches. Each GPC regression used all previous collected data (including that from the original 3.5 inch optimization). Fig. 8.7 shows a selected trial for each intermediate task parameter.

As opposed to the initial experimental trial, success was obtained for the 9 inch beam on the first experiment suggested by EI_π, demonstrating that shared knowledge between tasks can improve real-world optimization performance. Parameters for overcoming an 11 inch beam were then successfully predicted with no required optimization (bottom of Fig. 8.1).

Sharing data across additional task parameters further reinforces use of the selection bias against points with high confidence (as defined by Equation (8.7)). The exploration inherent when avoiding these points increases the quality of the prior for the more difficult tasks by better defining the boundaries of regions of high probability for simpler tasks.

Although generalizing results from an easier task to a more difficult task works well for many problems, there are caveats. Common choices for GP covariance functions are axis-aligned, resulting in poorer generalization if a trend across multiple tasks exists with principal direction that is not primarily along the task parameter axes. In addition, if a

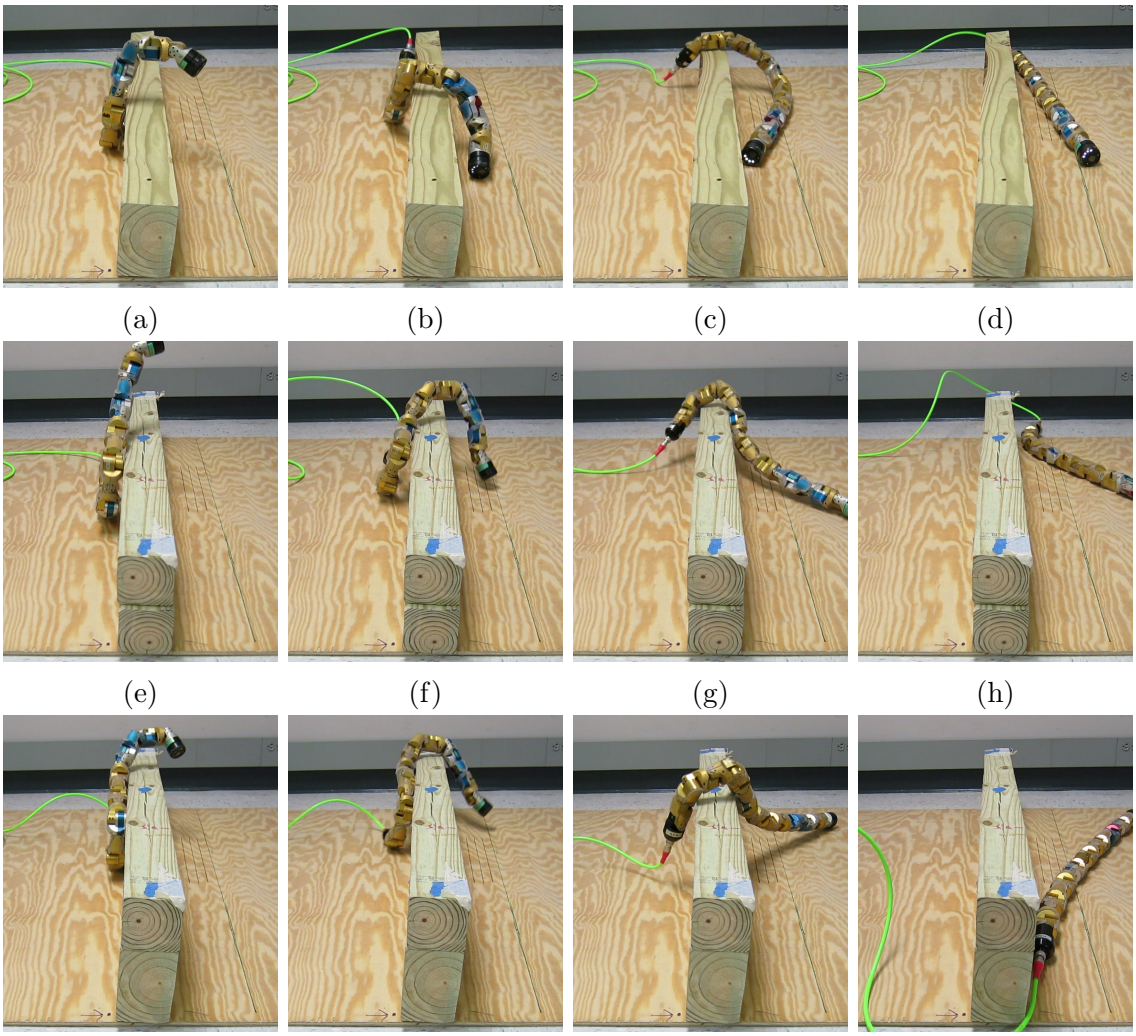


Figure 8.7: Successful trials from optimizations for 5.5 (top), 7 (center), and 9 (bottom) inch obstacles. The robot started at the left side of the obstacle, and moved over the obstacle to the right.

global optima for a difficult task is unrelated to an optima for a simple task, the sharing of knowledge across tasks is less likely to increase efficiency (unless it helps identify global properties of this function that could improve the search).

8.6 Conclusion and Future work

I have defined the stochastic binary optimization problem for expensive functions, presented a novel use of GPC to frame this problem as expensive black-box optimization, and presented a new algorithm using a derivation of EI in the stochastic binary case. This approach outperformed several baseline metrics as well as a leading continuous-armed bandit algorithm. I also used this algorithm to learn a robust motion for moving a snake robot over an obstacle, and used shared task knowledge to efficiently create an adaptive policy for obstacles of various heights.

The problem I define is not limited to the demonstrated snake robot application, but applies to many expensive problems with parameterized policies and stochastic success/failure feedback. This includes variations of applications where continuous-armed bandits are currently used, such as auction mechanisms and oblivious routing (see references in [56]), which could contain an offline training phase penalizing simple rather than continuous regret.

Chapter 9

Conclusion

This thesis has explored expensive optimization in the context of improving snake robot locomotion. I have developed methods addressing optimization in a number of settings, focusing on methods without tuning parameters. The effectiveness of these methods has been demonstrated on many problems, improving the locomotive capabilities of snake robots. In addition, data collection and parameterization methods have been explored for learning from demonstration, opening the way to optimization of new robot capabilities that are not easy to encode programmatically but can be demonstrated to the system.

9.1 The Future of Expensive Optimization

Somewhat surprisingly, the use and knowledge of these methods is not widespread in the machine learning or robotics communities. I believe there are a number of reasons for this, many of which I have attempted to address in some way in this thesis. Below I list several goals that practitioners must work towards in order for these techniques to gain more traction.

Provide More Code

The learning curve is high for expensive optimization methods. Although conceptually simple, there are many "tricks of the trade" necessary for good performance that are learned by practitioners of these methods over years of tinkering with them. For example, should LOO likelihood or marginal log likelihood be used to choose hyperparameters? What is the best covariance function? What selection metric is best (and what parameters should be used for it)? How important is it to explicitly deal with noise? How should one optimize the selection metric?

I have tried to address many of these questions in the thesis, especially in §4.2. Many are also addressed in works from other researchers. But a potential new user must sift through dozens of papers, reading between the lines, and then attempt to use all these lessons learned when writing their own code.

I believe this is one of the primary reasons these methods aren't used – the high learning curve, undocumented necessary expert knowledge, and lack of existing code. This serves as my motivation for releasing code with this thesis. It is not the solution to all of these problems, and there are design and usability decisions I would certainly change if I were to start over. However, it provides a start that I hope is better than a blank slate, and can serve as a stepping stone for improved code in the future.

Simplify Use and Understanding of Gaussian Processes

GPs can be finicky. I have spoken to researchers who have tried using them for months before giving up in frustration, and others who have a list of techniques they prefer besides GPs. I can also speak to this; the choice of covariance function and hyperparameters is the difference between a smooth interpolation of data and a few tall spikes in an otherwise flat plane.

This is another reason why I focused so heavily on the robust fitting of the surrogate

function in Chapter 4, and the reason I attempted to provide a gentle and intuitive introduction to GPs in §2.1. There are still many areas for further improvement of GPs not discussed in this thesis. For example, fitting to sparse data can often lead to many hyperparameter choices with comparable likelihoods. Rather than selecting the MLE hyperparameter, one can integrate over all the hyperparameters weighted by their likelihood (using slice sampling techniques) to give a more robust surrogate. Improved addition of prior information, such as using a prior estimate as the mean function, could greatly benefit practical use of these algorithms.

Fortunately, a software package for GPs does exist; the GPML [80] MATLAB library is actively developed, constantly improving and integrating new research into the codebase. The documentation is excellent, but is best paired with a book describing GPs in detail. I believe it is important that a higher level wrapper for this code is written to improve the accessibility of the GPs.

Consolidate Research from Disparate Fields

The same basic approach is known in various fields as "global optimization", "expensive black-box optimization", "kriging", "design and analysis of computer experiments (DACE)", "Bayesian optimization", and "simple regret for continuum armed bandits", used for different purposes with different terminology. This lack of consistent terminology contributes to confusion, and prevents the spread of advancements across disciplines. Workshops such as the Neural Information Processing Systems (NIPS) Bayesian Optimization workshop have begun to address this by bringing together researchers from various fields, but there is still progress to be made.

9.2 Future Work

There are several areas of the work in this thesis which I believe hold the most promise. First, it is important to address the remaining challenges listed above. I believe the increased quality of GP regression when integrating over all potential hyperparameters could greatly improve optimization algorithm performance, especially during initial steps where the current approach is a uniform random search or space-filling experimental design. However, the resulting non-Gaussian distributions are a consideration, as they complicate many necessary calculations.

A better optimization toolbox (where a deep understanding of low level parameters is not needed) would go a long way in acceptance and adoption of these optimization methods. Incorporating automatic smart parameter selection, such as the model selection work in this thesis, could help these method be usable "out-of-the-box."

Another field which addresses optimization problems with applications to physical robots is reinforcement learning (RL). Both the approaches in this thesis and RL have their strengths and ideal use cases and can seem largely complementary. For example, RL is primarily model-based whereas expensive optimization model-free, RL primarily searches locally while expensive optimization searches globally, etc. The work in [97] attempts to relate these fields in a single general framework by describing a continuum on which both approaches exist. However, there remains an interesting avenue for future work in finding a synthesis of these methods which provides an advantage over each individually. Perhaps a GP could be used to model an expensive system's dynamics, and an RL algorithm is run using the GP as a model; alternatively, introducing the exploration/exploitation tradeoff naturally accomplished with EI into an RL algorithm's update rule could provide for global instead of local optimization.

This thesis's work on learning from massive novice demonstrations provided an interesting preliminary study on the topic. Using the lessons learned and collecting more data could

enable groundbreaking work using existing techniques as well as better inform the clustering described in §7.3. Other untested techniques, such as identifying trajectory modes using kernel principal component analysis (PCA) with a dynamic time warping (DTW) kernel seem promising, and should also be tested with improved data.

Finally, although this thesis demonstrated these techniques on a snake robot, the methods are general and can be applied to many other robotic systems. More demonstrations will further increase the trust in and adoption of expensive optimization techniques, leading to improvements to this simple yet effective concept.

Appendices

Appendix A

Test Function Definitions

A.1 Adaptive Control Test Functions

These are test functions for adaptive control (Chapter 6).

Test function 1

$$f = -\tan^{-1}\left(.2((x_e-4)^2-(x_e-4)x_c+x_c^2)\right)-\tan^{-1}\left(.2((x_e)^2-x_e(x_c-4)+(x_c-4)^2)\right)$$

$$-1 \le x_e \le 5$$
$$-3 \le x_c \le 7$$

Test function 2

$$f = cos(x_c)sin(x_c) - .2 * cos(x_e) + 1.5sin(x_c x_e) + cos(.3x_c) + sin(x_e cos(.2x_c)) + .13x_c - .02x_e$$

$$1 \le x_e \le 20$$
$$1 \le x_c \le 1.5$$

Test function 3 (Same as function 2, with the following bounds)

$$-1 \le x_e \le 5$$
$$-3 \le x_c \le 7$$

A.2 Stochastic Binary Test Functions

These are test functions for stochastic binary optimization (Chapter 8). Bounds used were $0 \leq x \leq 10$ (for each parameter).

Test function 1

$$\Phi\left(\sin(x) - \cos(3x)/4 + \frac{x^3 - 13x^2 - 29x - 55}{50}\right)$$

Test Function 2

$$\sin(5x/4)/4 - \cos(3(x-1)/5)/20 - \frac{5x^3 + 54x^2 - 179x + 159}{100}$$

Test Function 3

$$3\Phi((-40x + 7)/16)/4 + \phi(x - 9/2)/2 + 5\phi(2x - 15)/2$$

Test Function 4

$$\left(\Phi\left(\sin(x_1) - \cos(3x_1)/4 + \frac{x_1^3 - 13x_1^2 - 29x_1 - 55}{50}\right)\right) \times$$
$$\left(\frac{3}{4}\Phi\left(-\frac{5}{2}x_2 + \frac{35}{8}\right) + \frac{9}{5} * \phi\left(\frac{1}{2}x_2 - \frac{9}{4}\right) + 1.64\phi\left(\frac{4}{3}x_2 - 10\right)\right)$$

Test Function 5

$$\left(\sin(5x_1/4)/4 - \cos(3(x_1-1)/5)/20 - \frac{5x_1^3 + 54x_1^2 - 179x_1 + 159}{100}\right) \times$$
$$\frac{3}{4}\Phi\left(-\frac{5}{2}x_1, +\frac{35}{8}\right)\frac{1}{2}\phi\left(x - \frac{9}{2}\right) + \frac{7}{4}\phi\left(\frac{4}{3}x_2 - 10\right)$$

Appendix B

Example Optimization Scripts

The sample scripts provided in this appendix give a brief glimpse at the commands and use of the optimization toolbox provided with this thesis, as used to generate data for this thesis. More complete examples and documentation is available with the code, downloadable at `http://www.mtesch.net/thesisCode/`.

B.1 Model Selection Example Scripts

These commands were used to generate the data for figures in §4.2.4 showing the effect of model selection on EI performance.

```
% Compare optimization with and without model selection

% Set up parameter structure
p = struct();
p.algorithm = 'responseSurface';
p.numTrials = 50;
p.numInitExperiments = 2;
p.numExperiments = 40;
p.retryOnFailure = 1;
p.mins = [0; 0];
p.maxs = [10; 10];
p.costFunction = 'bra_noise';
p.metric = 'expImprove';

% Set up covariance options
GPhypBoth = struct();
GPhypBoth(1).covFunc = {@covSEiso};
GPhypBoth(1).initCovHyps = [0, 0];
GPhypBoth(1).meanFunc = {@meanZero};
GPhypBoth(1).initMeanHyps = [];
```

```
GPhypBoth(1).likFunc = {@likGauss};
GPhypBoth(1).lik = -50;
GPhypBoth(1).infMethod = @infLOO;

GPhypBoth(2).covFunc = {@covSum,{@covSEiso, @covNoise}};
GPhypBoth(2).initCovHyps = [0, 0, 0];
GPhypBoth(2).meanFunc = {@meanZero};
GPhypBoth(2).initMeanHyps = [];
GPhypBoth(2).likFunc = {@likGauss};
GPhypBoth(2).lik = -50;
GPhypBoth(2).infMethod = @infLOO;

% Run simple tests
p.GPhyp = GPhypBoth(1);
optimize(p, 'simple');
% Run complex tests
p.GPhyp = GPhypBoth(2);
optimize(p, 'complex');
% Run auto-selection tests
p.GPhyp = GPhypBoth;
optimize(p, 'auto');
```

B.2 Stochastic EI Example Scripts

These commands were used to generate the data for figures in §4.3.3 showing the effect of incorporating noise into the derivation of the EI selection metric.

```
% Set up params struct
p = struct();
p.algorithm = 'responseSurface';
p.numTrials = 50;
p.numInitExperiments = 5;
p.numExperiments = 20;
p.retryOnFailure = 1;
p.mins = [0];
p.maxs = [10];
p.costFunction = 'f1_1_noise';

% Define covariance functions:
GPhyp = struct();
```

```
GPhyp(1).covFunc = {'covSum',{'covSEiso','covNoise'}};
GPhyp(1).initCovHyps = [0,0,0];
GPhyp(1).meanFunc = 'meanConst';
GPhyp(1).initMeanHyps = [0];
GPhyp(1).likFunc = 'likGauss';
GPhyp(1).lik = -50;
GPhyp(1).infMethod = 'infExact';
p.GPhyp = GPhyp;

% Choose algorithm and metric (if applicable):
p.metric = 'expImproveNoiseNoDist';
%p.metric = 'expImproveNoiseDist';
%p.metric = 'expImproveNoiseDist2d';
%p.metric = 'expImprove';
p.metric_option = 'max_sample';
%p.metric_option = 'max_mu';
%p.metric_option = 'max_mu_sample';
%p.metric_option = 'max_ucb_sample'; p.metric_beta = -2.0;

% Run the test
optimize(p,['result_directory']);
```

B.3 Stochastic Binary Example Scripts

These commands were used to generate the data for figures in Chapter 8.

```
% Build params structure:

p = struct();
%p.algorithm = 'nonBinaryResponseSurface';  % For baselines using GP
p.algorithm = 'binaryResponseSurface'; % For baselines using GPC
p.numTrials = 100;
p.numExperiments = 50;
p.costFunction = 'class1'; % select cost function
p.retryOnFailure = 1;
p.numInitExperiments = 5;
p.initExpSelection = 'latin';
p.numTestPoints = 1000;
p.mins = [0];
p.maxs = [10];
```

```
% Select metric and optional parameters:
%p.metric = 'binaryLatentUCB';
%p.metricBeta = 1.0;
p.metric = 'binaryLatentEI';
%p.metric = 'expImprove';
%p.metric = 'binaryProbabilityEI';
%p.metric = 'random';

% Optional: ignore highly confidence values!
%p.ignoreConfidence = 0.9; p.ignoreConfidenceRange = .1;

% Set up GP parameters
hyp = struct();
hyp.cov = [0.75; 2.5];
hyp.lik = []; % set to -50 if using GP, [] if using GPC
p.gpHyp = hyp;
p.covarianceFunction = {@covSEiso};
p.meanFitMethod = 'zero';

% Run 1-D tests:
for costFunctionNumber = 1:7
p.costFunction = ['class' num2str(costFunctionNumber)];
optimize(p, ['binary-class' num2str(costFunctionNumber) '-results]);
end
% Run 2-D tests:
p.mins = [0 0];
p.maxs = [10 10];
for costFunctionNumber = 8:9
p.costFunction = ['class' num2str(costFunctionNumber)];
optimize(p, ['binary-class' num2str(costFunctionNumber) '-results']);
end
```

Appendix C

Further Information

C.1 Marginal Likelihood

This derivation largely follows that from Rasmussen and Williams textbook on GPs [81].

Recall from §2.1 that a GP represents a probability distribution over functions; computing the marginal likelihood of the data requires marginalizing out the candidate function values, given here as f^*:

$$p(\tilde{\mathbf{y}}|\tilde{\mathbf{x}},\theta) = \int p(\tilde{\mathbf{y}}|f^*,\tilde{\mathbf{x}},\theta)p(f^*|\tilde{\mathbf{x}},\theta)df^* \tag{C.1}$$

Above, the first term in the integrand represents the likelihood of the data given the function f^*; for covariance functions without a noise parameter this equals 0 except for hyperparameters where the function exactly interpolates the data; for covariance functions with a diagonal noise term σ_n^2, $p(\tilde{\mathbf{y}}|f^*,\tilde{\mathbf{x}}) = \mathcal{N}(f^*,\sigma_n^2 I)$. The second term represents the prior over functions, and is given by $\mathcal{N}(0,K)$.

As discussed in [81], the nature of GPs allows this integral to be computed analytically, and the marginal likelihood is given by

$$p(f^*|\tilde{\mathbf{x}},\theta) = \frac{1}{(|K_y|)^{1/2}(2\pi)^{n/2}} e^{-1/2y^T K_y^{-1} y}. \tag{C.2}$$

If there is a stochastic GP with diagonal noise, the covariance matrices K_y should include the addition of the diagonal noise term.

Bibliography

[1] Pieter Abbeel and A.Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004.

[2] Rajeev Agrawal. The Continuum-Armed Bandit Problem. *SIAM Journal on Control and Optimization*, 33(6):1926–1951, November 1995.

[3] Peter Auer, N Cesa-Bianchi, and P Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, pages 235–256, 2002.

[4] Peter Auer, Ronald Ortner, and C Szepesvári. Improved rates for the stochastic continuum-armed bandit problem. *Learning Theory*, 2007.

[5] J. Baker and Johann Borenstein. The Joysnake A Haptic Operator Console for High-Degreeof-Freedom Robots. In *Robotics*, pages 12–15, 2006.

[6] T. Bakker, B. and Heskes. Task Clustering and Gating for Bayesian Multitask Learning. *Journal of Machine Learning Research*, (4):83–99, 2003.

[7] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, 1997.

[8] Jose M. Bernardo. Expected Information as Expected Utility. *The Annals of Statistics*, 7(3):686–690, May 1979.

[9] D. A. Berry and B. Fristedt. *Bandit Problems; Sequential Allocation of Experiments*. Chapman and Hall, New York, 1985.

[10] E Bonilla, F Agakov, and C Williams. Kernel multi-task learning using task-specific features. In *11th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.

[11] Abdeslam Boularias, Jens Kober, and Jan Peters. Model-free inverse reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[12] George E. P. Box and Norman R. Draper. *Empirical model-building and response surfaces*. Wiley, 1987.

[13] L Breiman. Random forests. *Machine learning*, 2001.

[14] R.W. Brockett. Control theory and singular riemannian geometry. In PeterJ. Hilton and GailS. Young, editors, *New Directions in Applied Mathematics*, pages 11–27. Springer New York, 1982.

[15] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in finitely-armed and continuous-armed bandits. *Theoretical Computer Science*, 412(19):1832–1852, April 2011.

[16] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X -Armed Bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.

[17] P. B. Cheung, L. F. Reis, K. T. Formiga, F. H. Chaudhry, and W. G. Ticona. Multiobjective evolutionary algorithms applied to the rehabilitation of a water distribution system: A comparative study. In C. M. Fonseca and Et. Al., editors, *Int. Conf. Evol. Multicriterion Optimization*, pages 662–676, Berlin, Germany, 2003. Springer-Verlag.

[18] G.S. Chirikjian and J.W. Burdick. A modal approach to hyper-redundant manipulator kinematics. *IEEE Transactions on Robotics and Automation*, 10(3):343–354, June 1994.

[19] G.S. Chirikjian and J.W. Burdick. The kinematics of hyper-redundant robot locomotion. *IEEE Transactions on Robotics and Automation*, 11(6):781–793, 1995.

[20] Jiang Chong and R. Srikant. Parametrized Stochastic Multi-armed Bandits with Binary Rewards. *CoRR*, abs/1111.4, 2011.

[21] Jared L. Cohon. *Multiobjective Programming and Planning*. Courier Dover Publications, 2004.

[22] V Coverstone-Carroll. Optimal multi-objective low-thrust spacecraft trajectories. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):387–402, June 2000.

[23] D.D. Cox and S. John. A statistical method for global optimization. In *1992 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1241–1246. Ieee, 1992.

[24] George B. Dantzig. Programming in a Linear Structure. Technical report, Comptroller, USAF, Washington, D.C., 1948.

[25] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182—-197, 2000.

[26] LCW Dixon and GP Szego. The global optimization problem: an introduction. *Towards Global Optimization*, 2:1 – 15, 1978.

[27] Mark Ebden. Gaussian Processes for Regression: A Quick Introduction. Technical Report August 2008.

[28] Michael Emmerich and Jan-willem Klinkenberg. The computation of the expected improvement in dominated hypervolume of Pareto front approximations. Technical Report 1, Leiden Institute for Advanced Computer Science, 2008.

[29] V. Federov. *Theory of Optimal Experiments*. Academic Press, 1972.

[30] Roman Garnett, Yamuna Krishnamurthy, Xuehan Xiong, Jeff Schneider, and Richard P Mann. Bayesian optimal active search and surveying. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, 2012.

[31] J. Gonzalez-Gomez, H. Zhang, Eduardo Boemo, and Jianwei Zhang. Locomotion capabilities of a modular robot with eight pitch-yaw-connecting modules. In *9th international conference on climbing and walking robots*. Citeseer, 2006.

[32] Grzegorz Granosik and Johann Borenstein. Serpentine Robots for Industrial Inspection and Surveillance. *Industrial Robotics.*, (February):633–662, 2007.

[33] Hans-Martin Gutmann. A Radial Basis Function Method for Global Optimization. *Journal of Global Optimization*, 19, 1999.

[34] Masaya Hara, Shogo Satomura, Hiroaki Fukushima, Tetsushi Kamegawa, Hiroki Igarashi, and Fumitoshi Matsuno. Control of a Snake-like Robot Using the Screw Drive Mechanism. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, XX(Xx):3883–3888, April 2007.

[35] G. H. Hardy and E. M. Wright. "The Functions theta(x) and psi(x)" and "Proof that theta(x) and psi(x) are of Order x.". In *An Introduction to the Theory of Numbers*, chapter 22, pages pp. 340–342. Clarendon Press, Oxford, England, 5th edition, 1979.

[36] Kazunari Hatazaki, Masashi Konyo, Kazuya Isaki, Satoshi Tadokoro, and Fumiaki Takemura. Active scope camera for urban search and rescue. *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2596–2602, October 2007.

[37] Ross L. Hatton and Howie Choset. Connection vector fields for underactuated systems. *2008 2nd IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics*, pages 451–456, October 2008.

[38] Ross L. Hatton and Howie Choset. Generating Gaits for Snake Robots by Annealed Chain Fitting and Keyframe Wave Extraction. In *IEEE International Conference on Intelligent Robots and Systems*, pages 840–845, St. Louis, USA, 2009.

[39] Ross L. Hatton and Howie Choset. Optimizing Coordinate Choice for Locomoting Systems. In *Proceedings of the IEEE International Conference on Robotics and*, July 2010.

[40] B Y Shigeo Hirose and Hiroya Yamada. Snake-Like Robots. *IEEE Robotics & Automation Magazine*, (March):88–98, 2009.

[41] Kenneth Holmström. An adaptive radial basis algorithm (ARBF) for expensive black-box global optimization. *Journal of Global Optimization*, 41(3), 2008.

[42] James K Hopkins, Brent W Spranklin, and Satyandra K Gupta. A survey of snake-inspired robot designs. *Bioinspiration & biomimetics*, 4(2):021001, June 2009.

[43] D. Huang, T. T. Allen, W. I. Notz, and N. Zeng. Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models. *Journal of Global Optimization*, 34(3):441–466, March 2006.

[44] D. Huang, T. T. Allen, W. I. Notz, and N. Zeng. Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models. *Journal of Global Optimization*, 34(3), 2006.

[45] Frank Hutter, HH Hoos, and K Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *Learning and Intelligent Optimization*, 2011.

[46] A.J. Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning attractor landscapes for learning motor primitives. *Advances in neural information processing systems*, 15:1523–1530, 2003.

[47] Antony Jameson and J.C. Vassberg. Computational fluid dynamics for aerodynamic design: Its current and future impact. *Fluid Dynamics*, 538, 2001.

[48] Aaron Johnson, Cornell Wright, Matthew Tesch, Kevin Lipkin, and Howie Choset. A Novel Architecture for Modular Snake Robots. Technical report, Robotics Institute, 2011.

[49] Donald R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

[50] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4), 1998.

[51] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274. Morgan Kaufmann, 2002.

[52] Ashish Kapoor, Kristen Grauman, Raquel Urtasun, and Trevor Darrell. Active Learning with Gaussian Processes for Object Categorization. *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, October 2007.

[53] A. J. KEANE. Statistical improvement criteria for use in multiobjective design optimization. *AIAA journal*, 44(4):879–891, 2006.

[54] Michael Kearns, Y. Mansour, and A.Y. Ng. Approximate planning in large POMDPs via reusable trajectories. *Advances in Neural Information Processing Systems*, 12:1001–1007, 2000.

[55] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[56] Robert Kleinberg. Nearly tight bounds for the continuum-armed bandit problem. In *Advances in Neural Information Processing Systems*, pages 697–704, 2004.

[57] Robert Kleinberg and Eli Upfal. Multi-Armed Bandits in Metric Spaces. In *STOC '08 Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 681–690, 2008.

[58] J. Knowles. ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, February 2006.

[59] Jens Kober and Jan Peters. Learning motor primitives for robotics. In *IEEE International Conference on Robotics and Automation*, pages 2112–2118. IEEE, 2009.

[60] Harold J Kushner. A new method for locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, 1964.

[61] Akina Kuwada, Shuichi Wakimoto, Koichi Suzumori, and Yudai Adomi. Automatic pipe negotiation control for snake-like robot. *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, (438):558–563, July 2008.

[62] J.S. Lehman. *Sequential Design of Computer Experiments for Robust Parameter Design*. PhD thesis, Ohio State University, 2002.

[63] P. Liljeback, K.Y. Pettersen, O. Stavdahl, and J.T. Gravdahl. Experimental Investigation of Obstacle-Aided Locomotion With a Snake Robot. *Robotics, IEEE Transactions on Robotics*, PP(99):1–8, 2011.

[64] Pal Liljeback, Kristin Y Pettersen, Ø yvind Stavdahl, and Jan Tommy Gravdahl. A hybrid model of obstacle-aided snake robot locomotion. In *2010 IEEE International Conference on Robotics and Automation*, pages 675–682. IEEE, May 2010.

[65] Tyler Lu, Dávid Pál, and Martin Pál. Showing Relevant Ads via Lipschitz Context Multi-Armed Bandits. *13th International Conference on Artificial Intelligence and Statistics*, 2010.

[66] Michael D. McKay, Richard J. Beckman, and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239 – 245, 1979.

[67] Matt Might. Community research and academic programming license.

[68] Lilyana Mihalkova and Raymond Mooney. Using active relocation to aid reinforcement learning. In *Proceedings of the 19th International FLAIRS Conference*, number May, pages 580–585, 2006.

[69] Thomas P. Minka. *A family of algorithms for approximate Bayesian inference*. Phd thesis, Massachusetts Institute of Technology, 2001.

[70] J Mockus, V Tiesis, and A Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2:117–129, 1978.

[71] Andrew W Moore and Jeff Schneider. Memory-based stochastic optimization. *Advances in Neural Information Processing Systems*, pages 1066–1072, 1996.

[72] B. Naujoks, L. Willmes, T. Back, and W. Haase. Evaluating Multi-criteria Evolutionary Algorithms for Airfoil Optimisation. *LECTURE NOTES IN COMPUTER SCIENCE*, (2439):841–850, 2003.

[73] John A. Nelder and Roger Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4), January 1965.

[74] M Neumann, P. Labenda, T. Predki, and L. Heckes. Snake-like, tracked, mobile robot with active flippers for urban search-and-rescue tasks. In *15th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR)*, 2012.

[75] Hannes Nickisch and CE Rasmussen. Approximations for binary Gaussian process classification. *Journal of Machine Learning Research*, 9:2035–2078, 2008.

[76] H. Ohno and S. Hirose. Design of slim slime robot and its gait of locomotion. *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, pages 707–715, 2001.

[77] V Picheny, D Ginsbourger, and Y Richet. Noisy Expected Improvement and on-line computation time allocation for the optimization of simulators with tunable fidelity. In *2nd International Conference on Engineering Optimization*, pages 1–10, Lisbon, Portugal, 2010.

[78] Dean Pomerleau. ALVINN: An Autonomous Land Vehicle In a Neural Network. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1989.

[79] Wolfgang Ponweiser, Tobias Wagner, and Markus Vincze. Clustered multiple generalized expected improvement: A novel infill sampling criterion for surrogate models. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3515–3522. Ieee, June 2008.

[80] Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian Processes for Machine Learning.

[81] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[82] Nathan Ratliff, David Bradley, J.A. Bagnell, and Joel Chestnutt. Boosting structured prediction for imitation learning. In *Advances in Neural Information Processing Systems*, 2006.

[83] Rommel G. Regis and Christine A. Shoemaker. A Stochastic Radial Basis Function Method for the Global Optimization of Expensive Functions. *INFORMS Journal on Computing*, 19(4), 2007.

[84] Herbert Robbins. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.*, 58, 1952.

[85] David Rollinson, Austin Buchan, and Howie Choset. State Estimation for Snake Robots. *IEEE International Conference on Intelligent Robots and Systems*, pages 1075–1080, 2011.

[86] David Rollinson and Howie Choset. Virtual Chassis for Snake Robots. *IEEE International Conference on Intelligent Robots and Systems (accepted)*, 2011.

[87] AM Ross. Useful bounds on the expected maximum of correlated normal variables. 2003.

[88] S. Ross and J.A. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9, 2010.

[89] K D Rothley, Oswald J Schmitx, and Jared L Cohon. Foraging to balance conflicting demands : novel insights from grasshoppers under predation risk. *Behavioral Ecology*, 8(5):551–559, 1997.

[90] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. *Adaptive Motion of Animals and Machines*, 2006.

[91] John Schulman, Ankush Gupta, Sibi Venkatesan, Mallory Tayson-Frederick, and Pieter Abbeel. A Case Study of Trajectory Transfer Through Non-Rigid Registration for a Simplified Suturing Scenario. In *26th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[92] John Schulman, Jonathan Ho, Cameron Lee, and Pieter Abbeel. Learning from Demonstrations Through the Use of Non-Rigid Registration. In *16th International Symposium on Robotics Research (ISRR)*, 2013.

[93] Burr Settles. Active Learning Literature Survey. Technical report, University of Wisconsin–Madison, 2009.

[94] E. Shammas, H. Choset, and A. Rizzi. Natural gait generation techniques for principally kinematic mechanical systems. In *Proceedings of Robotics: Science and Systems*. Citeseer, 2005.

[95] Snelson, Ed (Gatsby Computational Neuroscience Unit, UCL). Tutorial: Gaussian process models for machine learning, 2006.

[96] Rajendra S Solanki, Perry A Appino, and Jared L Cohon. Theory and Methodology Approximating the noninferior set in multiobjective linear programming problems. *European Journal Of Operational Research*, 68:356–373, 1993.

[97] Freek Stulp and Olivier Sigaud. Policy Improvement Methods: Between Black-Box Optimization and Episodic Reinforcement Learning. (hal-00738463):1–34, 2012.

[98] S Sundararajan and SS Keerthi. Predictive approaches for choosing hyperparameters in Gaussian processes. *Neural Computation*, 2001.

[99] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[100] Tzyh-Jong Tarn, Shan-Ben Chen, and Gu Fang, editors. *Robotic Welding, Intelligence and Automation*, volume 88 of *Lecture Notes in Electrical Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[101] Matthew Tesch, Kevin Lipkin, Isaac Brown, Ross Hatton, Aaron Peck, Justine Rembisz, and Howie Choset. Parameterized and Scripted Gaits for Modular Snake Robots. *Advanced Robotics*, 23(9):1131–1158, June 2009.

[102] E. Theodorou, Jonas Buchli, and S. Schaal. Learning policy improvements with path integrals. In *International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, pages 828–835, 2010.

[103] Emmanuel Vazquez and Julien Bect. Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *Journal of Statistical Planning and Inference*, 140(11):3088–3095, 2010.

[104] Emmanuel Vazquez, Julien Villemonteix, Maryan Sidorkiewicz, and Éric Walter. Global optimization based on noisy evaluations: An empirical study of two statistical approaches. *Journal of Physics: Conference Series*, 135:012100, November 2008.

[105] Antanas Žilinskas. A review of statistical models for global optimization. *Journal of Global Optimization*, 2(2):145–153, June 1992.

[106] B.J. Williams, T.J. Santner, and W.I. Notz. Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica*, 10(4):1133–1152, 2000.

[107] C. Wright, A. Buchan, B. Brown, J. Geist, M. Schwerin, D. Rollinson, M. Tesch, and H. Choset. Design and Architecture of the Unified Modular Snake Robot. In *2012 IEEE International Conference on Robotics and Automation*, St. Paul, MN, 2012.

[108] Cornell Wright, Aaron Johnson, Aaron Peck, Zachary McCord, Allison Naaktgeboren, Philip Gianfortoni, Manuel Gonzalez-Rivero, Ross L. Hatton, and Howie Choset. Design of a Modular snake robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2609–2614. IEEE, October 2007.

[109] H. Yamada and S. Hirose. Study on the 3D shape of active cord mechanism. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2890–2895.

[110] Milan Zeleny. *Linear Multiobjective Programming*. Springer-Verlag, 1974.

[111] Xuan Zhang, B. John Oommen, and Ole-Christoffer Granmo. Generalized Bayesian pursuit: A novel scheme for multi-armed Bernoulli bandit problems. *Artificial Intelligence Applications and Innovations*, 364:122–131, 2011.

[112] Anatoly Zhigljavsky and Antanas Žilinskas. *Stochastic Global Optimization*. Springer, 2008.

[113] Eckart Zitzler and Lothar Thiele. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. pages 292–304, September 1998.