LEARNING CONTROL VIA PROBABILISTIC TRAJECTORY OPTIMIZATION

A Dissertation
Presented to
The Academic Faculty

By

Yunpeng Pan

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Robotics in the
School of Aerospace Engineering

Georgia Institute of Technology

December 2017

LEARNING CONTROL VIA PROBABILISTIC TRAJECTORY OPTIMIZATION

To my parents.

ACKNOWLEDGEMENTS

I would like to thank several people for their help leading to the completion of this thesis. First and foremost, I have immense gratitude for my advisor Evangelos Theodorou, for providing invaluable guidance toward interesting research directions and for being supportive throughout my PhD journey. In particular, I am thankful for Evangelos' confidence in me during the early phase of my PhD study when I was exploring research topics.

I am indebted to Professor Byron Boots, for guiding me to explore new research areas. His perspectives on machine learning were very helpful which motivate my thinking. It was a great pleasure to work with him. I am grateful for Professor Eric Johnson for serving as my advisor during my master study in aerospace engineering, and I would like to express my gratitude to my committee members Professor Jonathan How and Professor Le Song for their insightful comments.

I am fortunate to have many collaborators and co-authors to work with. They include my labmates: Kaivalya Bakshi, Manan Gandhi, George Boutselis, Kamil Saigol, Keuntaek Lee, Harleen Brar; students from the Professor Boots' group: Xinyan Yan and Ching-An Cheng; Professor Panos Tsiotras and his students Wei Sun, Yannis Exarchos and Alfredo Valverde; Bo Dai from Professor Le Song's group; Professor Babak Mahmoudi and Mark Connolly from Emory University; Pierre Sebastian and Professor Demetri Yannopoulos at the University of Minnesota. I would like to thank Brian Goldfain and Paul Drew from Professor Jim Rehg's lab for their help on the AutoRally platform. I am also thankful for many colleagues with whom I had countless interesting discussions. They include current and former members in the autonomous control and decision system lab: Grady Williams, David Fan, Andrew Aldrich, Ethan Evans, Andrew Fillingim, Marcus Pereira, Pat Wang, Emre Yilmaz; members in the robot learning lab: Nolan Wagener, Mustafa Mukadam, Sasha Lambert, Gabriel Nakajima An; Yiming Zhao, Oktay Arslan, and Imon Chakraborty from Professor Panos Tsiotras's group; Yichen Wang from Professor Le Song's group;

Hancao Li from Professor Haddad's group; Zhaoyang Lv from Professor Jim Rehg's group.

Finally, I would like to thank my family, especially my wife Di for offering support and having patience and confidence in me.

TABLE OF CONTENTS

Acknowledgments v

List of Tables xii

List of Figures xiii

Chapter 1: Introduction 1

1.1 Motivation 1

1.1.1 Progress in Artificial Intelligence 1

1.1.2 Challenges in AI-based Robot Control 2

1.2 Objective and Scope of this Thesis 5

1.2.1 Reinforcement Learning and Optimal Control 5

1.2.2 Probabilistic Modeling and Inference 6

1.2.3 Imitation Learning and Deep Learning 6

1.2.4 Applications to High-Speed Autonomous Driving 7

1.2.5 Contributions and Outline 8

Chapter 2: Technical Background 11

2.1 Optimal Control and Reinforcement Learning 11

2.1.1 Problem Formulation 11

2.1.2 Dynamic Programming 12

2.1.3 Trajectory Optimization via Local Approximations 14

2.1.4 Path Integral and Linearly Solvable Optimal Control 16

2.2 Supervised Learning using Gaussian Processes 19

2.2.1 Supervised Learning and Bayesian Linear Regression 19

2.2.2 Gaussian Processes for Regression 20

Chapter 3: Probabilistic Differential Dynamic Programming 24

3.1 Introduction 24

3.2 Preliminaries 26

3.3 Trajectory Prediction via GP Inference 28

3.3.1 Approximate Inference via Moment Matching 29

3.3.2 Hyperparameter Optimization 31

3.3.3 Incorporating Prior Model Knowledge 31

3.4 Probabilistic Trajectory Optimization 36

3.4.1 Belief Dynamics and Local Approximation 37

3.4.2 Optimization Criterion 38

3.4.3 Control Policy and Value Function Approximation 41

3.4.4 Control constraint 43

3.4.5 Summary of algorithm 44

3.5 Theoretical Analysis 44

3.6 Further Analysis 53

3.6.1 Computational complexity 53

3.6.2 Relation to existing works 53

3.7 Experimental Evaluation 55

3.7.1 Tasks 56

3.7.2 Data efficiency 59

3.7.3 Computational efficiency 60

3.7.4 Nonparametric vs. semiparametric learning 60

3.7.5 Risk-sensitive vs. risk-neutral learning 61

3.8 Discussion 62

Chapter 4: Path Integral Control under Uncertain Dynamics 67

4.1 Introduction 67

4.2 Gradient-based Approach 68

4.2.1 Preliminaries 69

4.2.2 Linearized Hamilton-Jacobi-Bellman Equation for Uncertain Dynamics 71

4.2.3 Relation to Existing Works 73

4.2.4 Analytic Path Integral Control: a Forward-Backward Scheme 74

4.2.5 Generalization to Unlearned Tasks without Sampling 77

4.2.6 Experiments and Analysis 78

4.2.7 Summary and Discussion 82

4.3 Sampling-based Approach 82

4.3.1 Preliminaries 83

4.3.2 Model Learning via Sparse Spectrum Gaussian Processes 84

4.3.3 A Path Integral Control Approach with Covariance Adaptation . . . 87

4.3.4 Relation to Existing Works 93

4.3.5 Experiments and Analysis 95

4.3.6 Summary and Discussion 98

Chapter 5: Prediction under Uncertainty in Sparse Spectrum Gaussian Processes 99

5.1 Introduction 99

5.2 Sparse Spectral Representation of GPs 102

5.3 Prediction under Uncertainty 105

5.3.1 Exact moment matching (SSGP-EMM) 105

5.3.2 Linearization (SSGP-Lin) 109

5.4 Applications 111

5.4.1 Bayesian filtering 112

5.4.2 Stochastic Model Predictive Control 114

5.5 Experimentals and Analysis 117

5.5.1 Bayesian filtering 117

5.5.2 Model Predictive Control 119

5.5.3 Additional experiments on approximate inference 121

5.6 Discussion 124

5.6.1 Conditional independence between outputs 124

5.6.2 SSGP-EKF vs. SSGP-ADF 124

5.7 Summary 126

Chapter 6: Deep Imitation Learning for Agile Autonomous Driving 127

6.1 Introduction 127

6.2 Relation to Existing Works 129

6.3 Imitation Learning for Autonomous Driving 131

6.3.1 Problem Definition 131

6.3.2 Imitation Learning 132

6.3.3 Comparison of Imitation Learning Algorithms 136

6.4 The Autonomous Driving System 137

6.4.1 Algorithmic Expert with Model-Predictive Control 138

6.4.2 Learning a DNN Control Policy 139

6.4.3 The Autonomous Driving Platform 140

6.5 Experimental Setup 141

6.5.1 High-speed Navigation Task 141

6.5.2 Test Track 142

6.5.3 Data Collection 142

6.5.4 Policy Learning 143

6.6 Experimental Results 143

6.6.1 Online vs Batch Learning 143

6.6.2 Deep Neural Network Policy 145

6.7 Summary 145

Chapter 7: Conclusions 150

References 163

LIST OF TABLES

3.1 Comparison with DDP-related frameworks 55

4.1 Comparison with some notable and recent path integral-related approaches. 74

4.2 Comparison between our method under unknown dynamics and the iterative PI control with known dynamics model for the cart-pole swing up task. 96

5.1 Examples of continuous shift-invariant positive-definite kernels and their corresponding spectral densities, where $r = \frac{\sqrt{2\nu}\|x-x'\|_2}{\ell}$, K_ν is a modified Bessel function, and $h = \frac{2^d \pi^{\frac{d}{2}} \Gamma(\nu+\frac{d}{2})(2\nu)^\nu}{\Gamma(\nu)\ell^{2\nu}}$. 109

5.2 Comparison of our proposed methods and GP-EMM [49, 69] in terms of computational complexity and generalizability. 111

5.3 Comparison of our methods with GP-ADF [70] and GP-EKF [112] in terms of average NL_x (negative log-likelihood) of the ground truth states given estimates and RMSE (root-mean-square error). Lower values are better. The results correspond to the filtering task in sec 5.5.1. 118

6.1 Comparison of our method to prior work on imitation learning for autonomous driving 129

6.2 Test statistics. Total loss denotes the imitation loss in (6.6), which is the average of the steering and the throttle losses. Completion ratio is defined as the ratio of the traveled time steps to the targeted time steps (3,000). All results here represent the average performance over three independent evaluation trials. 144

LIST OF FIGURES

1.1 Media coverage of Deep Blue (left) and AlphaGo (right) achieving superhuman performance in chess and the game of Go. 2

1.2 Left: an autonomous helicopter successfully performs inverted flight. Right: a humanoid robot fails to maintain balance while stepping out of a vehicle. . 3

1.3 In reinforcement learning, an autonomous agent interacts with the environment by applying a control. As a consequence, the agent receives a cost (or reward) and a new state or observation. The goal is to find a sequence of control to minimize the accumulated future cost. 5

2.1 An one-dimensional example of GP inference. We use a prior mean function $\mathbf{m}(\mathbf{x}) = 0.63$ and covariance function (2.25). Black asterisks are noisy samples drawn from $f(\mathbf{x})$. Orange solid line is the GP predictive mean, and shaded area represents standard deviation for each input value $\mathbf{x}$. Note that the prediction mainly depends on the prior when the input is far away from observations. 22

3.1 One-dimensional examples of expectation of cost (a), variance of cost (b), risk-averse (c) and risk-seeking cost fucntions (d). The target state is zero. . 40

3.2 CDIP and Puma-560 tasks. 57

3.3 Cross-entropy parameter optimization for CDIP dynamics model with bad initial guesses. The horizontal axis is iteration number and vertical axis is parameter value. Dash lines are the true parameter values. Error bars show the mean and variance of sampling distributions at each iteration. Note that some parameters converge to the true values while some others converge to local minima. 58

3.4 Comparison between PDDP and PILCO in terms of the number of interactions with the physical system (a) and total computational time (b) required for learning the CDIP and Puma-560 tasks. PDDP-NP and PDDP-SP correspond to the nonparametric and semiparametric cases, respectively. The results were averaged over 5 independent trials. 59

3.5 Comparison of the predicted trajectory cost distributions between risk-neutral (a,b) and risk-sensitive (c,d) learning via PDDP. Early stage and final stage correspond to the predictions after 1 and 8 optimization stages. 62

3.6 Total trajectory costs by applying PDDP policy after each optimization stage for risk-neutral ($\epsilon = 0$) and risk-sensitive learning ($\epsilon = 0.2$). 3 and 5 rollouts were used for learning in both cases. 63

3.7 The quadrotor flight task. (a) Quadrotor simulation environment. (b) Expected trajectory cost reduction during PDDP optimization. (c) Trajectory costs over 10 independent trials using the optimized control policy. (d) State trajectories of the quadrotor task collected over 10 independent trials using the optimized controller. Dash lines are desired states. 64

3.8 Constrained and unconstarined control inputs (4 motor torques) for the quadrotor task. The constraint is incorporated by solving a QP (sec.3.4.4) . 65

4.1 Comparison in terms of sample efficiency and computational efficiency for (a) cart-pole and (b) double pendulum on a cart swing-up tasks. Left subfigures show the terminal desirability Ψ_T (for PILCO and PDDP, Ψ_T is computed using terminal state costs) at each trial. Right subfigures show computational time (in minute) at each trial. 80

4.2 Resutls for the PUMA-560 tasks. (a) 8 tasks tested in this experiment. Each number indicates a corresponding target posture. (b) Comparison of the controllers learned independently from trials and the composite controllers without sampling. Each composite controller is obtained (4.19) from 7 other independent controllers learned from trials. 81

4.3 (a) Sampling of belief trajectories via probabilistic inference. The solid line and error ellipse represent predictive mean and variance of the state, respectively. The computation of probabilistic inference can be distributed in parallel. (b) Cart-pole swing up task. 95

4.4 Comparison in terms of total trajectory cost reduction at each iteration during optimization. 97

4.5 Trajetory costs collected by executing optimized controls on the true dynamics model (5 independent trials). 97

5.1 Black points are ground truth states, red areas are filter distributions for (a) GP-ADF [70], (c) GP-EKF [112], our proposed methods (b) SSGP-ADF and (d) SSGP-EKF. The x-axis is the mean of initial belief $p(x_0)$, which is randomly distributed in $[-10, 10]$ and y-axis shows the mean and twice the standard deviation of filtered distribution $p(x_1|y_1)$ after observing y_1. 116

5.2 Recursive filtering task for high-speed autonomous driving. Figure (b) shows trajectories of all the states of a 30 seconds continuous driving (1,200 steps), where blue lines are the ground truth, and red lines and red areas are the mean and twice the standard deviation of the filtered distributions respectively. In (c), the red line and area are the mean and twice the standard deviation of NL_x over six 30 seconds driving with varying number of features. 117

5.3 PUMA-560 and quadrotor tasks 120

5.4 Comparison of the drifting performance using 50 (left), 150 (middle) and 400 (right) random features. Blue lines are the solution provided in [124]. Performance improves with a larger number of features, and with a moderate number of features, MPC with SSGP-Lin behaves very closely to the ground truth solution. 121

5.5 Cost comparison for arm and quadrotor tasks. 122

5.6 (a)-(b): Approximate inference accuracy test. The vertical axis is the squared error of cost predictions for (a) quadrotor system and (b) Puma 560 system. Error bars represent standard deviations over 10 independent rollouts. (c)-(d): Comparison of computation time on a log scale between (c) SSGP-Lin and GP-EMM; (d) SSGP-EMM and GP-EMM. The horizontal axis is the input and output dimension (equal in this case). Vertical axis is the CPU time in seconds. 123

5.7 KL divergences between SSGP-EMM and GP-EMM, SSGP-Lin and GP-Lin 125

6.1 High-speed off-road driving task 128

6.2 System diagram 137

6.3 The DNN control policy 137

6.4 The AutoRally car and the test track. 147

6.5 Examples of vehicle trajectories. (a) Demonstration of the MPC expert. (b) Crashing case when using batch imitation learning. (c) The vehicle avoids crashing when using online imitation learning. Subfigures (b) and (c) depict test runs of the policies after training on 9,000 samples 148

6.6 Performance 148

6.7 From left to right: input RGB image, averaged feature maps for each max-pooling layer. 149

6.8 Performance comparison between our DNN policy and its CNN sub-network in terms of batch imitation learning loss, where the horizontal axis is the size of data used to train the neural network policies. 149

SUMMARY

A central problem in the field of robotics is to develop real-time planning and control algorithms for autonomous systems to behave intelligently under uncertainty. While classical optimal control provides a general theoretical framework, it relies on strong assumption of full knowledge of the system dynamics and environments. Alternatively, modern reinforcement learning (RL) offers a computational framework for controlling autonomous systems with minimal prior knowledge and user intervention. However, typical RL approaches require many interactions with the physical systems, and suffer from slow convergence. Furthermore, both optimal control and RL have the difficulty of scaling to high-dimensional state and action spaces.

In order to address these challenges, we present probabilistic trajectory optimization methods for solving optimal control problems for systems with unknown or partially known dynamics. Our methods share two key characteristics: (1) we incorporate explicit uncertainty into modeling, prediction and decision making using Gaussian processes; (2) our algorithms bypass the *curse of dimensionality* via local approximation of the value function or linearization of the Hamilton-Jacobi-Bellman (HJB) equation. Compared to related approaches, our methods offer superior combination of data efficiency and scalability. We present experimental results and comparative analyses to demonstrate the strengths of the proposed methods.

In addition, we develop fast Bayesian approximate inference methods which enable probabilistic trajectory optimizer to perform real-time receding horizon control. It can be used to train deep neural network controllers that map raw observations to actions directly. We show that our approach can be used to perform high-speed off-road autonomous driving with low-cost sensors, and without on-the-fly planning and optimization.

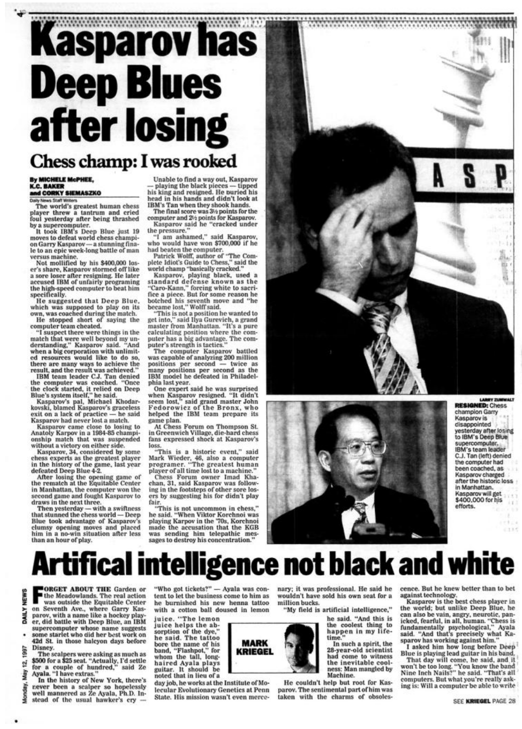
CHAPTER 1
INTRODUCTION

1.1 Motivation

Artificial intelligence (AI) empowers machines to behave intelligently. Researchers across the globe have been working on AI since 1950s [1]. In over half a century after its ‘birth’ as an academic field, there were ups and downs in AI in terms of funding, public interest and commercial success [1]. In the following, we motivate this thesis by discussing the progress and challenges in AI and robotics in recent decades.

1.1.1 Progress in Artificial Intelligence

Over the last few decades, the fields of AI and robotics have witnessed remarkable progress in the development autonomous systems that could achieve human-level performance in specific tasks, thanks to the technological advancement of computer processing power and data storage. For instance, IBM researchers have pioneered the development of game-learning computer program ‘TD-Gammon’ [2], and the chess-playing supercomputer ‘Deep Blue’ [3] in 1990s. While ‘TD-Gammon’ performed slightly worse than expert human backgammon players, ‘Deep Blue’ achieved unprecedented superhuman performance, beating the world chess champion Garry Kasparov [3]. It was one of the first and biggest public leaps forward in AI (see fig. 1.1). However, the impressive performance of ‘Deep Blue’ relies on brute force computation provided by a massively parallel computer system, and rule-based reasoning programmed by AI researchers. Two decades later, the capabilities of storing and processing massive amount of data, and the rise of *Deep Learning* have empowered researchers to address more challenging problems. For example, Google researchers presented the ‘Deep Q Network’ [4] to play Atari games and ‘AlphaGo’ [5] to play the



Kasparov has Deep Blues after losing

Chess champ: I was rooked

By MICHELE McPHEE, K.C. BAKER and CORKY SIEMASZKO
Daily News Staff Writers

The world's greatest human chess player threw a tantrum and cried foul yesterday after being thrashed by a supercomputer.

It took IBM's Deep Blue just 19 moves to defeat world chess champion Garry Kasparov — a stunning finale to an epic week-long battle of man versus machine.

Not mollified by his $400,000 loser's share, Kasparov stormed off like a sore loser after resigning. He later accused IBM of unfairly programing the high-speed computer to beat him specifically.

He suggested that Deep Blue, which was supposed to play on its own, was coached during the match.

He stopped short of saying the computer team cheated.

"I suspect there were things in the match that were well beyond my understanding," Kasparov said. "And when a big corporation with unlimited resources would like to do so, there are many ways to achieve the result, and the result was achieved."

IBM team leader C.J. Tan denied the computer was coached. "Once the clock started, it relied on Deep Blue's system itself," he said.

Kasparov's pal, Michael Khodarkovski, blamed Kasparov's graceless exit on a lack of practice — he said Kasparov had never lost a match.

Kasparov came close to losing to Anatoly Karpov in a 1984-85 championship match that was suspended without a victory on either side.

Kasparov, 34, considered by some chess experts as the greatest player in the history of the game, last year defeated Deep Blue 4-2.

After losing the opening game of the rematch at the Equitable Center in Manhattan, the computer won the second game and fought Kasparov to draws in the next three.

Then yesterday — with a swiftness that stunned the chess world — Deep Blue took advantage of Kasparov's clumsy opening moves and placed him in a no-win situation after less than an hour of play.

Unable to find a way out, Kasparov — playing the black pieces — tipped his king and resigned. He buried his head in his hands and didn't look at IBM's Tan when they shook hands.

The final score was 3½ points for the computer and 2½ points for Kasparov.

Kasparov said he "cracked under the pressure."

"I am ashamed," said Kasparov, who would have won $700,000 if he had beaten the computer.

Patrick Wolff, author of "The Complete Idiot's Guide to Chess," said the world champ "basically cracked."

Kasparov, playing black, used a standard defense known as the "Caro-Kann," forcing white to sacrifice a piece. But for some reason he botched his seventh move and "he became lost," Wolff said.

"This is not a position he wanted to get into," said Ilya Gurevich, a grand master from Manhattan. "It's a pure calculating position where the computer has a big advantage. The computer's strength is tactics."

The computer Kasparov battled was capable of analyzing 200 million positions per second — twice as many positions per second as the IBM model he defeated in Philadelphia last year.

One expert said he was surprised when Kasparov resigned. "It didn't seem lost," said grand master John Fedorowicz of the Bronx, who helped the IBM team prepare its game plan.

At Chess Forum on Thompson St. in Greenwich Village, die-hard chess fans expressed shock at Kasparov's loss.

"This is a historic event," said Mark Wieder, 46, also a computer programer. "The greatest human player of all time lost to a machine."

Chess Forum owner Imad Khachan, 31, said Kasparov was following in the footsteps of other sore losers by suggesting his for didn't play fair.

"This is not uncommon in chess," he said. "When Viktor Korchnoi was playing Karpov in the '70s, Korchnoi made the accusation that the KGB was sending him telepathic messages to destroy his concentration."

LARRY ZUMWALT

RESIGNED: Chess champion Garry Kasparov is disappointed yesterday after losing to IBM's Deep Blue supercomputer. IBM's team leader C.J. Tan (left) denied the computer had been coached, as Kasparov charged after the historic loss in Manhattan. Kasparov will get $400,000 for his efforts.

Artifical intelligence not black and white

DAILY NEWS • Monday, May 12, 1997

MARK KRIEGEL

FORGET ABOUT THE Garden or the Meadowlands. The real action was outside the Equitable Center on Seventh Ave., where Garry Kasparov, with a name like a hockey player, did battle with Deep Blue, an IBM supercomputer whose name suggests some starlet who did her best work on 42d St. in those halcyon days before Disney.

The scalpers were asking as much as $500 for a $25 seat. "Actually, I'd settle for a couple of hundred," said Ze Ayala. "I have extras."

In the history of New York, there's never been a scalper so hopelessly well mannered as Ze Ayala, Ph.D. Instead of the usual hawker's cry — "Who got tickets?" — Ayala was content to let the business come to him as he burnished his new henna tattoo with a cotton ball doused in lemon juice. "The lemon juice helps the absorption of the dye," he said. The tattoo bore the name of his band, "Flashpot," for whom the tall, long-haired Ayala plays guitar. It should be noted that in lieu of a day job, he works at the Institute of Molecular Evolutionary Genetics at Penn State. His mission wasn't even mercenary; it was professional. He said he wouldn't have sold his own seat for a million bucks.

"My field is artificial intelligence," he said. "And this is the coolest thing to happen in my lifetime."

In such a spirit, the 28-year-old scientist had come to witness the inevitable coolness: Man mangled by Machine.

He couldn't help but root for Kasparov. The sentimental part of him was taken with the charms of obsolescence. But he knew better than to bet against technology.

Kasparov is the best chess player in the world; but unlike Deep Blue, he can also be vain, angry, neurotic, panicked, fearful, in all, human. "Chess is fundamentally psychological," Ayala said. "And that's precisely what Kasparov has working against him."

I asked him how long before Deep Blue is playing lead guitar in his band.

That day will come, he said, and it won't be too long. "You know the band Nine Inch Nails?" he said. "That's all computers. But what you're really asking is: Will a computer be able to write

SEE **KRIEGEL** PAGE 28

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
CONSERVATION
SONGBIRDS À LA CARTE
Illegal harvest of millions of Mediterranean birds
PAGE 452
RESEARCH ETHICS
SAFEGUARD TRANSPARENCY
Don't let openness backfire on individuals
PAGE 459
POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
Dawkins's calling card forty years on
PAGE 462
NATURE.COM/NATURE
28 January 2016 £10
Vol. 529, No. 7587

Figure 1.1: Media coverage of Deep Blue (left) and AlphaGo (right) achieving superhuman performance in chess and the game of Go.

game of Go. Instead of using brute force computation and rule-based reasoning, the central idea of these approaches is to create a large data set from previous experience using *Reinforcement Learning* (RL) [6], a machine learning paradigm for optimal decision-making (see fig 1.3), and train a complex neural network using this data set in a supervised fashion. These developments have pushed real-world applications of AI to a new level (see fig 1.1). In these tasks, autonomous agents can achieve superhuman performance without explicit rules imposed by human.

1.1.2 Challenges in AI-based Robot Control

Despite all the success, there are some distinct differences between applying AI to decision-making in board (or video) games and control of real physical robots. First, in the aforementioned applications, there is a 'simulator of the world' and an autonomous agent is free to explore this world in simulation. However, the real-world scenario is so uncertain, the

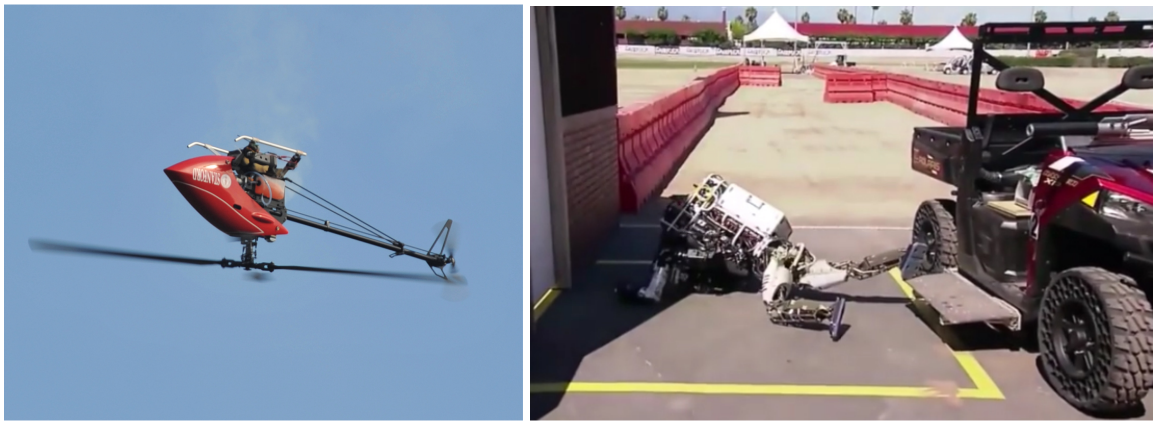


Figure 1.2: Left: an autonomous helicopter successfully performs inverted flight. Right: a humanoid robot fails to maintain balance while stepping out of a vehicle.

gap between the simulator and real world is significantly large, therefore it is very challenging to apply controllers learned from simulation to real-world settings. Second, robot control tasks have continuous state and action spaces, in contrast to the discrete domain in board or video games. Again the aforementioned approaches, e.g., [4, 5] cannot be directly applied to control of physical robots. Third, in board games or robot planning tasks in a 'grid world', we assume that global information such as state and cost (or reward) is provided. However, this information will not be available for physical robotic systems without using expensive sensors, and an explicit, user-designed performance criterion.

In the following we briefly discuss these challenges. The first one is the difficulty of building a 'simulator of the world' for an autonomous agent to explore. In Markov Decision Processes (MDPs), this amount to specifying the transition probability from one state to another, assuming we have access to all states. However, the real dynamics of physical systems and environment are usually highly uncertain. And physics-based models can be very inaccurate. For example, helicopter aerodynamics (fig 1.2) is so complex and not well-understood, there is no explicit model that could accurately predict the helicopter dynamics [7] due to the difficulty of capturing important physical properties such as the effect of inertia. One solution is combining physics-based knowledge with a data-driven model [8, 9]. This hybrid approach has shown impressive results in dynamics modeling and difficult control tasks such as inverted flight. However, in this task, the training data was using human pilot demonstrations, which implies that the task-related state-action spaces of the system have been explored. In addition, when making multi-step predictions using this model, the modeling error will be accumulated and such method does not provide a solution to cope with the effect of modeling error.

The second issue is continuous state and action spaces. Based on the Bellman's *Dynamic Programming* principle [10], optimal decision-making in continuous domains suffer from *curse of dimensionality*: the number of states grows exponentially with the dimensions of the state space. Therefore it is impossible to use brute force computation (such as [3]) to explore the whole state space except for low-dimensional systems (i.e., inverted pendulum). Similarly, the action space is also large, the number of actions increases exponentially with the dimension of the action space. So even learning-based methods such as Q-learning [6, 4] cannot be applied here since they require exploring the whole action space. For example, controlling a humanoid robot (fig 1.2) is extremely difficulty not only because the system is too complex to model, but also because of its high dimensional state and action spaces.

The third challenge is the lacks of information about the state and cost. Decision-making in without fully observable states is usually addressed in partially observable Markov decision process (POMDP) [11] which is a generalization of MDP. However, solving PODMP is know to be computationally intractable. Even approximate solutions does not scale to high-dimensional problems [12]. Regarding the cost or reward function, there are many control tasks that are too complex to be specified explicitly. For instance, driving a car like a human driver is a difficult control problem without a explicit performance criterion, therefore it is not a well-defined RL problem. This challenge leads us to explore alternative solutions outside the typical RL paradigm.

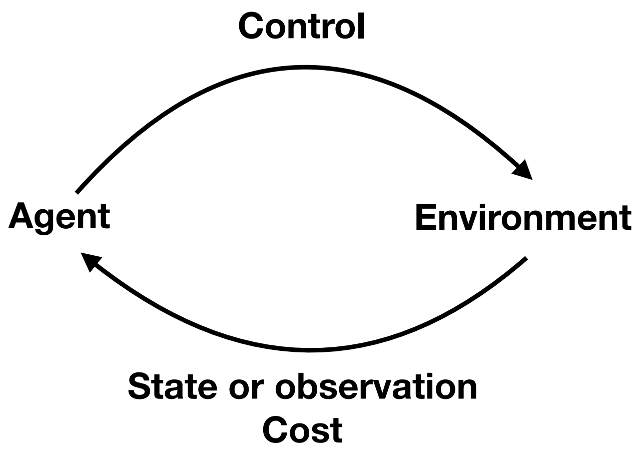


Figure 1.3: In reinforcement learning, an autonomous agent interacts with the environment by applying a control. As a consequence, the agent receives a cost (or reward) and a new state or observation. The goal is to find a sequence of control to minimize the accumulated future cost.

1.2 Objective and Scope of this Thesis

In this section, we introduce the research topics that will be covered in the thesis as well as the structure of this thesis.

1.2.1 Reinforcement Learning and Optimal Control

A main theme of this thesis is solving control problems with minimal knowledge of the system and environment. This can be viewed as a reinforcement learning (RL) problem [6]. RL is one of the three major paradigms in machine learning, besides supervised and unsupervised learning. The high-level idea is learning to perform optimally by interacting with the world. It has little assumption about the system and environment. More precisely, at each time step, an autonomous agent with no prior knowledge about the world interacts with the surroundings by applying controls. The agent receives a cost and a new state or observation as a result of this interaction. The algorithm seeks the actions to minimize the accumulated future cost. See fig 1.3 for a block diagram of RL. Besides engineering, RL has also been applied to solve decision-making problems in other disciplines such as finance and recource management [13]. On the other hand, optimal control concerns the problem of choosing a sequence of actions to minimize a cost function, subject to a known dynamics constraint. Therefore the key difference between RL and optimal control is the knowledge of a dynamics model. Both RL and optimal control are rooted in Bellman's work on *Dynamic Programming* [10] which is the 'Principle of Optimality' for decision-making.

1.2.2 Probabilistic Modeling and Inference

One of the key problems in machine learning is to build models to represent uncertainties that capture statistical dependencies between random variables. A probabilistic model is such a model that describes data that one could observe from a real system, and the uncertainty of the model is expressed using probability theory. Probabilistic inference amounts to predicting the probability distribution over a random variable using a probabilistic model. A complete review of probabilistic modeling and inference is outside the scope of this thesis. We refer the interested reader to [14] for a comprehensive study. Instead we focus on a class of approaches called *Gaussian processes* (GPs) [15]. In contrast to building parametric models and fitting parameters to data, in GPs we specify a prior probability distribution over a function directly. GP models can be applied to nonlinear regression problems, with the goal of predicting the probability distribution over a function value given a training data set. Motivated by the difficulty of modeling complex dynamics, in this thesis we will use GPs to represent dynamics model in a probabilistic fashion.

1.2.3 Imitation Learning and Deep Learning

As mentioned in section 1.1.2, there are many control tasks that are too complex to define explicitly. 'Driving a car like a human driver' is one example. In this case, however, we may take advantage of *expert demonstrations* and reformulate the control problem as a problem of *imitation learning*. The field of imitation learning focuses on developing algorithms with goal of improving performance by mimicking a expert's decisions and behaviors. A key advantage of imitation learning over reinforcement learning is that the complexity of the control task is implicitly incorporated in the expert's demonstration and there is no need to specify the criterion of the control task manually. In fact, imitation learning can significantly simplify the standard robot control pipeline in which the task is decomposed into several different modules such as perception, motion planning and control, see [16] for an example of a robot control pipeline. In contrast, we may learn a mapping from sensory signals to control actions directly given demonstrations. Now the difficulty of RL has been transformed to the difficulty of supervised learning, i.e., learning a complex representation of the control policy given input-output training data where the input data is high-dimensional (e.g., images).

To address this challenge we take advantage of recent breakthroughs in *deep learning* [17]. More specifically, the idea of deep learning is to create a multi-layer architecture of nonlinear functions, the ouput in each layer is the input to the successive layer. In our case, the control policy can be represented by a convolutional neural network (CNN) with multiple layers [18]. The CNN usually has millions of parameters which can be trained in a supervised fashion using the data set collected from demonstration. CNNs have shown unprecedented success in tasks such image recognition [18], speech recognition [17], and aforementioned tasks such as playing video games [4] and the game of Go [5].

1.2.4 Applications to High-Speed Autonomous Driving

High-speed autonomous off-road driving is a challenging robotics problem [19, 20, 21]. To succeed in this task, a robot is required to perform both precise steering and throttle maneuvers in a physically-complex, uncertain environment by executing a series of high-frequency decisions. Compared with most previously studied autonomous driving tasks, the robot must reason about unstructured, stochastic natural environments and operate at high speed. Consequently, designing a control policy by following the traditional model-plan-then-act approach [19, 22] becomes challenging, as it is difficult to adequately characterize the robot's interaction with the environment *a priori*. Recent model predictive control (MPC) approach [20] relies on expensive and accurate Global Positioning System (GPS) and Inertial Measurement Unit (IMU) for state estimation and demands high-frequency online replanning for generating control commands. We aim to relax these requirements by designing a reflexive driving policy that uses only *low-cost, on-board* sensors (e.g. camera, wheel speed sensors) using imitation learning and deep learning (as mentioned in section 1.2.3).

1.2.5 Contributions and Outline

The main contribution of this thesis is the computational framework of *probabilistic trajectory optimization* with applications to reinforcement learning and imitation learning. The rest of the chapters are organized as follow:

- **Chapter 2: Technical Background and Related Work.** In this chapter, we provide important technical background that is necessary to understand this thesis. We will cover problem formulation of optimal control and the dynamic programming principle. Motivated by the curse of dimensionality in dynamic programming, two families of approximation techniques will be discussed. In addition, we briefly review Gaussian process regression, a Bayesian nonparametric technique for supervised learning, which is central to incorporating uncertainty into dynamics modeling for planning and control in this thesis.

- **Chapter 3: Probabilistic Differential Dynamic Programming.** This chapter describes a model-based reinforcement learning approach that combines probabilistic inference using Gaussian processes and trajectory optimization using Differential Dynamic Programming (DDP). The resulting method, Probabilistic Differential Dynamic Programming (PDDP), features the benefits from both fields. Experimental results and comparative study show that PDDP performs well in terms of computational and data efficiency.

- **Chapter 4: Path Integral Control under Uncertainty.** In this chapter we present two data-driven optimal control framework that are derived using the path integral (PI) control approach. We find iterative control laws analytically without a priori policy parameterization based on probabilistic representation of the learned dynamics model. The two proposed algorithms operates in gradient-based and sampling-based fashions, respectively. We present results showing that incorporating model uncertainty into the path integral framework is important to achieving robust performance.

- **Chapter 5: Prediction under Uncertainty using Sparse Spectrum Gaussian Processes.** This chapter introduces two analytic moment-based approaches with closed-form expressions for Sparse Spectrum Gaussian Processes (SSGPs) regression with uncertain inputs. Our methods are more general and scalable than their standard GP counterparts, and are naturally applicable to multi-step prediction or uncertainty propagation. We show that efficient algorithms for Bayesian filtering and stochastic model predictive control can use these methods, and we evaluate our algorithms with comparative analyses and both real-world and simulated experiments.

- **Chapter 6: High-speed Off-road Autonomous Driving via Deep Imitation Learning.** In this chapter we present an end-to-end imitation learning system for agile, off-road autonomous driving using only low-cost on-board sensors. By imitating an optimal controller, we train a deep neural network control policy to map raw, high-dimensional observations to continuous steering and throttle commands. Compared with recent approaches to similar tasks, our method requires neither state estimation nor online planning to navigate the vehicle. Real-world experimental results demonstrate successful autonomous off-road driving, matching the state-of-the-art performance.

- **Chapter 7: Conclusions and Future Work.** This chapter concludes this thesis by summarizing the contributions and outlining possible research directions for future work.

CHAPTER 2
TECHNICAL BACKGROUND

In this chapter, we provide a brief introduction to the important technical background which is the foundation of this work. In section 2.1, we describe the optimal control and reinforcement learning (RL) problem formulation, the optimality principle, and two families of techniques for solving optimal control problems. These techniques use different approximation schemes to bypass the *curse of dimensionality*. In section 2.2, we briefly review Bayesian linear regression which leads to the introduction of Gaussian process regression, a key method used in this thesis for learning system dynamics.

2.1 Optimal Control and Reinforcement Learning

Optimal control is a general mathematical framework that deals with choosing actions to optimize a performance criterion, subject to a dynamics constraint. Although derived from different communities, the difference between optimal control and RL is subtle. It is usually assumed that knowledge of the system dynamics is given in optimal control, but in RL the dynamics are unknown. Next we formulate the optimal control/RL problem mathematically.

2.1.1 Problem Formulation

We consider a continuous-time, nonlinear dynamical system described by the following stochastic differential equation

$$\mathrm{d}\mathbf{x} = f(\mathbf{x},\mathbf{u})\mathrm{d}t + \mathbf{C}\mathrm{d}\omega, \quad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state, $\mathbf{u} \in \mathbb{R}^m$ is the control and $\omega \in \mathbb{R}^p$ is standard Brownian motion noise. $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and $\mathbf{C} \in \mathbb{R}^{n \times p}$ are the transition dynamics function and diffusion coefficient. We consider a finite-horizon control problem which is defined as finding a control policy π that minimize

$$J(\mathbf{x}(t_0)) = \mathbb{E}\left[h\Big(\mathbf{x}(T)\Big) + \int_{t_0}^{T} \mathcal{L}\Big(\mathbf{x}(t), \pi(\mathbf{x}(t), t)\Big)\mathrm{d}t\right], \tag{2.2}$$

where $h : \mathbb{R}^n \to \mathbb{R}$ is the terminal cost function, $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the instantaneous cost rate. The control policy

$$\mathbf{u}(t) = \pi(\mathbf{x}(t), t)$$

is a function that maps states and time to actions. The cost $J(\mathbf{x}(t_0))$ is defined as the expectation of the total cost accumulated from t_0 to T. $\mathbb{E}$ denotes the expectation operator. Here we may assume that the states are fully observable. Next we review a fundamental framework for solving this optimal control problem.

2.1.2 Dynamic Programming

Dynamic programming is a well-known framework for solving optimal control problems. First we introduce the *value function* or *cost-to-go* which is the minimum of the accumulated future cost

$$V(\mathbf{x}(t_0)) = \min_{\mathbf{u}} \Big(J(\mathbf{x}(t_0))\Big). \tag{2.3}$$

The concept of value function is central to the Dynamic Programming framework, which was derived from one of the two fundamental optimality principles for optimal control (besides Pontryagin's maximum principle) by Richard Bellman [23], it states:

Definition 1. *[23] Bellman's principle of optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must consti-*

tute an optimal policy with regard to the state resulting from the first decision.

The other optimality principle for optimal control is Pontryagin's maximum principle [24]. It provides a necessary but not a sufficient condition for optimality in deterministic systems, and it deals with a single trajectory instead of value function or policy over the whole state space. However, it is outside the scope of this thesis so we skip the discussion on it. For the rest of our analysis, we discretize the time using the Euler scheme as $k = 1, 2, ..., H$ with time step $\Delta t = \frac{T-t_0}{H-1}$. In order to simplify notation, we use subscripts to denote time steps for time-varying variables, e.g., $\mathbf{x}_k = \mathbf{x}(t_k)$. The discretized system dynamics can be written as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}_k + \mathbf{C}\sqrt{\Delta t}\xi_k \tag{2.4}$$

where $\Delta \mathbf{x}_k = \Delta t f(\mathbf{x}_k, \mathbf{u}_k)$ and ξ_k is i.i.d $\mathcal{N}(0, \mathbf{I})$. We also define $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \Delta t f(\mathbf{x}_k, \mathbf{u}_k)$ for simplicity.

According to dynamic programming, the optimal control problem can be solved by decomposing it into a sequence of single decisions. Mathematically, it is defined by the *Bellman equation*

$$V(\mathbf{x}_k, k) = \min_{\mathbf{u}_k} \Bigg(\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) + \mathbb{E}\Big[V\big(\mathbf{x}_k + \Delta \mathbf{x}_k, k+1\big)\Big] \Bigg). \tag{2.5}$$

Obviously, in order to solve this equation, we need work backward in time. However, solving this equation suffers form the *curse of dimensionality*. More specifically, when the dimension of state and action spaces is high, the total number of states and actions is enormous after discretization. In this case, solving the Bellman equation becomes computationally intractable. Next we describe two families of techniques to approximate dynamic programming that can bypass this issue.

2.1.3 Trajectory Optimization via Local Approximations

Solving general nonlinear optimal control problems via dynamic programming is difficult. But it is known that if the dynamics model is linear and cost function is quadratic, i.e.,

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \qquad \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_k^{\mathsf{T}}\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^{\mathsf{T}}\mathbf{R}\mathbf{u}_k, \tag{2.6}$$

Then the problem can be solved exactly. For simplification of analysis, we ignore the noise term in this section. Note that the noise term will not affect the resulting control policy or value function as long as the coeffiicent $\mathbf{C}$ in (4.20) is independent of state and control. The above problem (2.6) can be solved via the Linear-Quadratic-Regulator (LQR) for deterministic systems or Linear-Quadratic-Gaussian controller (LQG) for stochastic systems [25].

For nonlinear systems, we can approximate the problem around the neighborhood of a trajectory as a linear dynamics, quadratic cost problem. And use a LQR-like scheme to solve it. Now we review a family of approaches for solving nonlinear optimal control problems based on this idea. First we define the nominal trajectory of state and control $\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k$ and variations of state $\delta\mathbf{x}_k = \mathbf{x}_k - \bar{\mathbf{x}}_k$ and control $\delta\mathbf{u}_k = \mathbf{u}_k - \bar{\mathbf{u}}_k$. Next we create a local model of the dynamics along the nominal trajectory using the 1st order Taylor expansion

$$\delta\mathbf{x}_{k+1} = \left(\mathbf{I} + \frac{\partial \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)}{\partial \mathbf{x}}\right)\delta\mathbf{x}_k + \frac{\partial \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)}{\partial \mathbf{u}}\delta\mathbf{u}_k, \tag{2.7}$$

similarly, the cost function is approximated along the nominal trajectory as a quadratic function via 2nd order Taylor expansion

$$\begin{aligned}\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) =& \mathcal{L}(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k) + (\frac{\partial \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)}{\partial \mathbf{x}})^{\mathsf{T}} \delta \mathbf{x}_k + (\frac{\partial \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)}{\partial \mathbf{u}})^{\mathsf{T}} \delta \mathbf{u}_k + \\ & \frac{1}{2} \begin{bmatrix} \delta \mathbf{x}_k \\ \delta \mathbf{u}_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \frac{\partial \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)}{\partial \mathbf{x} \partial \mathbf{x}} & \frac{\partial \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)}{\partial \mathbf{x} \partial \mathbf{u}} \\ \frac{\partial \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)}{\partial \mathbf{u} \partial \mathbf{x}} & \frac{\partial \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)}{\partial \mathbf{u} \partial \mathbf{u}} \end{bmatrix} \begin{bmatrix} \delta \mathbf{x}_k \\ \delta \mathbf{u}_k \end{bmatrix}.\end{aligned} \tag{2.8}$$

Therefore the problem is reduced to a linear-quadratic problem locally along the nominal trajectory. The approximate optimal control law can be obtained as a linear function

$$\delta \mathbf{u}_k^* = \pi^*(\mathbf{x}_k) = \mathbf{I}_k + \mathbf{L}_k \delta \mathbf{x}_k \tag{2.9}$$

where the forms of open-loop and feedback terms $\mathbf{I}_k, \mathbf{L}_k$ will be discussed in the next chapter. Plugging the optimal control policy back into the Bellman equation (2.5) leads to a quadratic approximation of the value function, which can be back-propagated through time. Updating the optimal control $\mathbf{u}_k^* = \bar{\mathbf{u}}_k + \delta \mathbf{u}_k^*$ and apply it to the nonlinear dynamics to generate a new state trajectory forward in time. This becomes the new nominal trajectory and a local model can be created. Therefore the nonlinear control problem can be solved via iterative backward-forward sweeps until convergence.

The first algorithm in this family, Differential Dynamic Programming (DDP), was developed in 1970s [26]. Note that DDP uses second-order approximation of the dynamics instead of linear approximation discussed here. Variations of DDP have been proposed within the control, robotics and machine learning communities. These variations include generalizations to stochastic systems [27, 28], extensions to model predictive control [29], and min-max and cooperative game theoretic formulations [30, 31]. We will cover more technical details of this class of algorithms in the next chapter.

2.1.4 Path Integral and Linearly Solvable Optimal Control

Another way to scalable solution to optimal control problems is via exponential transformation of the value function. In control theory, this technique was introduced in [32, 33]. In the past decade it has been explored in terms of path integral interpretations and theoretical generalizations [34, 35, 36, 37], discrete time formulations [38], and scalable reinforcement learning/control algorithms [39, 40, 41, 42, 43, 44]. The resulting stochastic optimal control frameworks are known as Path Integral (PI) control for continuous time, Kullback Leibler (KL) control for discrete time, or more generally Linearly Solvable Optimal Control [38, 45]. This class of methods usually assume the dynamics model is affine in control and control cost is quadratic, i.e.,

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k) + \mathbf{B}(\mathbf{x}_k)\Big(\mathbf{u}_k + \mathbf{C}\sqrt{\Delta t}\xi_k\Big), \qquad \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) = l(\mathbf{x}_k) + \frac{1}{2}\mathbf{u}_k^{\mathsf{T}}\mathbf{R}\mathbf{u}_k, \tag{2.10}$$

where $F(\cdot)$ can $l(\cdot)$ can be arbitrary nonlinear functions. ξ has been defined in (2.4). In order to derive the solution, we need to introduce the Hamilton-Jacobi-Bellman (HJB) equation, which is the continuous-time counterpart of the discrete-time Bellman equation (2.5). Next we will use continuous time notation with subscript t with size Δt instead of discrete time step k. The HJB equation takes the following form (derivation is skipped) [10]:

$$-\partial_t V = \min\Bigg(l_t + \frac{1}{2}\mathbf{u}_t^{\mathsf{T}}\mathbf{R}\mathbf{u}_t + \big(F_t + \mathbf{B}_t\mathbf{u}_t\big)^{\mathsf{T}}\nabla_x V + \frac{1}{2}\mathbf{Tr}\left(\mathbf{B}_t\mathbf{\Sigma}_\varepsilon\mathbf{B}_t^{\mathsf{T}}\nabla_{xx}V\right)\Bigg), \tag{2.11}$$

where $l_t = l(\mathbf{x}_t), F_t = F(\mathbf{x}_t), \mathbf{\Sigma}_\varepsilon = \mathbf{C}\mathbf{C}^{\mathsf{T}}$, $\mathbf{B}_t = \mathbf{B}(\mathbf{x}_t)$, $\nabla_x V = \frac{\partial V(\mathbf{x},t+\Delta t)}{\partial \mathbf{x}}$, $\nabla_{xx} V = \frac{\partial V(\mathbf{x},t+\Delta t)}{\partial^2 \mathbf{x}}$ and $\partial_t V = \lim_{\Delta t \to t}\frac{V(xb,t+\Delta t) - V(\mathbf{x}_t)}{\Delta t}$. Given the fact the the control cost is quadratic, we can take derivative of the right-hand-side of the above equation with respect to $\mathbf{u}_t$ and set it to zero, we obtain the optimal control

$$\mathbf{u}_t^* = \mathbf{R}^{-1}\mathbf{B}_t^{\mathsf{T}}\nabla_x V. \tag{2.12}$$

Plug the above control law back into the HJB equation yields

$$-\partial_t V = l_t + (\nabla_x V)^{\mathsf{T}} F_t + (\nabla_x V)^{\mathsf{T}}\mathbf{B}_t\mathbf{R}^{-1}\mathbf{B}_t^{\mathsf{T}}\nabla_x V + \frac{1}{2}\,\mathbf{Tr}\left(\mathbf{B}_t\mathbf{\Sigma}_\varepsilon\mathbf{B}_t^{\mathsf{T}}\nabla_{xx}V\right). \tag{2.13}$$

Solution to the above equation is the optimal value function. However, solving such nonlinear, second-order partial differential equation (PDE) is difficult. The idea of using exponential transformation of the value function can reduce this PDE to a linear one. More precisely, if we set

$$\Psi_t = \exp(-\frac{V_t}{\lambda})$$

where λ is a scaling parameter. The above PDE becomes

$$\begin{aligned}\frac{\lambda}{\Psi_t}\partial_t\Psi_t = l_t - \frac{\lambda}{\Psi_t}(\nabla_x\Psi_t)^{\mathrm{T}}F_t - \frac{\lambda^2}{2\Psi_t^2}(\nabla_x\Psi_t)^{\mathrm{T}}\mathbf{B}_t\mathbf{R}^{-1}\mathbf{B}_t^{\mathrm{T}}\nabla_x\Psi_t \\ +\frac{\lambda}{2\Psi_t^2}\,\mathbf{Tr}((\nabla_x\Psi_t)^{\mathrm{T}}\mathbf{B}_t\mathbf{\Sigma}_\varepsilon\mathbf{B}_t^{\mathsf{T}}\nabla_x\Psi_t) - \frac{\lambda}{2\Psi_t}\,\mathbf{Tr}(\nabla_{xx}\Psi_t\mathbf{B}_t\mathbf{B}_t^{\mathsf{T}}),\end{aligned} \tag{2.14}$$

which is still a nonlinear PDE. However, the nonlinear terms cancel out under the constraint $\lambda\mathbf{R}^{-1} = \mathbf{\Sigma}_\varepsilon$ [34]. And the PDE is reduced to

$$\partial_t\Psi_t = \frac{1}{\lambda}l_t\Psi_t - (\nabla_x\Psi_t)^{\mathrm{T}}F_t - \frac{1}{2}\,\mathbf{Tr}(\nabla_{xx}\Psi_t\mathbf{B}_t\mathbf{B}_t^{\mathsf{T}}), \tag{2.15}$$

which is linear in differential operators. This PDE can be solved via the Feynman-Kac formula [46]

$$\Psi_t = \int \mathrm{p}(\tau_t|\mathbf{x}_t)\exp\big(-\frac{1}{\lambda}(\sum_{j=t}^{T} l(\mathbf{x}_j)\Delta t)\big)\Psi_T \mathrm{d}\tau_t, \tag{2.16}$$

where τ_t is the state trajectory from time t to T. The corresponding control law can be obtained as

$$\mathbf{u}_t^* = \mathbf{R}^{-1}\mathbf{B}_t^{\mathsf{T}}\nabla_x V = \mathbf{R}^{-1}\mathbf{B}_t^{\mathsf{T}}\frac{\nabla_x \Psi_t}{\Psi_t}. \tag{2.17}$$

It is worth noting that the control can be computed in a sampling-based or gradient-based fashion. More technical details will be covered in chapter 4.

2.2 Supervised Learning using Gaussian Processes

In this section, we first briefly review supervised learning and Bayesian analysis of standard linear regression model. Next we introduce Gaussian processes (GPs), a more powerful class of models for nonlinear regression problems. GP regression will be used to learn dynamics models for RL and model predictive control (MPC) in the rest of the thesis.

2.2.1 Supervised Learning and Bayesian Linear Regression

Supervised learning is one of the three major paradigms in machine learning (the other two are unsupervised learning and reinforcement learning). The task is to learn a function or predictor from a set of labeled data in order to predict unseen data. Classification and regression are two major tasks in supervised learning. As mentioned in the first chapter, one of the major challenges in robot control is to build a dynamics model in order to predict future states, given the current state and action. If we have a set of data observed from the real system, this task can be viewed as a regression problem in statistical learning, and the analysis can be performed in the context of Bayesian inference. We start by reviewing the standard linear model for regression

$$h(x) = \mathbf{x}^{\mathsf{T}} w, \tag{2.18}$$

where $\mathbf{x}$ is a n-dimensional input variable. And we collect a set of observation

$$\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1, ..., N\}, \quad \text{where} \quad y = h(\mathbf{x}) + \varepsilon, \tag{2.19}$$

where y is a scalar target variable, ε is i.i.d. and normally distributed noise $\mathcal{N}(0, \sigma_\varepsilon^2)$. The goal of linear regression is to estimate the weight w from data, and it can be performed in the context of Bayesian inference. This treatment is called the Bayesian linear regression

[14]. The zero-mean Gaussian noise assumption leads to Gaussian likelihood

$$\mathrm{p}(\mathbf{y}|\mathbf{X}, w) = \mathcal{N}(\mathbf{X}^\mathsf{T} w, \sigma_\varepsilon^2 \mathbf{I}), \tag{2.20}$$

where $\mathbf{X}$ is the design matrix and $\mathbf{y}$ is target vector collected from the data set $\mathcal{D}$. We can assume a prior distribution over the weight as

$$\mathrm{p}(w) = \mathcal{N}(0, \Sigma_w). \tag{2.21}$$

The posterior distribution over the weight can be derived from the Bayes theorem as

$$\mathrm{p}(w|\mathbf{X}, \mathbf{y}) = \frac{\mathrm{p}(\mathbf{y}|\mathbf{X}, w)\mathrm{p}(w)}{\mathrm{p}(\mathbf{y}|\mathbf{X})}. \tag{2.22}$$

The numerator of the above expression has the form of a Gaussian distribution over w because both prior and likelihood are Gaussians. More precisely

$$\mathrm{p}(w|\mathbf{X}, \mathbf{y}) = \mathcal{N}(A^{-1}\mathbf{X}\mathbf{y}, \sigma_\varepsilon^2 A^{-1}), \tag{2.23}$$

where $A = \mathbf{X}\mathbf{X}^\mathsf{T} + \sigma_\varepsilon^2 \Sigma_w$. However, linear models have very limited expressiveness. In order to derive a more complex and flexible model, instead of dealing with distributions over weights, we can perform Bayesian inference in function spaces directly using Gaussian processes.

2.2.2 Gaussian Processes for Regression

Gaussian process (GP) is used to define a distribution over functions. Formally it is defined as

Definition 1. *A Gaussian process is a collection of random variables, any finite subset of which has a joint Gaussian distribution [47].*

GP can be viewed as a generalization of the Gaussian distribution to an infinite-dimensional function space. Similar to a Gaussian distribution, a Gaussian process is completely specified by a mean function and a covariance function, i.e.,

$$f(\mathbf{x}) \sim \mathcal{GP}\Big(\mathbf{m}(\mathbf{x}), \mathbf{k}(\mathbf{x}, \mathbf{x}')\Big),$$

where

$$\mathbf{m}(\mathbf{x}) = \mathbb{E}_f\Big[f(\mathbf{x})\Big], \quad \mathbf{k}(\mathbf{x}, \mathbf{x}') = \mathbb{COV}_f\Big[f(\mathbf{x}), f(\mathbf{x}')\Big], \tag{2.24}$$

where $\mathbb{E}_f$ and $\mathbb{COV}_f$ denote the expectation and covariance operators with respect to the random function f. The covariance function $\mathbf{k}(\mathbf{x}, \mathbf{x}')$ is also called a *kernel*. Without any prior knowledge of the model, we may assume a prior mean function $\mathbf{m}(\cdot) = 0$ and a Squared Exponential (SE) covariance function plus a noise covariance

$$\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{W}^{-1}(\mathbf{x}_i - \mathbf{x}_j)), \tag{2.25}$$

and $\mathbb{COV}(y_i, y_j) = \mathbf{k}(\mathbf{x}_j, \mathbf{x}_j) + \sigma_\omega^2\delta_{ij}$ where δ_{ij} is a Kronecker delta which is one iff $i = j$ and zero otherwise. $\mathbf{W} = \mathrm{diag}([\ l_1^2 \ \ ... \ \ l_n^2\ ])$. The hyperparameters $\boldsymbol{\theta}$ of the kernel consist of the signal variance σ_f^2 that controls the variations of the function values from its mean, the noise variance σ_ε^2 that determines the noise magnitude we have in the data, and the characteristic length-scales for input space $l_1, ..., l_n$ that describe the smoothness of the function. The kernel function is interpreted as a similarity measure of random variables. In contrast to parametric approaches such as linear regression that rely on assumed structures and finite number of parameters, the GP approach puts a prior on function directly, therefore it is nonparametric.

Given the same data set $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, the joint distribution of the observed output and

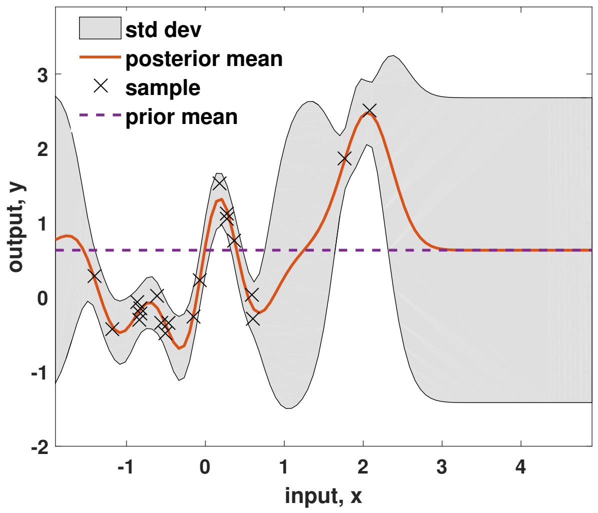


Figure 2.1: An one-dimensional example of GP inference. We use a prior mean function $\mathbf{m}(\mathbf{x}) = 0.63$ and covariance function (2.25). Black asterisks are noisy samples drawn from $f(\mathbf{x})$. Orange solid line is the GP predictive mean, and shaded area represents standard deviation for each input value $\mathbf{x}$. Note that the prediction mainly depends on the prior when the input is far away from observations.

the output corresponding to a given test input $\mathbf{x}^*$ can be written as

$$\mathrm{p}\begin{pmatrix} \mathbf{y} \\ f(\mathbf{x}^*) \end{pmatrix} = \mathcal{N}\Big(0, \begin{bmatrix} \mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_\varepsilon^2 \mathbf{I} & \mathbf{k}(\mathbf{X},\mathbf{x}^*) \\ \mathbf{k}(\mathbf{x}^*,\mathbf{X}) & \mathbf{k}(\mathbf{x}^*,\mathbf{x}^*) \end{bmatrix}\Big),$$

where $\mathbf{K}$ is a matrix with entries $\mathbf{K}_{ij} = \mathbf{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$. The posterior distribution, or predictive distribution, can be obtained by conditioning the joint distribution on the observed state transitions. Assuming independent outputs (no correlation between each output dimension), the predictive distribution is $\mathrm{p}(f(\mathbf{x}^*)|\mathbf{x}^*, \mathcal{D}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where the mean and variance are specified as

$$\begin{aligned} \boldsymbol{\mu} &= \mathbf{k}(\mathbf{x}^*,\mathbf{X})(\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_\varepsilon \mathbf{I})^{-1}\Delta\mathbf{X}, \\ \boldsymbol{\Sigma} &= \mathbf{k}(\mathbf{x}^*,\mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*,\mathbf{X})(\mathbf{K}(\mathbf{X},\mathbf{X}) + \sigma_\varepsilon \mathbf{I})^{-1}\mathbf{k}(\mathbf{X},\mathbf{x}^*). \end{aligned} \tag{2.26}$$

A toy example of GP regression is shown in fig (2.1).

Prediction under Uncertain Input

Our goal is to use GP to model the system dynamics for trajectory optimization, therefore it is necessary to make multistep predictions using the learned GP dynamics model. In this case, we need to perform GP regression iteratively when the test input is a probability distribution. The uncertainty of the input comes from the prediction in the previous step. Given a input distribution $\mathbf{x}^* \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*)$, the predictive distribution is

$$\mathrm{p}(f(\mathbf{x}^*)) = \int \mathrm{p}(f(\mathbf{x}^*)|\mathbf{x}^*)\mathrm{p}(\mathbf{x}^*)\mathrm{d}\mathbf{x}^*. \tag{2.27}$$

Generally, this predictive distribution cannot be computed analytically and the nonlinear mapping of an input Gaussian distribution leads to a non-Gaussian predictive distribution. However, the predictive distribution can be approximated by a Gaussian via moment matching, i.e., computing the exact posterior mean and variance [48, 49, 50] . This approach will be covered in the next chapter.

CHAPTER 3
PROBABILISTIC DIFFERENTIAL DYNAMIC PROGRAMMING

Motivated by the challenges of improving robustness to modeling error and solving optimal control problems in continuous state and action spaces, in this chapter we introduce a probabilistic trajectory optimization for model-based reinforcement learning.

3.1 Introduction

While model-free reinforcement learning (RL) methods have demonstrated promising results in learning control applications [51, 39, 52, 53], they typically require human expert demonstrations and a relatively large number direct interactions with the physical system. In contrast, model-based RL was developed to address the issue of sample inefficiency by learning transition dynamics models explicitly from data, which can also help to provide better generalization [54, 55]. However, model-based methods suffer from two severe issues: 1) classical value function approximation methods [56, 57] and modern global policy search methods [55] are computationally inefficient for moderate to high-dimensional problems; and 2) model errors significantly degrade the performance. To address the aforementioned issues, in this chapter we propose an RL framework that relies on Differential Dynamic Programming (DDP) and Guassian Process (GP) regression.

Originally introduced in the 70's [26], DDP solves nonlinear optimal control problems via successive local approximations of dynamics and cost functions along nominal trajectories. DDP iteratively generates locally optimal feedforward and feedback control policies along with an optimal state trajectory. Compared with global optimal control approaches, DDP shows superior computational efficiency and scalability to high-dimensional problems. In the last decade, variations of DDP have been proposed within the control, robotics and machine learning communities. These variations include generalizations to stochastic systems [27, 28], extensions to model predictive control [29] , and min-max and cooperative game theoretic formulations [30, 31].However, DDP relies on explicit dynamics models and It is sensitive to model errors. To address these issues, various data-driven approaches have been developed, such as minimax DDP using Receptive Field Weighted Regression (RFWR) [30], and DDP using expert-demonstrated trajectories [58]. One issue with these approaches is that the learned models are deterministic and do not explicitly take into account any model uncertainty. As a consequence, the aforementioned methods have either limited model expressiveness due to the fixed model structure [58], or require a large amount of data to learn a good transition model [30, 59]. In this work we perform modeling and prediction using Gaussian Processes (GPs).

Gaussian Processes (GPs) are used to define distributions over continuous functions and offer a powerful way to perform Bayesian nonparametric estimation of functions, e.g., unknown transition dynamics. Over the last decade, there has been an increasing interest in developing control/RL algorithms using GPs. For instance, the works by Rasmussen and Kuss is one of the first GP-based RL algorithm [60]; Nguyen-Tuong and Peters explored inverse dynamics learning and tracking control via local GPs and semiparametric GPs [61, 62, 63]; Deisenroth et al. proposed model-based policy search using GPs [64][50]; Hemakumara et al. used GPs for unmanned aerial vehicle controls [65] and Chowdhary et al. incorporated GPs for adaptive controls [66]. Dallaire et al. use GPs to learn the transition, observation and reward models in POMDPs[67]. These works have demonstrated the remarkable applicability of GP-based methods in robotics and autonomous learning.

In this work we integrate GPs into trajectory optimization and propose a probabilistic variant of DDP, called Probabilistic Differential Dynamic Programming (PDDP). The resulting algorithm performs probabilistic trajectory optimization that combines the benefits of DDP and GP inference. The major characteristics of PDDP are summarized as follows:

- It features data efficiency that is comparable to the state-of-the-art method.
- Computationally, it is significantly faster than other GP-based policy search methods.

- Convergence is guaranteed under certain conditions. Our theoretical analysis generalizes previous work.
- Prior model knowledge and risk-sensitive criterion can be incorporated into our proposed framework.

PDDP is related to a number of recently developed model-based RL approaches that use GPs to represent dynamics models. In particular, the PILCO framework developed by Deisenroth et al. [50] has achieved unprecedented performances in terms of data-efficiency. PILCO requires an external optimizer (e.g.,CG or BFGS) for solving non-convex optimization to obtain optimal policy parameters. In contrast, PDDP does not require a policy parameterization nor an extra optimizer[1]. Compared with other DDP-related approaches for stochastic [27, 28], unknown [59, 68] or partially known dynamics [30], PDDP explicitly takes into account model uncertainty and features the attractive characteristics of GP-based methods, i.e., data efficiency. In addition, we incorporate cost uncertainty into the optimization criterion for risk-sensitive learning. We will further discuss the similarities and differences between our method and the aforementioned methods in section 3.6.2.

3.2 Preliminaries

We consider a dynamical system described by the following differential equation

$$\mathrm{d}\mathbf{x} = f(\mathbf{x}, \mathbf{u})\mathrm{d}t + \mathbf{C}\mathrm{d}\omega, \quad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state, $\mathbf{u} \in \mathbb{R}^m$ is the control and $\omega \in \mathbb{R}^p$ is standard Brownian motion noise. $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and $\mathbf{C} \in \mathbb{R}^{n \times p}$ are the transition dynamics function and diffusion matrix. The finite-horizon control problem is defined as finding a sequence of state and a

[1]We use an off-the-shelf optimizer for kernel parameter optimization but not policy optimization.

control policy π that minimize

$$J(\mathbf{x}(t_0)) = \mathbb{E}_{\mathbf{x}}\Big[h\Big(\mathbf{x}(T)\Big) + \int_{t_0}^{T} \mathcal{L}\Big(\mathbf{x}(t), \pi(\mathbf{x}(t), t)\Big)\mathrm{d}t\Big], \tag{3.2}$$

where $h : \mathbb{R}^n \to \mathbb{R}$ is the terminal cost function, $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the instantaneous cost rate. The control policy $\mathbf{u}(t) = \pi(\mathbf{x}(t), t)$ is a function that maps states and time to actions. The cost $J(\mathbf{x}(t_0))$ is defined as the expectation of the total cost accumulated from t_0 to T. $\mathbb{E}_{\mathbf{x}}$ denotes the expectation operator with respect to $\mathbf{x}$. We assume that the states are fully observable. For the rest of our analysis, we discretize the time using the Euler scheme as $k = 1, 2, ..., H$ with time step $\Delta t = \frac{T-t_0}{H-1}$. In order to simplify notation, we use subscripts to denote time steps for time-varying variables, e.g., $\mathbf{x}_k = \mathbf{x}(t_k)$. The discretized system dynamics can be written as $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k + \mathbf{C}\sqrt{\Delta t}\xi_k$ where $\Delta\mathbf{x}_k = \Delta t f(\mathbf{x}_k, \mathbf{u}_k)$ and ξ_k is i.i.d $\mathcal{N}(0, \mathbf{I})$. We also define $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \Delta t f(\mathbf{x}_k, \mathbf{u}_k)$ for simplicity. Finding the globally optimal control policy is computationally intractable except for linear or low-dimensional systems. In this work we seek locally optimal policy which is an approximation of the globally optimal policy in the neighborhood of a nominal trajectory.

Base on the definition of Gaussian process (GP) in section 2.2 of chapter 2, we define the belief as follow

Definition 2. *A belief* $\mathbf{v}$ *of a dynamical system is defined as the probability distribution of the state* $\mathbf{x}$ *given a set of sampled control inputs and state observations.*

In this work, we assume the state is observable, and that a belief can be represented by a Gaussian distribution over the state of a dynamical system, i.e., $\mathrm{p}(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The belief can be obtained via approximate inference in probabilistic models given data. In the next section, we discuss predicting state distributions over a trajectory using GP inference.

3.3 Trajectory Prediction via GP Inference

One of the major assumptions of deterministic model-based RL methods is that the optimal policy obtained using the learned model is close to the nominal optimal policy which could be obtained if the true model is available. Due to this assumption, the performance of most deterministic model-based RL methods degrades when the learned model is far from the actual dynamics. Probabilistic models on the other hand can explicitly incorporate uncertainty when predicting the system behaviors and reduce the effect of model errors. In our approach, we use a probabilistic model learning and approximate inference scheme based on GPs. We have introduced GP in section 2.2 of chapter 2. Now we discuss how to compute predictive distributions iteratively over a trajectory using GPs.

Assume that we are given the initial state $\mathbf{x}_k$ and we have computed the predictive distribution of state $\mathbf{x}_{k+1}$, which is Gaussian distributed $\mathrm{p}(\mathbf{x}_{k+1}) = \mathcal{N}(\boldsymbol{\mu}_{k+1}, \boldsymbol{\Sigma}_{k+1})$ where the state mean and variance are

$$\boldsymbol{\mu}_{k+1} = \mathbf{x}_k + \boldsymbol{\mu}_{f_k}, \qquad \boldsymbol{\Sigma}_{k+1} = \boldsymbol{\Sigma}_{f_k}.$$

When performing two-step prediction, the input state-control pair $\tilde{\mathbf{x}}_{k+1}$ becomes uncertain. Here we define the input distribution over state-control pair at k as $\mathrm{p}(\tilde{\mathbf{x}}_k) = \mathrm{p}(\mathbf{x}_k, \mathbf{u}_k) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)$. Thus the distribution over state transition becomes

$$\mathrm{p}(\mathbf{f}(\tilde{\mathbf{x}}_k)) = \int \mathrm{p}(\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\mathbf{x}}_k)\mathrm{p}(\tilde{\mathbf{x}}_k)\mathrm{d}\tilde{\mathbf{x}}_k.$$

Generally, this predictive distribution cannot be computed analytically and the nonlinear mapping of an input Gaussian distribution leads to a non-Gaussian predictive distribution. However, the predictive distribution can be approximated by a Gaussian via computing the exact posterior mean and variance $\mathrm{p}(\mathbf{f}(\tilde{\mathbf{x}}_k)) = \mathcal{N}(\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k})$ [48, 49]. Thus the state distribution at $k+1$ is also a Gaussian $\mathcal{N}(\boldsymbol{\mu}_{k+1}, \boldsymbol{\Sigma}_{k+1})$ [50]

$$\begin{aligned} \boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{\mu}_{f_k}, \\ \boldsymbol{\Sigma}_{k+1} &= \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_{f_k} + \boldsymbol{\Sigma}_{\mathbf{f}_k,\mathbf{x}_k} + \boldsymbol{\Sigma}_{\mathbf{x}_k,\mathbf{f}_k,}, \end{aligned} \tag{3.3}$$

where $\boldsymbol{\Sigma}_{\mathbf{f}_k,\mathbf{x}_k}$ is the cross covariance between state and the corresponding state transition. The computation of $\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k}, \boldsymbol{\Sigma}_{\mathbf{f}_k,\mathbf{x}_k}$ will be discussed in the next section.

3.3.1 Approximate Inference via Moment Matching

Given an input joint distribution $\mathcal{N}(\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)$, we employ the moment matching approach [48, 49, 50] to compute the predictive distribution. This technique is also known as Assumed Density Filtering (ADF) that minimizes the Kullback−Leibler divergence between the true posterior $\mathrm{p}(\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)$ and the Gaussian approximation $\mathrm{p}(\mathbf{f}(\tilde{\mathbf{x}}_k)) \sim \mathcal{N}(\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k})$ with respect to the natural parameters of the Gaussian $\mathrm{p}(\mathbf{f}(\tilde{\mathbf{x}}_k))$, i.e.,

$$\min_{\zeta} \ \mathrm{KL}\Big(\mathrm{p}(\mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k) \parallel \mathrm{p}(\mathbf{f}(\tilde{\mathbf{x}}_k))\Big)$$

with natural parameters $\zeta = (\boldsymbol{\Sigma}_{f_k}^{-1}\boldsymbol{\mu}_{f_k}, \frac{1}{2}\boldsymbol{\Sigma}_{f_k}^{-1})$. Next we compute the moments $\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k}$ in closed-form. Applying the law of iterated expectation, the predictive mean $\boldsymbol{\mu}_{f_k}$ is evaluated as

$$\begin{aligned} \boldsymbol{\mu}_{f_k} &= \mathbb{E}_{\mathbf{f},\tilde{\mathbf{x}}_k}\Big[\mathbf{f}(\tilde{\mathbf{x}}_k)\Big|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\Big] = \mathbb{E}_{\tilde{\mathbf{x}}_k}\Big[\mathbb{E}_{\mathbf{f}}\big[\mathbf{f}(\tilde{\mathbf{x}}_k)\big|\tilde{\mathbf{x}}_k\big]\Big|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\Big] \\ &= \int \mathbb{E}_{\mathbf{f}}\big[\mathbf{f}(\tilde{\mathbf{x}}_k)\big|\tilde{\mathbf{x}}_k\big]\mathcal{N}\big(\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\big)\mathrm{d}\tilde{\mathbf{x}}_k. \end{aligned}$$

To make of analysis concise we omit the conditioning on the training pairs $\tilde{\mathrm{X}}, \Delta\mathrm{X}$ and hyperparameters $\boldsymbol{\theta}$. The predictive distributions are explicitly conditioned on the input arguments, i.e., deterministic $\tilde{\mathrm{x}}_k$ or uncertain $\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k$. Next we compute the predictive covariance matrix

$$\boldsymbol{\Sigma}_{f_k} = \begin{bmatrix} \mathbb{VAR}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\mathbf{f}_1(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] & \dots & \mathbb{COV}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\mathbf{f}_1(\tilde{\mathbf{x}}_k), \mathbf{f}_n(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] \\ \vdots & \ddots & \vdots \\ \mathbb{COV}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\mathbf{f}_n(\tilde{\mathbf{x}}_k), \mathbf{f}_1(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] & \dots & \mathbb{VAR}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\mathbf{f}_n(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k] \end{bmatrix},$$

where the variance term on the diagonal for output dimension i is obtained using the law of total variance

$$\begin{aligned} \mathbb{VAR}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\mathbf{f}_i(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k,\tilde{\boldsymbol{\Sigma}}_k] &= \mathbb{E}_{\tilde{\mathbf{x}}_k}\Big[\mathbb{VAR}_{\mathbf{f}}\big[\mathbf{f}_i(\tilde{\mathbf{x}}_k)\big|\tilde{\mathbf{x}}_k\big]\Big|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\Big] \\ &\quad + \mathbb{E}_{\tilde{\mathbf{x}}_k}\Big[\mathbb{E}_{\mathbf{f}}\big[\mathbf{f}_i(\tilde{\mathbf{x}}_k)\big|\tilde{\mathbf{x}}_k\big]^2\Big|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\Big] - \boldsymbol{\mu}_{fi_k}^2. \end{aligned}$$

and the off-diagonal covariance term for output dimension i, j is given by the expression

$$\begin{aligned} &\mathbb{COV}_{\mathbf{f},\tilde{\mathbf{x}}_k}\big[\mathbf{f}_i(\tilde{\mathbf{x}}_k), \mathbf{f}_j(\tilde{\mathbf{x}}_k)\big|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\big] = \\ &\mathbb{E}_{\tilde{\mathbf{x}}_k}\Big[\mathbb{E}_{\mathbf{f}}[\mathbf{f}_i(\tilde{\mathbf{x}}_k)\big|\tilde{\mathbf{x}}_k]\mathbb{E}_{\mathbf{f}}[\mathbf{f}_j(\tilde{\mathbf{x}}_k)\big|\tilde{\mathbf{x}}_k]\Big|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\Big] - \boldsymbol{\mu}_{fi_k}\boldsymbol{\mu}_{fj_k}. \end{aligned}$$

The input-output cross-covariance is formulated as

$$\begin{aligned} \boldsymbol{\Sigma}_{\mathbf{f}_k,\tilde{\mathbf{x}}_k} &= \mathbb{COV}_{\mathbf{f},\tilde{\mathbf{x}}_k}\big[\tilde{\mathbf{x}}_k, \mathbf{f}(\tilde{\mathbf{x}}_k)\big|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\big] \\ &= \mathbb{E}_{\tilde{\mathbf{x}}_k}\Big[\tilde{\mathbf{x}}_k\mathbb{E}_{\mathbf{f}}\big[\mathbf{f}(\tilde{\mathbf{x}}_k)\big|\tilde{\mathbf{x}}_k\big]^{\mathsf{T}}\Big|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\Big] - \tilde{\boldsymbol{\mu}}_k\boldsymbol{\mu}_{f_k}^{\mathsf{T}}. \end{aligned} \tag{3.4}$$

$\boldsymbol{\Sigma}_{\mathbf{f}_k,\mathbf{x}_k} = \mathbb{COV}_{\mathbf{f},\tilde{\mathbf{x}}_k}[\mathbf{x}_k, \mathbf{f}(\tilde{\mathbf{x}}_k)|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k]$ can be easily obtained as a sub-matrix of the above matrix. See [69, 50] for detailed derivations of the moment matching approach. This technique has been used to perform prediction under uncertain inputs in GP-based estimation [70] and control [44] tasks.

Despite providing the analytic expressions of the moments, the moment matching approach does not provide an explicit error between our Gaussian approximation and the true posterior distribution. An explicit error bound may be achieved by using numerical approximation methods. For instance, [71] uses numerical quadrature, and the error bound grows exponentially with the predictive horizon. However, such methods are known to suffer from the *curse of dimensionality*. Hence, the analytic moment based approach presented above is more suitable for the trajectory optimization framework developed in this work.

3.3.2 Hyperparameter Optimization

The hyper-parameters $\boldsymbol{\theta}$ are learned by maximizing the log-marginal likelihood of the training outputs given the inputs

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left\{ \log\left(\mathrm{p}\Big(\Delta\mathbf{X}|\tilde{\mathbf{X}}, \boldsymbol{\theta}\Big)\right)\right\}. \tag{3.5}$$

where

$$\begin{aligned}\log\left(\mathrm{p}\Big(\mathrm{d}\mathbf{X}|\tilde{\mathbf{X}}, \Theta\Big)\right) =& -\frac{1}{2}\mathrm{d}\mathbf{X}^{\mathrm{T}}\Big(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_n^2\mathbf{I}\Big)^{-1}\mathrm{d}\mathbf{X} \\ & -\frac{1}{2}\log\Big|\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) + \sigma_n^2\mathbf{I}\Big| - \frac{H+1}{2}\log 2\pi.\end{aligned}$$

This optimization problem is solved using off-the-shelf optimizers [47].

3.3.3 Incorporating Prior Model Knowledge

As a nonparametric approach, GP features flexible representation of the transition dynamics. However, model-based RL methods rely on training data collected from trajectory rollouts to make predictions. Therefore the learned models do not generalize very well to unexplored regions of the state-action spaces. In this section, we consider the case when a parametric structure of the dynamics model, or basis function is known, but the parameters are unknown. Incorporating explicit prior for GP regression has been introduced and discussed in [72, 47]. Over the last decade, studies show semiparametric models outperform purely parametric or nonparametric models for inverse and forward dynamics model identification of robotic systems [62, 73, 74]. However, these related work considered a special case when the dynamics are linear functions of the model parameters. In this work we consider a general case when the dynamics can be nonlinear in the parameters. First we define the basis function $\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})$ therefore the dynamics model can be described in a semiparametric form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \underbrace{\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})}_{\textbf{parametric}} + \underbrace{\mathbf{f}_e(\tilde{\mathbf{x}}_k)}_{\textbf{nonparametric}} \tag{3.6}$$

where $\mathbf{f}_e(\tilde{\mathbf{x}}_k) = \mathbf{f}(\tilde{\mathbf{x}}_k) - \boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})$ is the GP error (unmodeled) dynamics. This term represents the discrepancy between the parametric model and data sampled from the true model. We have discussed GP model learning and inference in chapter 2 and section 3.3.1,3.3.2 in this chapter. Given a basis function and training samples, we will learn the unknown model parameters by minimizing the loss function $l(\boldsymbol{\Theta}) = \frac{1}{2}\sum_{j=1}^{N}\Big(\Delta\mathbf{x}_j - \boldsymbol{\phi}(\tilde{\mathbf{x}}_j, \boldsymbol{\Theta})\Big)^2$ which is the accumulated error between the parametric model prediction and corresponding training data.

Parameter estimation via cross-entropy optimization

The Cross-Entropy (CE) approach [75] is generally used for estimating rare event probabilities via importance sampling. The key idea is to treat the optimization problem as a estimation of rare-event probabilities, which correspond to finding parameters $\boldsymbol{\Theta}$ that happens to be close to the optimal parameters $\boldsymbol{\Theta}^* = \arg\min l(\boldsymbol{\Theta})$. In contrast to gradient-based optimization methods, CE is a gradient-free method based on random sampling. The CE method performs optimization with the following basic steps: *random sampling* – draw K samples of $\boldsymbol{\Theta}$ from a Gaussian distribution. *Loss evaluation* – compute loss $l(\boldsymbol{\Theta}_i)$ for each sample i. *Sort* – sort $\boldsymbol{\Theta}_i$ by loss function values in ascending order. *Update* – Use K_l samples correspond to low loss (in the sorted list) to update mean and variance of the distribution. Then go back to draw samples from this updated distribution. The algorithm terminates when the variance of the distribution becomes very small. The returned solution is the mean of final sampling distribution. The CE algorithm is summarized in algorithm 1.

Algorithm 1 Cross-Entropy method for parameter learning

```
1: Initialization: Choose a multi-variate Gaussian distribution $\mathcal{N}(\mathbf{\Theta}_{\text{samp}}, \mathbf{\Sigma}_{\text{samp}})$
2: Sampling: Draw K samples of $\mathbf{\Theta}$ from $\mathcal{N}(\mathbf{\Theta}_{\text{samp}}, \mathbf{\Sigma}_{\text{samp}})$
3: Loss: Evaluate loss $l(\mathbf{\Theta}_i)$ for each sample $i$
4: Sort: Sort $\mathbf{\Theta}_{1,\ldots,K}$ w.r.t. $l(\mathbf{\Theta}_{1,\ldots,K})$ in ascending order
5: Update: Re-compute mean and variance of the sampling distribution $\mathbf{\Theta}_{\text{samp}} = \sum_{i=1}^{K_l} \frac{1}{K_l}\mathbf{\Theta}_i$ and $\mathbf{\Sigma}_{\text{samp}} = \sum_{i=1}^{K_l} \frac{1}{K_l}\left(\mathbf{\Theta}_i - \mathbf{\Theta}_{\text{samp}}\right)\left(\mathbf{\Theta}_i - \mathbf{\Theta}_{\text{samp}}\right)^{\mathsf{T}}$
6: Iterate: Return to step 2 until convergence
7: Return: Optimal estimate of parameter $\mathbf{\Theta}^* = \mathbf{\Theta}_{\text{samp}}$
```

Approximate inference via linearization

In order to compute the predictive distribution over $\mathbf{x}_{k+1}$ under uncertain input $\mathrm{p}(\tilde{\mathbf{x}}_k) = (\tilde{\boldsymbol{\mu}}_k, \tilde{\mathbf{\Sigma}}_k)$, we perform the first-order Taylor expansion of the expectation $\mathbb{E}[\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \mathbf{\Theta})|\tilde{\mathbf{x}}_k]$ around a reference point. Here we consider the mean of state-action pair $\tilde{\boldsymbol{\mu}}_k$

$$\begin{aligned}\mathbb{E}_{\tilde{\mathbf{x}}_k}[\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \mathbf{\Theta})|\tilde{\mathbf{x}}_k] &= \mathbb{E}_{\tilde{\mathbf{x}}_k}[\boldsymbol{\phi}(\tilde{\boldsymbol{\mu}}_k, \mathbf{\Theta})|\tilde{\boldsymbol{\mu}}_k] \\ &\quad + \underbrace{\frac{\partial \mathbb{E}_{\tilde{\mathbf{x}}_k}[\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \mathbf{\Theta})|\tilde{\mathbf{x}}_k]}{\partial \tilde{\mathbf{x}}_k}\Bigg|_{\tilde{\mathbf{x}}_k=\tilde{\boldsymbol{\mu}}_k}^{\mathsf{T}}}_{\mathbf{\Phi}_k} \left(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k\right).\end{aligned}$$

The predictive mean is obtained by evaluating the function at input mean $\tilde{\boldsymbol{\mu}}_k$. More precisely

$$\boldsymbol{\mu}_{\phi_k} \approx \boldsymbol{\phi}(\tilde{\boldsymbol{\mu}}_k, \mathbf{\Theta}). \tag{3.7}$$

Based on the linearized model, the predictive covariance is evaluated as

$$\begin{aligned}\boldsymbol{\Sigma}_{\phi_k} &= \mathbb{VAR}_{\tilde{\mathbf{x}}_k}\left[\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})\middle|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\right] \\ &\approx \mathbb{VAR}_{\tilde{\mathbf{x}}_k}\left[\boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k\middle|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\right] + \boldsymbol{\Sigma}_w \\ &= \mathbb{E}_{\tilde{\mathbf{x}}_k}\left[(\boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k - \boldsymbol{\Phi}_k\tilde{\boldsymbol{\mu}}_k)(\boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k - \boldsymbol{\Phi}_k\tilde{\boldsymbol{\mu}}_k)^{\mathsf{T}}\right] + \boldsymbol{\Sigma}_w \\ &= \boldsymbol{\Phi}_k\mathbb{E}_{\tilde{\mathbf{x}}_k}\left[(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)^{\mathsf{T}}\right]\boldsymbol{\Phi}_k^{\mathsf{T}} + \boldsymbol{\Sigma}_w \\ &= \boldsymbol{\Phi}_k\tilde{\boldsymbol{\Sigma}}_k\boldsymbol{\Phi}_k^{\mathsf{T}} + \boldsymbol{\Sigma}_w,\end{aligned} \tag{3.8}$$

where $\boldsymbol{\Sigma}_w$ is a diagonal matrix with entries being the noise variances σ_w^2. The cross-covariance between $\mathbf{x}_k$ and $\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})$ is computed as

$$\begin{aligned}\boldsymbol{\Sigma}_{\tilde{\mathbf{x}}_k,\phi_k} &= \mathbb{COV}_{\tilde{\mathbf{x}}_k}\left[\tilde{\mathbf{x}}_k, \boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})\middle|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\right] \\ &\approx \mathbb{COV}_{\tilde{\mathbf{x}}_k}\left[\tilde{\mathbf{x}}_k, \boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k\middle|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\right] \\ &= \mathbb{E}_{\tilde{\mathbf{x}}_k}\left[(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)(\boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k - \boldsymbol{\Phi}_k\tilde{\boldsymbol{\mu}}_k)^{\mathsf{T}}\right] \\ &= \mathbb{E}_{\tilde{\mathbf{x}}_k}\left[(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)(\tilde{\mathbf{x}}_k - \tilde{\boldsymbol{\mu}}_k)^{\mathsf{T}}\right]\boldsymbol{\Phi}_k^{\mathsf{T}} \\ &= \tilde{\boldsymbol{\Sigma}}_k\boldsymbol{\Phi}_k^{\mathsf{T}},\end{aligned} \tag{3.9}$$

The cross-covariance $\boldsymbol{\Sigma}_{\mathbf{x}_k,\phi_k} = \mathbb{COV}_{\tilde{\mathbf{x}}_k}\left[\mathbf{x}_k, \boldsymbol{\Phi}_k\tilde{\mathbf{x}}_k\middle|\tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k\right]$ is obtained as a sub-matrix. The results in (3.7),(3.8) and (3.9) are used to perform long-term prediction, i.e.,

$$\begin{aligned}\boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{\mu}_{\phi_k} = \boldsymbol{\mu}_k + \boldsymbol{\phi}(\tilde{\boldsymbol{\mu}}_k, \boldsymbol{\Theta}), \\ \boldsymbol{\Sigma}_{k+1} &= \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_{\phi_k} + \boldsymbol{\Sigma}_{\mathbf{x}_k,\phi_k} + \boldsymbol{\Sigma}_{\phi_k,\mathbf{x}_k} \\ &= \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_w + \boldsymbol{\Phi}_k\tilde{\boldsymbol{\Sigma}}_k\boldsymbol{\Phi}_k^{\mathsf{T}} + \boldsymbol{\Phi}_k\tilde{\boldsymbol{\Sigma}}_k + \tilde{\boldsymbol{\Sigma}}_k\boldsymbol{\Phi}_k^{\mathsf{T}}.\end{aligned} \tag{3.10}$$

Different from the exact moment matching used for GP inference, the approach presented here is an approximation of the moments. In general, exact moments can not be computed analytically because the basis function $\boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta})$ from a physical model is unlikely to have the same form as in GP posterior mean or variance. The partially known parametric model is very useful when predicting in the region with sparse (or no) training samples. However, the analysis above does not take into account the estimation error of the parametric model, therefore it could still give problematic long-term predictions.

Semiparametric model

Now we combine the benefits of both parametric and nonparametric models. The semiparametric model uses the parametric part to generalize to regions in state-action space that are far from the training data, where the nonparametric part is used to model the error dynamics, i.e., difference between the true dynamics and parametric part. For instance the posterior GP mean becomes

$$\begin{aligned}\mathbb{E}_{\mathbf{f}}\big[\mathbf{f}(\tilde{\mathbf{x}}_k)\big|\tilde{\mathbf{x}}_k\big] =& \boldsymbol{\phi}(\tilde{\mathbf{x}}_k, \boldsymbol{\Theta}) + \mathbf{k}(\tilde{\mathbf{x}}_k, \tilde{\mathbf{X}})(\mathbf{K}(\tilde{\mathbf{X}}, \tilde{\mathbf{X}}) \\ &+ \sigma_w \mathbf{I})^{-1}\Big(\Delta\mathbf{X} - \boldsymbol{\Phi}(\tilde{\mathbf{X}}, \boldsymbol{\Theta})\Big),\end{aligned} \tag{3.11}$$

where $\Delta\mathbf{X} - \boldsymbol{\Phi}(\tilde{\mathbf{X}}, \boldsymbol{\Theta}) = \{\Delta\mathbf{x}_1^s - \boldsymbol{\phi}(\tilde{\mathbf{x}}_1^s, \boldsymbol{\Theta}), ..., \Delta\mathbf{x}_N^s - \boldsymbol{\phi}(\tilde{\mathbf{x}}_N^s, \boldsymbol{\Theta})\}$ is a collection of sampled state transition errors. It can be seen that if the query state-action pair $\tilde{\mathbf{x}}_k$ is very far away from the training samples $\tilde{\mathbf{X}}$, i.e., $\tilde{\mathbf{x}}_k - \tilde{\mathbf{x}}_i^s \to \infty,\ i = 1, .., N$, the kernel $\mathbf{k}(\mathbf{x}_k, \mathbf{x}_i^s) = \sigma_f^2 \exp(-\frac{1}{2}(\mathbf{x}_k - \mathbf{x}_i^s)^{\mathrm{T}}\mathbf{W}(\mathbf{x}_k - \mathbf{x}_i^s)) \to 0$. In this case the nonparametric part almost vanish and the parametric part dominates in making predictions. Therefore the parametric model is essential in generalizing the model to regions in state-action space that are not covered by the training samples. This feature enables accurate prediction under insufficient exploration.

Given a Gaussian predictive distribution over the error dynamics $\mathrm{p}\big(\mathbf{f}_e(\tilde{\mathbf{x}}_k)\big) = \mathcal{N}\big(\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k}\big)$, and based on the approximate inference methods introduced in chapter 2 and sections 3.3.1,3.3.3 and 3.3.3 in this chapter, the semiparametric dynamics model can be written as

$$\begin{aligned}\boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{\mu}_{\phi_k} + \boldsymbol{\mu}_{f_k} \\ \boldsymbol{\Sigma}_{k+1} &= \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_{\phi_k} + \boldsymbol{\Sigma}_{f_k} + \boldsymbol{\Sigma}_w \\ &\quad + \boldsymbol{\Sigma}_{\mathbf{f}_k,\mathbf{x}_k} + \boldsymbol{\Sigma}_{\mathbf{x}_k,\mathbf{f}_k} + \boldsymbol{\Sigma}_{\phi_k,\mathbf{x}_k} + \boldsymbol{\Sigma}_{\mathbf{x}_k,\phi_k}.\end{aligned} \tag{3.12}$$

where the terms $\boldsymbol{\mu}_{f_k}, \boldsymbol{\Sigma}_{f_k}$ in (3.12) denote predictive distribution over the transition error dynamics. Note that (3.12) is a generalized version of (4.27). If $\phi(\cdot,\cdot)$ outputs zeros, (3.12) becomes (4.27). The above semiparametric formulation relies on partial knowledge of the dynamics model, which is represented by explicit basis function of the state, actions and unknown parameters. In practice this prior knowledge may be available for mechanical systems. Studies in the last five years have shown that semiparametric models perform substantially better than purely parametric and nonparametric models [62, 73, 74]. This advantage is more significant in real world experiments than in simulations [62, 73]. This is because the GP error dynamics model also absorbs the modeling error of basis functions in real physical systems. In contrast, when performing experiments in simulations, the basis functions are assumed to be good representations of the dynamics and used to generate sample data. Overall, the proposed model learning and inference scheme takes into account uncertainty of unknown or partially known dynamics which will be used for controller design in the next section.

3.4 Probabilistic Trajectory Optimization

Now we introduce Probabilistic Differential Dynamic Programming (PDDP), a trajectory optimizer that employs a combination of probabilistic inference and Differential Dynamic Programming (DDP).

3.4.1 Belief Dynamics and Local Approximation

In DDP-related algorithms, a local model along a nominal trajectory $(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$, is created based on: i) a first or second-order approximation of the dynamics model; ii) a second-order approximation of the value function. In our proposed PDDP framework, we will create a local model along a trajectory of state distribution-control pair $(\mathrm{p}(\bar{\mathbf{x}}_k), \bar{\mathbf{u}}_k)$. In order to incorporate uncertainty explicitly in the local model, we use the Gaussian belief $\mathbf{v}_k = [\boldsymbol{\mu}_k \ \mathrm{vec}(\boldsymbol{\Sigma}_k)]^{\mathrm{T}}$ over state $\mathbf{x}_k$ where $\mathrm{vec}(\boldsymbol{\Sigma}_k)$ is the vectorization of $\boldsymbol{\Sigma}_k$. Based on eq.(4.27) or (3.12), the belief dynamics model can be written as

$$\mathbf{v}_{k+1} = \mathcal{F}(\mathbf{v}_k, \mathbf{u}_k). \tag{3.13}$$

Now we create a local model of the belief dynamics. Firstly we define the control and state variations $\delta\mathbf{v}_k = \mathbf{v}_k - \bar{\mathbf{v}}_k$ and $\delta\mathbf{u}_k = \mathbf{u}_k - \bar{\mathbf{u}}_k$. Next we perform Taylor expansion of the belief dynamics around $(\mathbf{v}_k, \mathbf{u}_k)$

$$\delta\mathbf{v}_{k+1} = \mathcal{F}_k^b \delta\mathbf{v}_k + \mathcal{F}_k^u \delta\mathbf{u}_k + O(\|\delta\mathbf{v}_k\|^2 + \|\delta\mathbf{u}_k\|^2), \tag{3.14}$$

In this work we compute the first-order expansion in (3.14) but keep the higher order terms $O(\cdot)$ for theoretical analysis. The Jacobian matrices $\mathcal{F}_k^b$ and $\mathcal{F}_k^u$ are specified as

$$\mathcal{F}_k^b = \begin{bmatrix} \frac{\partial \boldsymbol{\mu}_{k+1}}{\partial \boldsymbol{\mu}_k} & \frac{\partial \boldsymbol{\mu}_{k+1}}{\partial \boldsymbol{\Sigma}_k} \\ \frac{\partial \boldsymbol{\Sigma}_{k+1}}{\partial \boldsymbol{\mu}_k} & \frac{\boldsymbol{\Sigma}_{k+1}}{\partial \boldsymbol{\Sigma}_k} \end{bmatrix}, \quad \mathcal{F}_k^u = \begin{bmatrix} \frac{\partial \boldsymbol{\mu}_{k+1}}{\partial \mathbf{u}_k} \\ \frac{\partial \boldsymbol{\Sigma}_{k+1}}{\partial \mathbf{u}_k} \end{bmatrix}. \tag{3.15}$$

The partial derivatives $\frac{\partial \boldsymbol{\mu}_{k+1}}{\partial \boldsymbol{\mu}_k}, \frac{\partial \boldsymbol{\mu}_{k+1}}{\partial \boldsymbol{\Sigma}_k}, \frac{\partial \boldsymbol{\Sigma}_{k+1}}{\partial \boldsymbol{\mu}_k}, \frac{\partial \boldsymbol{\Sigma}_{k+1}}{\partial \boldsymbol{\Sigma}_k}, \frac{\partial \boldsymbol{\mu}_{k+1}}{\partial \mathbf{u}_k}, \frac{\partial \boldsymbol{\Sigma}_{k+1}}{\partial \mathbf{u}_k}$ are computed analytically. Their forms are omitted due to the space limit. For numerical implementation, the dimension of the belief can be reduced by eliminating the redundancy of $\boldsymbol{\Sigma}_k$ and the principle square root of $\boldsymbol{\Sigma}_k$ may be used for numerical robustness.

3.4.2 Optimization Criterion

Since the state is a random variable, we need to compute the expected costs and we define them as functions of the belief

$$\tilde{\mathcal{L}}(\mathbf{v}_k, \mathbf{u}_k) = \mathbb{E}_{\mathbf{x}}[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)], \;\; \tilde{h}(\mathbf{v}_k) = \mathbb{E}_{\mathbf{x}}[h(\mathbf{x}_k)].$$

For instance, in many optimal control approaches, a quadratic cost function is used

$$\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) = (\mathbf{x}_k - \mathbf{x}_k^{goal})^{\mathrm{T}}\mathbf{Q}(\mathbf{x}_k - \mathbf{x}_k^{goal}) + \mathbf{u}_k^{\mathrm{T}}\mathbf{R}\mathbf{u}_k, \tag{3.16}$$

where $\mathbf{x}_k^{goal}$ is the target state. Given the distribution $\mathrm{p}(\mathbf{x}_k) = \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, the expectation of original quadratic cost function is formulated as the belief-control cost

$$\tilde{\mathcal{L}}(\mathbf{v}_k, \mathbf{u}_k) = \mathbb{E}_{\mathbf{x}}\Big[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\Big] = \mathcal{L}(\boldsymbol{\mu}_k, \mathbf{u}_k) + \mathrm{tr}(\boldsymbol{\Sigma}_k\mathbf{Q}) \tag{3.17}$$

The cost expectation (3.17) scales linearly with the state covariance, therefore it penalize both distance between the expected state and the target state, and the uncertainty of the predictive state. In this work, we also consider a risk-sensitive cost function [76]

$$\tilde{\mathcal{L}}(\mathbf{v}_k, \mathbf{u}_k, \epsilon) = \begin{cases} \frac{1}{\epsilon}\log \mathbb{E}_{\mathbf{x}}\Big[\exp\big(\epsilon\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\big)\Big], & \epsilon \neq 0 \\ \mathbb{E}_{\mathbf{x}}\Big[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\Big], & \epsilon = 0 \end{cases} \tag{3.18}$$

where ϵ is the risk sensitivity parameter. Taking the second-order Taylor expansion of the above cost function about $\mathbb{E}[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)]$ yields [76]

$$\tilde{\mathcal{L}}(\mathbf{v}_k, \mathbf{u}_k, \epsilon) \approx \mathbb{E}_{\mathbf{x}}\big[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\big] + \frac{\epsilon}{2}\mathbb{VAR}_{\mathbf{x}}\big[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k)\big], \tag{3.19}$$

where the variance of cost is obtained as

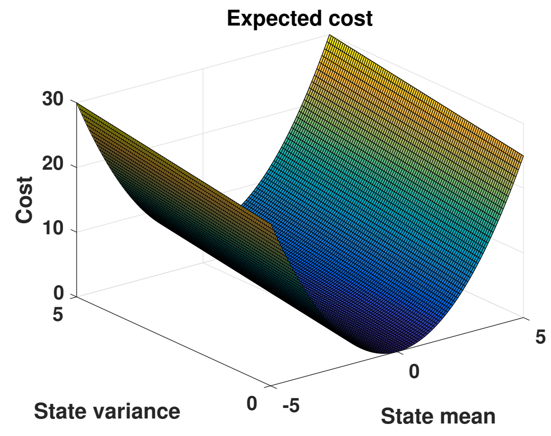
$$\begin{aligned}\mathbb{VAR}_{\mathbf{x}}\Big[\mathcal{L}(\mathbf{x}_k,\mathbf{u}_k)\Big] &= \mathrm{tr}(2\mathbf{Q}\boldsymbol{\Sigma}_k\mathbf{Q}\boldsymbol{\Sigma}_k)\\ &+ 4(\boldsymbol{\mu}_k-\mathbf{x}_k^{goal})^{\mathrm{T}}\mathbf{Q}\boldsymbol{\Sigma}_k\mathbf{Q}(\boldsymbol{\mu}_k-\mathbf{x}_k^{goal}).\end{aligned} \tag{3.20}$$

Partial derivatives of the cost w.r.t. belief and control $\frac{\partial}{\partial \mathbf{v}_k}\tilde{\mathcal{L}}(\mathbf{v}_k,\mathbf{u}_k)=\{\frac{\partial}{\partial \boldsymbol{\mu}_k}\tilde{\mathcal{L}}(\mathbf{v}_k,\mathbf{u}_k),\frac{\partial}{\partial \boldsymbol{\Sigma}_k}\tilde{\mathcal{L}}(\mathbf{v}_k,\mathbf{u}_k)\}$ and $\frac{\partial}{\partial \mathbf{u}_k}\tilde{\mathcal{L}}(\mathbf{v}_k,\mathbf{u}_k)$ can be computed analytically. Their expressions are omitted due to the space limit. The cost function (3.19) captures not only the expected cost, but also the predictive uncertainty of the cost. It can be viewed a generalization of the cost expectation used in RL and stochastic control. More precisely, when $\epsilon=0$ the cost becomes *risk-neutral*, which is equal to the expected cost (3.17). $\epsilon>0$ corresponds to *risk-averse* and $\epsilon<0$ corresponds to *risk-seeking* criterion. In the risk-averse case, the trajectory optimizer explicitly avoid regions with high uncertainty. In the risk-seeking case, trajectories with higher-variance are preferred. When no prior knowledge of the dynamics model is available, the risk-averse strategy is a good fit for the PDDP frameworks. Since it relies on local approximations of the dynamics and data sampled along one or multiple trajectories, the learned model is accurate only within the neighborhood of these trajectories. The risk-averse policies avoids wide cost distribution explicitly, therefore extensive explorations in highly uncertainty regions are avoided, and the regions close to the sampled data are preferred. This criterion is especially beneficial for the case when very little samples are available. For the rest of this chapter, the risk-sensitive learning corresponds to the risk-averse case when $\epsilon>0$.

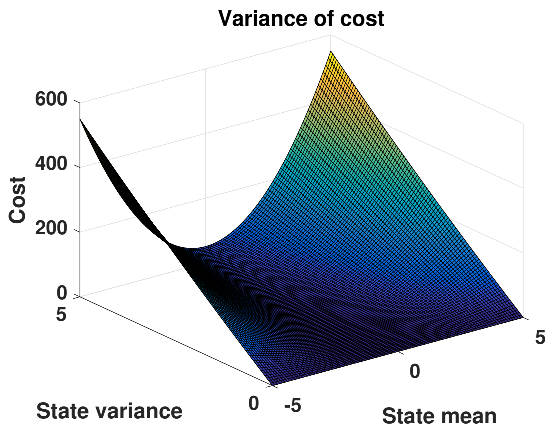
For a general non-quadratic cost function we approximate it as a quadratic function along the nominal belief and control trajectory $(\mathbf{b}, \bar{\mathbf{u}})$, i.e.,

$$\tilde{\mathcal{L}}(\mathbf{v}_k, \mathbf{u}_k) = \tilde{\mathcal{L}}_k^0 + (\tilde{\mathcal{L}}_k^b)^\mathsf{T} \delta\mathbf{v}_k + (\tilde{\mathcal{L}}_k^u)^\mathsf{T} \delta\mathbf{u}_k + \tag{3.21}$$
$$\frac{1}{2} \begin{bmatrix} \delta\mathbf{v}_k \\ \delta\mathbf{u}_k \end{bmatrix}^\mathrm{T} \begin{bmatrix} \tilde{\mathcal{L}}_k^{bb} & \tilde{\mathcal{L}}_k^{bu} \\ \tilde{\mathcal{L}}_k^{ub} & \tilde{\mathcal{L}}_k^{uu} \end{bmatrix} \begin{bmatrix} \delta\mathbf{v}_k \\ \delta\mathbf{u}_k \end{bmatrix} + O(\|\delta\mathbf{v}_k\|^2, \|\delta\mathbf{u}_k\|^2),$$

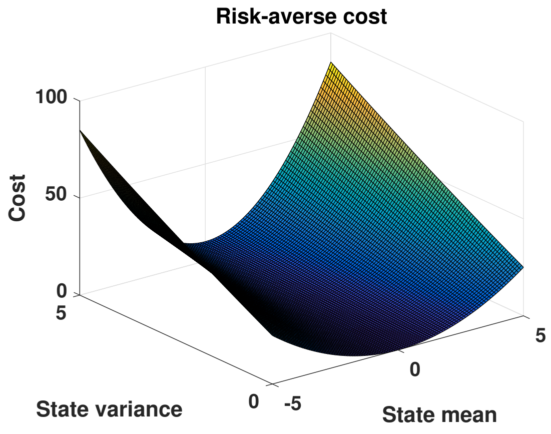
where superscripts denote partial derivatives, e.g., $\tilde{\mathcal{L}}_k^b = \nabla_b \tilde{\mathcal{L}}_k(\mathbf{v}_k, \mathbf{u}_k)$ and $\tilde{\mathcal{L}}_k^0 = \tilde{\mathcal{L}}(\mathbf{b}_k, \bar{\mathbf{u}}_k)$. We will use this superscript rule for all cost-related terms.



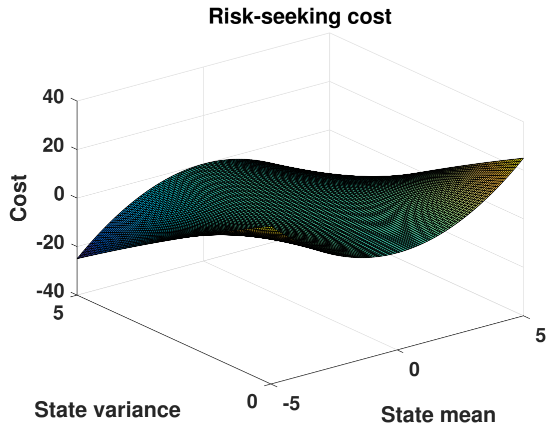
(a)



(b)



(c)



(d)

Figure 3.1: One-dimensional examples of expectation of cost (a), variance of cost (b), risk-averse (c) and risk-seeking cost fucntions (d). The target state is zero.

3.4.3 Control Policy and Value Function Approximation

According to the Dynamic Programming principle [26], the Bellman equation in discrete-time is specified as follows

$$V(\mathbf{v}_k, k) = \min_{\mathbf{u}_k} \underbrace{\tilde{\mathcal{L}}(\mathbf{v}_k, \mathbf{u}_k) + V\big(\mathcal{F}(\mathbf{v}_k, \mathbf{u}_k), k+1\big)}_{Q(\mathbf{v}_k, \mathbf{u}_k)}, \tag{3.22}$$

where $V(\mathbf{v}_k, k)$ is the value function for belief $\mathbf{v}$ at time step k. At the terminal time step $V(\mathbf{v}_H, H) = \tilde{h}(\mathbf{v}(H))$. Given the belief dynamics (3.14) and cost (3.21), our goal is to obtain a quadratic approximation of the value function along a nominal belief trajectory $\mathbf{b}_k$

$$V(\mathbf{v}_k, k) = V_k^0 + (V_k^b)^{\mathsf{T}} \delta\mathbf{v}_k + \frac{1}{2} \delta\mathbf{v}_k^{\mathsf{T}} V_k^{bb} \delta\mathbf{v}_k + O(\|\delta\mathbf{v}_k\|^3). \tag{3.23}$$

We can do so by expanding the Q-function defined in (3.22) along $(\mathbf{b}_k, \bar{\mathbf{u}}_k)$

$$\begin{aligned} Q_k(\mathbf{b}_k + \delta\mathbf{v}_k, \bar{\mathbf{u}}_k + \delta\mathbf{u}_k) &= Q_k^0 + (Q_k^b)^{\mathsf{T}} \delta\mathbf{v}_k + (Q_k^u)^{\mathsf{T}} \delta\mathbf{u}_k + \\ \frac{1}{2} \begin{bmatrix} \delta\mathbf{v}_k \\ \delta\mathbf{u}_k \end{bmatrix}^{\mathrm{T}} &\begin{bmatrix} Q_k^{bb} & Q_k^{bu} \\ Q_k^{ub} & Q_k^{uu} \end{bmatrix} \begin{bmatrix} \delta\mathbf{v}_k \\ \delta\mathbf{u}_k \end{bmatrix} + O\left(\left\| \begin{bmatrix} \delta\mathbf{v}_k \\ \delta\mathbf{u}_k \end{bmatrix} \right\|^2 \right), \end{aligned} \tag{3.24}$$

where

$$\begin{aligned} &Q_k^b = \mathcal{L}_k^b + (\mathcal{F}_k^b)^{\mathsf{T}} V_{k+1}^b, \quad Q_k^u = \mathcal{L}_k^u + (\mathcal{F}_k^u)^{\mathsf{T}} V_{k+1}^b, \\ &Q_k^{bb} = \mathcal{L}_k^{bb} + (\mathcal{F}_k^b)^{\mathrm{T}} V_{k+1}^{bb} \mathcal{F}_k^b, \quad Q_k^{ub} = \mathcal{L}_k^{ub} + (\mathcal{F}_k^u)^{\mathrm{T}} V_{k+1}^{bb} \mathcal{F}_k^b, \\ &Q_k^{uu} = \mathcal{L}_k^{uu} + (\mathcal{F}_k^u)^{\mathrm{T}} V_{k+1}^{bb} \mathcal{F}_k^u. \end{aligned} \tag{3.25}$$

In order to find the optimal control policy, we compute the local variations in control $\delta\hat{\mathbf{u}}_k$ that minimize the quadratic approximation of the Q-function in (3.24)

$$\begin{aligned} \delta\hat{\mathbf{u}}_k &= \arg\min_{\delta\mathbf{u}_k}\Big[Q_k(\mathbf{b}_k+\delta\mathbf{v}_k,\bar{\mathbf{u}}_k+\delta\mathbf{u}_k)\Big] \\ &= \underbrace{-(Q_k^{uu})^{-1}Q_k^{u}}_{\mathbf{l}_k}\underbrace{-(Q_k^{uu})^{-1}Q_k^{ub}}_{\mathbf{L}_k}\delta\mathbf{v}_k \end{aligned} \tag{3.26}$$

The optimal control can be found as $\hat{\mathbf{u}}_k = \bar{\mathbf{u}}_k + \delta\hat{\mathbf{u}}_k$. The control policy is a linear function of the belief $\mathbf{v}_k$, therefore the controller is deterministic. To ensure convergence we apply line search by adding a parameter $0 < \varepsilon \leq 1$ to the feedforward gain. More precisely

$$\hat{\mathbf{u}}_k = \bar{\mathbf{u}}_k + \varepsilon\mathbf{l}_k + \mathbf{L}_k\delta\mathbf{v}_k. \tag{3.27}$$

We start by setting $\varepsilon = 1$ and reduce it if the trajectory cost under the new control is higher than the cost of the nominal trajectory. If the new cost is lower than the nominal cost we accept the control $\hat{\mathbf{u}}_k$ and reset $\varepsilon = 1$. We will show the global convergence of the optimal control and value function in section 3.5. Plugging the optimal control (3.27) into the approximated Q-function (3.24) results in the following backward propagation of parameters for value function (3.23)

$$\begin{aligned} &V_k^0 = \mathcal{L}_k^0 + V_{k+1}^0 - (\frac{1}{2}\varepsilon^2 - \varepsilon)(Q_k^u)^{\mathsf{T}}\mathbf{l}_k \\ &V_k^b = Q_k^b + Q_k^{bu}\mathbf{l}_k, \qquad V_k^{bb} = Q_k^{bb} + Q_k^{bu}\mathbf{L}_k. \end{aligned} \tag{3.28}$$

After the backward pass, we apply the policy (3.27) to generate a new control and belief trajectory by propagating the belief dynamics 4.28 forward in time. The belief propagation is performed using approximate inference methods, i.e., exact moment matching (3.3.1) and linear approximation (3.3.3).

3.4.4 Control constraint

Control constrains can be taken into account in different fashions, it has been shown that using naive clamping and squashing functions performs unfavorably compared to directly incorporating the constraints while minimizing the Q-function [77]. In this work we take into account the control constraints by solving a quadratic programming (QP) problem subject to a box constraint

$$\begin{aligned} &\min_{\delta \mathbf{u}_k} \; Q_k(\mathbf{v}_k + \delta\mathbf{v}_k, \mathbf{u}_k + \delta\mathbf{u}_k) \\ &\text{sub. to } \; \mathbf{u}_{\min} \leq \mathbf{u}_k + \delta\mathbf{u}_k \leq \mathbf{u}_{\max} \end{aligned} \tag{3.29}$$

where $\mathbf{u}_{\min}$ and $\mathbf{u}_{\max}$ correspond to the lower and upper bounds of the controller. The QP problem (3.29) can be solved efficiently due to the fact that at each time step the scale of the QP problem is relatively small. And warm-start can be used to further speed up the optimization in the backward pass. Solving (3.29) directly is not feasible since $\delta\mathbf{v}$ is not known in the backward sweep. The optimum consists of feedforward and feedback parts, i.e., $\mathbf{I}_k + \mathbf{L}_k\delta\mathbf{v}_k = \arg\min_{\delta\mathbf{u}_k}\left[Q_k(\mathbf{v}_k + \delta\mathbf{v}_k, \mathbf{u}_k + \delta\mathbf{u}_k)\right]$, here we adopt the strategy in [77] using the Projected-Newton algorithm [78]. The feedforward gain is computed by solving the QP problem

$$\begin{aligned} &\mathbf{I}_k = \arg\min_{\delta\mathbf{u}_k}\left[\delta\mathbf{u}_k^{\mathsf{T}} Q_k^{uu}\delta\mathbf{u}_k + Q_k^b\delta\mathbf{u}_k\right] \\ &\text{sub. to } \; \mathbf{u}_{\min} \leq \mathbf{u}_k + \delta\mathbf{u}_k \leq \mathbf{u}_{\max} \end{aligned} \tag{3.30}$$

The algorithm gives the decomposition of $Q_k^{uu} = \begin{bmatrix} Q_{k,ff}^{uu} & Q_{k,fc}^{uu} \\ Q_{k,cf}^{uu} & Q_{k,cc}^{uu} \end{bmatrix}$ where the indices f, c correspond to clamped (when $\mathbf{u}_k = \mathbf{u}_{\min}$ or $\mathbf{u}_{\max}$) or free ($\mathbf{u}_{\min} < \mathbf{u}_k < \mathbf{u}_{\max}$) parts, respectively. The feedback gain associated to the free part is obtained by $\mathbf{L}_k = -Q_{k,ff}^{uu} Q_k^{ub}$. The rows of $\mathbf{L}_k$ corresponding to clamped controls are set to be zero. An example of the constrained and unconstrained controllers for the quadrotor tasks is shown in fig.(3.8).

3.4.5 Summary of algorithm

The PDDP framework for RL can be summarized in **Algorithm** 2. From a high-level view, the main loop of this algorithm consists of 3 basic components. 1) Model learning (Step 3): Learning a probabilistic GP dynamics model using data sampled from interactions with physical systems. If prior knowledge (basis function) of the system dynamics is given, model parameters may be learned as discussed in section 3.3.3; 2) Probabilistic trajectory optimization (Step 5-7): this iterative scheme has 3 steps at each iteration. First (step 5), we compute the linear approximation (3.14) of the belief dynamics(4.28) and quadratic approximation of the cost (3.21) along a nominal trajectory. Second (step 6), we compute a local optimal control policy (3.27) by backward-propagation of the value function (3.28). Control constraints can be incorporated in the backward propagation, see section 3.4.4. In addition we employ a line search strategy to ensure convergence, see section 3.4.3 for details. Third (step 7), we update and apply the control to obtain a new belief trajectory using approximate inference methods introduced in sections 3.3.1,3.3.3. This trajectory is used as the nominal for the next iteration. Regarding the termination condition (Step 8), we consider 3 stopping criteria: i) maximum iteration number is reached, ii) policy improvement is small enough, and iii) predictive variance exceeds a threshold. The optimization techniques stop the iteration process when at least one of the corresponding termination criteria is satisfied. 3) Interaction (Step 10), in order to collect new state and control samples we apply the optimized trajectory and control policy to the physical systems and generate a trajectory rollout. Additional trajectories can be generated using variations of the optimized trajectory.

3.5 Theoretical Analysis

In this section, we provide theoretical analysis and show that PDDP is globally convergent. The convergence property of the classical DDP has been discussed in [79] for scalar sys-

Algorithm 2 : Probabilistic Differential Dynamic Programming for reinforcement learning

```
1: Initialization: Apply random or pre-specified control inputs to the physical system
   (4.20). Collect data.
2: repeat                                        ▷ Main reinforcement learning loop
3:     Model learning: Learn GP hyper-parameters $\boldsymbol{\theta}$ by evidence maximization (3.5)
       given sample data. Optimize dynamics model parameters if necessary (see section
       3.3.3).
4:     repeat                                    ▷ Probabilistic trajectory optimization loop
5:         Local approximation: Obtain linear approximation of the belief dynamics
           (3.14) and quadratic approximation of the cost (3.21) along a nominal trajectory
           $(\bar{\mathbf{v}}_k, \bar{\mathbf{u}}_k)$. See section 3.4.1.
6:         Backward sweep: Compute the approximation of the value function (3.28)
           and obtain optimal policy for control correction $\delta\hat{\mathbf{u}}_k$ (3.26). See section 3.4.3.
7:         Forward sweep: Update control $\bar{\mathbf{u}}_k$ and perform approximate inference to ob-
           tain a new nominal trajectory $(\bar{\mathbf{v}}_k, \bar{\mathbf{u}}_k)$. See sections 3.3.1,3.3.3.
8:     until Termination condition is satisfied
9:     return Optimal trajectory $(\bar{\mathbf{v}}_k, \bar{\mathbf{u}}_k)$ and control policy
10:    Interaction: Apply the optimized control policy $\mathbf{L}_k$ to the system (4.20) along the
       optimized trajectory $\bar{\mathbf{u}}_k$ and record data. Additional rollouts can be generated using
       variations of the learned controller.
11: until Task learned
12: Return: Optimal trajectory and control policy
```

tems. However, the analysis is based on the a-priori assumption that the algorithm would converge. [80] addresses the global convergence of DDP. But it only applies when the running cost $\mathcal{L}(\cdot,\cdot) = 0$. Here we provide a more general proof that guarantees convergence. First, we define the following vectors

$$\Delta\mathbf{U} = [\delta\mathbf{u}_1^{\mathsf{T}}, ..., \delta\mathbf{u}_{H-1}^{\mathsf{T}}]^{\mathsf{T}}, \quad \mathbf{U} = [\mathbf{u}_1^{\mathsf{T}}, ..., \mathbf{u}_{H-1}^{\mathsf{T}}]^{\mathsf{T}} \in \mathbb{R}^{m\times(H-1)}. \tag{3.31}$$

In our analysis we make the following assumptions

Assumption 1. *Q_k^{uu} is positive definite for $k = 1, .., H-1$.*

Assumption 2. *The initial state $\mathbf{x}_1$ is known.*

Remark 1. *The condition in assumption 1 can be easily ensured by applying the Levenberg-Marquardt-related trick as in [27]. The details are omitted due to page limitation. Assumption 2 is common for most episodic RL methods.*

Lemma 1. *For the total cost J defined in (3.2), belief dynamics defined in section 3.4.1 and optimization criterion defined in section 3.4.2, the following is true*

$$\nabla_{\mathbf{u}_k} J = (\mathcal{F}_k^u)^\mathsf{T} \eta_{k+1} + \mathcal{L}_k^u \tag{3.32}$$

for $k = 1, ..., H-1$, where $\eta_k = (\mathcal{F}_k^b)^\mathsf{T} \eta_{k+1} + \mathcal{L}_k^b$ and $\eta_H = \tilde{h}^b$.

Proof. For $k = 1, ..., H-1$ we have

$$\begin{aligned}
\nabla_{\mathbf{u}_k} J &= \nabla_{\mathbf{u}_k} \mathbb{E}_{\mathbf{x}} \Big[\sum_{\tau=k}^{H-1} \mathcal{L}(\mathbf{x}_\tau, \mathbf{u}_\tau) + h(\mathbf{x}_H) \Big] \\
&= \nabla_{\mathbf{u}_k} \Big[\sum_{\tau=k}^{H-1} \tilde{\mathcal{L}}(\mathbf{v}_\tau, \mathbf{u}_\tau) + \tilde{h}(\mathbf{v}_H) \Big] \\
&= \frac{\partial \tilde{\mathcal{L}}(\mathbf{v}_k, \mathbf{u}_k)}{\partial \mathbf{u}_k} + \frac{\partial \tilde{\mathcal{L}}(\mathbf{v}_{k+1}, \mathbf{u}_{k+1})}{\partial \mathbf{u}_k} + \cdots + \frac{\partial \tilde{\mathcal{L}}(\mathbf{v}_{H-1}, \mathbf{u}_{H-1})}{\partial \mathbf{u}_k} + \frac{\partial \tilde{h}(\mathbf{v}_H)}{\partial \mathbf{u}_k} \\
&= \mathcal{L}_k^u + ((\mathcal{L}_{k+1}^b)^\mathsf{T} \mathcal{F}_k^u)^\mathsf{T} + ((\mathcal{L}_{k+2}^b)^\mathsf{T} \mathcal{F}_{k+1}^b \mathcal{F}_k^u)^\mathsf{T} + \cdots \\
&\quad + ((\mathcal{L}_{H-1}^b)^\mathsf{T} \mathcal{F}_{H-2}^b \cdots \mathcal{F}_{k+1}^b \mathcal{F}_k^u)^\mathsf{T} + ((\tilde{h}^b)^\mathsf{T} \mathcal{F}_{H-1}^b \cdots \mathcal{F}_{k+1}^b \mathcal{F}_k^u)^\mathsf{T} \\
&= \mathcal{L}_k^u + (\mathcal{F}_k^u)^\mathsf{T} \underbrace{\big(\mathcal{L}_{k+1}^b + (\mathcal{F}_{k+1}^b)^\mathsf{T} (\underbrace{\mathcal{L}_{k+2}^b + \cdots + \tilde{h}^b \mathcal{F}_{H-1}^b \cdots \mathcal{F}_{k+2}^b}_{\eta_{k+2}}) \big)}_{\eta_{k+1}}
\end{aligned} \tag{3.33}$$

As a result, (3.32) is true. $\square$

Next, we show that the control updates found in (3.27) is a descent direction.

Lemma 2. *For the total cost J defined in (3.2) and control updates $\Delta\mathbf{U}$ defined in (3.27, 3.31), the following is true*

$$(\nabla_{\mathbf{U}} J)^\mathsf{T} \Delta\mathbf{U} = -\varepsilon \sum_{k=1}^{H-1} \lambda_k + O(\varepsilon^2) \tag{3.34}$$

where $\lambda_k = (Q_k^u)^\mathsf{T} (Q_k^{uu})^{-1} Q_k^u$.

Proof. In order to show (3.34), it is sufficient to prove

$$\sum_{\tau=k}^{H-1} (\nabla_{\mathbf{u}_\tau} J)^\mathsf{T} \delta \mathbf{u}_\tau = -\varepsilon \sum_{\tau=k}^{H-1} \lambda_\tau + (V_k^b - \eta_k)^\mathsf{T} \delta \mathbf{v}_k + O(\varepsilon^2) \tag{3.35}$$

for $\tau = 1, ..., H-1$. It is straightforward to see that when $k = 1$ we have $\delta \mathbf{v}_1 = 0$. And the above expression is equivalent to (3.34). From now on, we focus on (3.35). First we consider the case when $\tau = H-1$. Applying the results from Lemma 1, the policy (3.27) and expressions from (3.25) (3.28) we have

$$\begin{aligned}
&(\nabla_{\mathbf{u}_{H-1}} J)^\mathsf{T} \delta \mathbf{u}_{H-1} = \big((\mathcal{F}_{H-1}^u)^\mathsf{T} \eta_H + \mathcal{L}_{H-1}^u\big)^\mathsf{T} (\varepsilon \mathbf{I}_{H-1} + \mathbf{L}_{H-1} \delta \mathbf{v}_{H-1}) \\
&= (Q_{H-1}^u)^\mathsf{T} (\varepsilon \mathbf{I}_{H-1} + \mathbf{L}_{H-1} \delta \mathbf{v}_{H-1}) \\
&= -\varepsilon (Q_{H-1}^u)^\mathsf{T} (Q_{H-1}^{uu})^{-1} Q_{H-1}^u + (Q_{H-1}^u)^\mathsf{T} (\mathbf{L}_{H-1} \delta \mathbf{v}_{H-1}) \\
&= -\varepsilon \lambda_{H-1} + \big((Q_{H-1}^u)^\mathsf{T} \mathbf{L}_{H-1} + (Q_{H-1}^b)^\mathsf{T} - (Q_{H-1}^b)^\mathsf{T}\big) \delta \mathbf{v}_{H-1} \\
&= -\varepsilon \lambda_{H-1} + \big((V_{H-1}^b)^\mathsf{T} - ((\mathcal{F}_{H-1}^b)^\mathsf{T} \tilde{h}^b + \mathcal{L}_{H-1}^b)^\mathsf{T}\big) \delta \mathbf{v}_{H-1} \\
&= -\varepsilon \lambda_{H-1} + (V_{H-1}^b - \eta_{H-1})^\mathsf{T} \delta \mathbf{v}_{H-1}
\end{aligned} \tag{3.36}$$

Therefore (3.35) is true for $\tau = H-1$. Now we assume that (3.35) is true for $\tau = i+1$. More precisely we have

$$\sum_{\tau=i+1}^{H-1} (\nabla_{\mathbf{u}_\tau} J)^\mathsf{T} \delta \mathbf{u}_\tau = -\varepsilon \sum_{\tau=i+1}^{H-1} \lambda_\tau + (V_{i+1}^b - \eta_{i+1})^\mathsf{T} \delta \mathbf{v}_{i+1} + O(\varepsilon^2) \tag{3.37}$$

Now we consider the case when $\tau = i$. Applying the results from Lemma 1, the policy (3.27) and expressions from (3.25) (3.28) we have

$$
\begin{aligned}
&\sum_{\tau=i}^{H-1}(\nabla_{\mathbf{u}_\tau}J)^{\mathsf{T}}\delta\mathbf{u}_\tau = \sum_{\tau=i+1}^{H-1}(\nabla_{\mathbf{u}_\tau}J)^{\mathsf{T}}\delta\mathbf{u}_\tau + (\nabla_{\mathbf{u}_i}J)^{\mathsf{T}}\delta\mathbf{u}_i + O(\varepsilon^2)\\
=&-\varepsilon\sum_{\tau=i+1}^{H-1}\lambda_\tau + (V_{i+1}^b-\eta_{i+1})^{\mathsf{T}}\delta\mathbf{v}_{i+1} + (\eta_{i+1}^{\mathsf{T}}\mathcal{F}_i^u + (\mathcal{L}_i^u)^{\mathsf{T}})\delta\mathbf{u}_i + O(\varepsilon^2)\\
=&-\varepsilon\sum_{\tau=i+1}^{H-1}\lambda_\tau + (V_{i+1}^b-\eta_{i+1})^{\mathsf{T}}(\mathcal{F}_i^b\delta\mathbf{v}_i + \mathcal{F}_i^u\delta\mathbf{u}_i + O(\|\delta\mathbf{u}_i\|^2\\
&+\|\delta\mathbf{v}_i\|^2)) + (\eta_{i+1}^{\mathsf{T}}\mathcal{F}_i^u + (\mathcal{L}_i^u)^{\mathsf{T}})\delta\mathbf{u}_i + O(\varepsilon^2)\\
=&-\varepsilon\sum_{\tau=i+1}^{H-1}\lambda_\tau + (V_{i+1}^b-\eta_{i+1})^{\mathsf{T}}\mathcal{F}_i^b\delta\mathbf{v}_i + \Big((\mathcal{F}_i^u)^{\mathsf{T}}V_{i+1}^b + \mathcal{L}_i^u\Big)^{\mathsf{T}}\delta\mathbf{u}_i + O(\varepsilon^2)\\
=&-\varepsilon\sum_{\tau=i+1}^{H-1}\lambda_\tau + (V_{i+1}^b-\eta_{i+1})^{\mathsf{T}}\mathcal{F}_i^b\delta\mathbf{v}_i + (Q_i^u)^{\mathsf{T}}\\
&\quad\Big(-\varepsilon(Q_i^{uu})^{-1}Q_i^u - (Q_i^{uu})^{-1}Q_i^{ub}\delta\mathbf{v}_i\Big) + O(\varepsilon^2)\\
=&-\varepsilon\sum_{\tau=i+1}^{H-1}\lambda_\tau - \varepsilon(Q_i^u)^{\mathsf{T}}(Q_i^{uu})^{-1}Q_i^u + (Q_i^b - \mathcal{L}_i^b - \eta_i + \mathcal{L}_i^b)^{\mathsf{T}}\delta\mathbf{v}_i\\
&-(Q_i^u)^{\mathsf{T}}(Q_i^{uu})^{-1}Q_i^{ub}\delta\mathbf{v}_i + O(\varepsilon^2)\\
=&-\varepsilon\sum_{\tau=i+1}^{H-1}\lambda_\tau - \varepsilon\lambda_i + \Big(Q_i^b - (Q_i^u)^{\mathsf{T}}(Q_i^{uu})^{-1}Q_i^{ub}) - \eta_i\Big)^{\mathsf{T}}\delta\mathbf{v}_i + O(\varepsilon^2)\\
=&-\varepsilon\sum_{\tau=i}^{H-1}\lambda_\tau + (V_i^b-\eta_i)^{\mathsf{T}}\delta\mathbf{v}_i + O(\varepsilon^2),
\end{aligned}
\tag{3.38}
$$

where we have used the fact that $\delta\mathbf{v}_k = O(\varepsilon)$ and $\delta\mathbf{u}_k = O(\varepsilon)$ (see [80]). Now we complete the proof of (3.35) for $k = i$. By induction (3.35) is true for $k = 1, ..., H-1$, hence (3.34) is true. □

Remark 2. *Lemma 2 implies for sufficiently small ε and non-zero λ_k, the control update $\delta\mathbf{u}_k$ is a descent direction along the nominal trajectory.*

Next we provide an expression for the variation of the total cost after using the optimal control policy (3.27).

Lemma 3. *The total cost variation at each iteration is*

$$\Delta J = (\frac{1}{2}\varepsilon^2 - \varepsilon)\sum_{k=1}^{H-1}\lambda_k + O(\varepsilon^3), \tag{3.39}$$

where $\lambda_k = (Q_k^u)^\mathsf{T}(Q_k^{uu})^{-1}Q_k^u$.

Proof. First, let $(\mathbf{b}, \bar{\mathbf{u}})$ be the nominal state-control trajectory at iteration i. We denote the total cost at iteration i (along the nominal trajectory) by

$$J^{(i)} = \sum_{k=1}^{H-1}\tilde{\mathcal{L}}(\mathbf{b}_k, \bar{\mathbf{u}}_k) + \tilde{h}(\mathbf{b}_H) \tag{3.40}$$

and denote the cost at iteration $i+1$ (after applying the policy) (3.27) by $J^{(i+1)}$. In particular, $J^{(i+1)}$ will be equal to V_k^0 in (3.28), plus some extra terms which appear when higher order expansions of Q are considered in (3.24). One can obtain

$$\begin{aligned}
J^{(i+1)} &= V(\mathbf{v}_1, 1) = V_1^0 + O(\varepsilon^3)\\
&= \tilde{\mathcal{L}}(\mathbf{b}_1, \bar{\mathbf{u}}_1) + (\frac{1}{2}\varepsilon^2 - \varepsilon)\lambda_1 + V(\mathbf{b}_2) + O(\varepsilon^3)\\
&= \tilde{\mathcal{L}}(\mathbf{b}_1, \bar{\mathbf{u}}_1) + (\frac{1}{2}\varepsilon^2 - \varepsilon)\lambda_1 + \tilde{\mathcal{L}}(\mathbf{b}_2, \bar{\mathbf{u}}_2) + (\frac{1}{2}\varepsilon^2 - \varepsilon)\lambda_2 + V(\mathbf{b}_3) + O(\varepsilon^3)\\
&= \sum_{k=1}^{H-1}\tilde{\mathcal{L}}(\mathbf{b}_k, \bar{\mathbf{u}}_k) + \tilde{h}(\mathbf{b}_H) + (\frac{1}{2}\varepsilon^2 - \varepsilon)\big(\lambda_1 + \cdots + \lambda_{H-1}\big) + O(\varepsilon^3)\\
&= J^{(i)} + (\frac{1}{2}\varepsilon^2 - \varepsilon)\sum_{k=1}^{H-1}\lambda_k + O(\varepsilon^3)
\end{aligned} \tag{3.41}$$

Therefore $\Delta J = J^{(i+1)} - J^{(i)} = (\frac{1}{2}\varepsilon^2 - \varepsilon)\sum_{k=1}^{H-1}\lambda_k + O(\varepsilon^3)$ which concludes the proof. □

Remark 3. *Based on* (3.39)*, a valid candidate for* ε *in* (3.27) *should satisfy at each iteration* *the following line search condition*

$$J^{(i+1)} - J^{(i)} \leq -\kappa\varepsilon \sum_{k=1}^{H-1} \lambda_k, \tag{3.42}$$

where κ *is a small positive number* $(e.g., \kappa \leq 0.01)$.

Now we show the convergence of the proposed PDDP algorithm.

Theorem 1. *As the iteration number approaches infinity, the total cost* J *and control sequence* $\mathbf{U}$ *converge to a stationary point for arbitrary initialization.*

Proof. From the fact $0 < \varepsilon \leq 1$ it is straightforward to see

$$\varepsilon \leq 1 \Longrightarrow (\frac{1}{2}\varepsilon - 1) \leq -\frac{1}{2} \Longrightarrow (\frac{1}{2}\varepsilon^2 - \varepsilon) \leq -\frac{1}{2}\varepsilon, \tag{3.43}$$

therefore from Lemma 3 we have the cost reduction at the ith iteration

$$\Delta J^{(i)} \leq -\frac{1}{2}\varepsilon \sum_{k=1}^{H-1} (Q_k^u)^{\mathsf{T}} (Q_k^{uu})^{-1} Q_k^u + O(\varepsilon^3). \tag{3.44}$$

From lemma 2, there exists an $\varepsilon_1 \leq 1$ such that

$$(\nabla_{\mathbf{U}} J)^{\mathsf{T}} \Delta \mathbf{U} = -\varepsilon \sum_{k=1}^{H-1} (Q_k^u)^{\mathsf{T}} (Q_k^{uu})^{-1} Q_k^u, \quad \forall \varepsilon \in (0, \varepsilon_1]. \tag{3.45}$$

Therefore the control update (3.26) is a descent direction. In addition, from (3.44), there exists an $\varepsilon_2 \in (0, \varepsilon_1]$ such that

$$\Delta J^{(i)} \leq -\frac{1}{2}\varepsilon \sum_{k=1}^{H-1} (Q_k^u)^{\mathsf{T}} (Q_k^{uu})^{-1} Q_k^u, \quad \forall \varepsilon \in (0, \varepsilon_2], \tag{3.46}$$

which indicates that $J^{(i)}$ is monotonic decreasing for arbitrary initialization. To proceed, we assume that the search space is compact. Hence, since J is a continuous function of $\mathbf{U}$, there exists a control sequence $\mathbf{U}^*$ such that

$$\lim_{i\to\infty} J^{(i)} = J(\mathbf{U}^*). \tag{3.47}$$

Therefore the total cost J is convergent. In addition, (3.47) implies that as $i \to \infty$, for arbitrary initialization and $k = 1, ..., H-1$

$$\begin{aligned} \Delta J^{(i)} \to 0 &\Longrightarrow (Q_k^u)^\mathsf{T}(Q_k^{uu})^{-1}Q_k^u \to 0 \\ Q_k^{uu} \text{ is p.d. } &\Longrightarrow (Q_k^{uu})^{-1}Q_k^u \to \mathbf{0} \end{aligned} \tag{3.48}$$

where $\mathbf{0} = [0, ..., 0]^\mathsf{T} \in \mathbb{R}^{m\times 1}$. The above condition indicates the feedforward policy $\mathbf{I}_k$ defined in (3.26) vanishes as $i \to \infty$. Hence $\delta\mathbf{u}_k \to \mathbf{L}_k\delta\mathbf{v}_k$. The initial state is given so

$$\begin{aligned} &\delta\mathbf{v}_1 = \mathbf{0} \implies \delta\mathbf{u}_1 \to \mathbf{L}_1\delta\mathbf{v}_1 \to \mathbf{0} \\ &\Longrightarrow \delta\mathbf{v}_2 = \mathcal{F}_1^b\delta\mathbf{v}_1 + \mathcal{F}_1^u\delta\mathbf{u}_1 + O(\|\delta\mathbf{v}_1\|^2 + \|\delta\mathbf{u}_1\|^2) \to \mathbf{0} \\ &\Longrightarrow \delta\mathbf{u}_2 \to \mathbf{L}_2\delta\mathbf{v}_2 \to \mathbf{0} \end{aligned} \tag{3.49}$$

It is straightforward to extend this analysis and we have $\delta\mathbf{u}_k \to \mathbf{0}$ for $k = 1, ..., H-1$. Therefore the controls will converge to $\mathbf{U}^*$.

Lastly we show that $\mathbf{U}^*$ is a stationary point. First note that (3.48) implies that $Q_k^u \to \mathbf{0}$ for all time instants. As a consequence $Q_k^b = V_k^b$. By using the expression in Lemma 1 we have

$$\begin{aligned} \nabla_{u_{H-1}^*}J &= (\mathcal{F}_{H-1}^u)^\mathsf{T}\tilde{h}^b + \mathcal{L}_{H-1}^u = Q_{H-1}^u \to \mathbf{0} \\ \nabla_{u_{H-2}^*}J &= (\mathcal{F}_{H-2}^u)^\mathsf{T}\eta_{H-1}^b + \mathcal{L}_{H-2}^u \\ &= (\mathcal{F}_{H-2}^u)^\mathsf{T}\underbrace{\left((\mathcal{F}_{H-1}^b)^\mathsf{T}\tilde{h}^b + \mathcal{L}_{H-1}^b\right)}_{Q_{H-1}^b = V_{H-1}^b} + \mathcal{L}_{H-2}^u = Q_{H-2}^u \to \mathbf{0}. \end{aligned} \tag{3.50}$$

We continue the above analysis backward in time for $\nabla_{u_{H-3}^*}J \to 0, ..., \nabla_{u_1^*}J \to 0$ hence $\nabla_{\mathbf{U}^*}J \to 0$. It is deduced that the controls converge to a stationary point $\mathbf{U}^*$ for arbitrary initialization. □

Remark 4. *The above analysis shows that our method converges to a stationary solution under some mild assumptions. Moreover, an explicit expression for the decrease in the cost after each iteration is provided (3.39). Note that global optimality of the solution to the original problem is not guaranteed here for two reasons: 1) the nonlinear belief dynamics leads to a non-convex optimization problem. 2) The belief is a Gaussian approximation and not necessarily the true posterior distribution, we have discussed this problem in section 3.3.1.*

Remark 5. *Our method can be classified as a second-order optimal control method and therefore is expected to outperform standard first-order gradient-based methods, e.g., [50]. Moreover, under the aforementioned assumptions, PDDP is also capable of achieving locally quadratic convergence rates, i.e., as we get closer to the stationary solution, there exists a constant* $c > 0$ *such that*

$$\|\mathbf{U}^{(i+1)} - \mathbf{U}^{*}\| \leq c\|\mathbf{U}^{(i)} - \mathbf{U}^{*}\|^{2}. \tag{3.51}$$

This has been proven for scalar systems with no terminal cost term in [79] and generic problems in [81]. We refer the reader to these papers for more details, since the same proof can be used here. Note that this extension requires second order expansions of the belief dynamics (4.28)*, which would add an extra term in* Q^{bb}*,* Q^{bu} *and* Q^{uu} *3.25. Nevertheless, in this work we choose to use only the first order approximation of the belief dynamics* (4.28)*. Empirically, neglecting the Hessians of the dynamics results in a more computationally efficient algorithm without sacrificing the quality of solution (see [82] for discussion).*

3.6 Further Analysis

3.6.1 Computational complexity

Approximate inference: The major computational effort is devoted to GP inferences. In particular, the complexity of one-step moment matching is $\mathcal{O}\big((N)^2n^2(n+m)\big)$ [50], which is fixed during the iterative process of PDDP. We found a small number of sampled trajectories ($N \leq 5$) are able to provide good performances for a system of moderate size (6-12 state dimensions). However, for higher dimensional problems, sparse or local approximation of GP (e.g. [61][83][84], etc) may be used to reduce the computational cost of GP dynamics propagation.

Controller learning: According to (3.26), learning policy parameters $\mathbf{I}_k$ and $\mathbf{L}_k$ requires computing the inverse of $\mathbf{Q}_k^{uu}$, which has the computational complexity of $\mathcal{O}(m^3)$, where m is the dimension of control input. For the case of control constrained, the QP also lead to the complexity of $\mathcal{O}(m^3)$. It comes from the Cholesky factorization of the Projected Newton solver. As a local trajectory optimization method, PDDP has a comparable scalability to the classical DDP.

3.6.2 Relation to existing works

Here we summarize the novel features of PDDP in comparison with some notable DDP-related frameworks for stochastic systems (see also Table 4.2). First, PDDP is inherently different from iLQG [27] and sDDP [28], in which the dynamics model is known, and model uncertainty is ignored. Second, PDDP shares some similarities with the belief space iLQG [82] framework, which performs the belief space trajectory optimization based on an extended Kalman filter. Belief space iLQG assumes a dynamics model is given and the stochasticity comes from the process noises. PDDP, however, is a data-driven approach that learns the dynamics models and controls from data, and it takes into account model uncertainties using GPs. Third, PDDP is also comparable with iLQG-LD [59], which is based on learned dynamics using Locally Weighted Projection Regression (LWPR). The Minimax DDP [30] uses a similar model learning approach (RFWR) with partially known dynamics. These methods do not incorporate model uncertainty therefore requires a relatively large amount of data to learn an accurate model. Furthermore, PDDP does not suffer from the high computational cost of finite differences used to numerically compute the first-order expansions [27][82] and second-order expansions [28] of the underlying dynamics. PDDP computes Jacobian matrices analytically (3.15). Furthermore, a recent method AGP-iLQR[68] shows impressive performance and also uses GPs as dynamics model. However, it drops the uncertainty when performing multi-step predictions. In contrast, PDDP takes into account explicit uncertainty and optimizes w.r.t. the belief over state. In terms of optimization criterion, AGP-iLQR incorporates predictive variance of the state transition, this term appears linearly in the cost function. PDDP utilizes the distribution over cost (expectation and variance) as the performance criterion, which is more general. In addition, the proposed approach is a significant extension of the preliminary work on PDDP [85]. For instance, in this work we develop a semiparametric learning scheme by incorporating prior model knowledge; we explore risk-sensitive learning using predicted cost variance; and we take into account control constraints by solving a QP problem (sec. 3.4.4). All of these extensions substantially improve the applicability of the PDDP framework.

PDDP is also related to other RL frameworks that are not based on local trajectory optimization. The most notable algorithm is PILCO [50], a model-based policy search approach using GPs. Their main differences can be summarized as follows: first, PDDP is a self-contained trajectory optimization method that features fast convergence, in contrast, PILCO requires an extra optimizer for policy improvement (e.g., BFGS). Second, PILCO requires designs of policy parameterization and solves relatively higher-dimensional optimization problems (depending on the number of parameters), while PDDP does not require any policy parameterization. Third, PDDP uses a forward-backward sweep scheme, the policy improvement at time step k takes into account improvement at future steps. In contrast, PILCO as well as most policy gradient methods find time-independent policies, which is less efficient.

Many other GP-based planning and Bayesian RL algorithms focus on discrete domains or finite state/action spaces. These work usually provide an error bound for their approximations, e.g.,[86, 87]. Our problem of interest is inherently different since we focus on continuous domains.

	PDDP	**Belief space iLQG**[82]/**sDDP**[88]	**iLQG-LD**[59]	**iLQG**[27], **sDDP**[28]	**AGP-iLQR**[68]	**Minimax DDP**[30, 89]
Optimization variables	belief $\boldsymbol{\mu}, \boldsymbol{\Sigma}$, control $\mathbf{u}$	belief $\boldsymbol{\mu}, \boldsymbol{\Sigma}$, control $\mathbf{u}$	state $\mathbf{x}$, control $\mathbf{u}$	state $\mathbf{x}$, control $\mathbf{u}$	mean of state $\boldsymbol{\mu}$, control $\mathbf{u}$	state $\mathbf{x}$, control $\mathbf{u}$
Optimization criterion	Cost distribution	Expected cost	Expected cost	Expected cost	Expected cost and predictive state variance	Cost incorporates a disturbance term
Dynamics model	Unknown (GP) or partially known (GP+parametric model)	Known	Unknown (LWPR)	Known	Unknown (Approximate GP)	Partially known (RFWR)
Linearization	Analytic Jacobian	Finite differences	Analytic Jacobian	Finite differences	Analytic Jacobian	Analytic Jacobian

Table 3.1: Comparison with DDP-related frameworks

3.7 Experimental Evaluation

In this section we evaluate the PDDP framework using three nontrivial simulated examples. We demonstrate the performance of PDDP by comparative analyses. All experiments were performed in MATLAB on a 3.7GHz Intel i7 PC.

3.7.1 Tasks

In this chapter we consider 3 simulated tasks: Cart-double inverted pendulum swing-up, PUMA-560 robotic arm reaching, and quadrotor flight.

Cart-double inverted pendulum swing-up

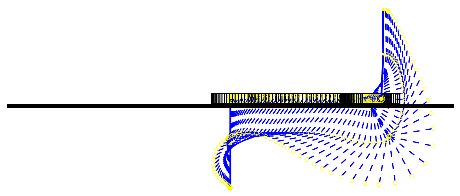
Cart-Double Inverted Pendulum (CDIP) swing-up is a challenging control problem because the system is highly underactuated with 3 degrees of freedom and only 1 control input. The system has 6 state-dimensions (cart position/velocity, link 1,2 angles and angular velocities). The physical model parameters are: masses of the cart and two links, lengths of two links, and the coefficient of friction. The goal of the swing-up problem is to find a sequence of control input to force both pendulums from initial position (π,π) to the inverted position (2π,2π). The balancing task requires the velocity of the cart, angular velocities of both pendulums to be zero. The time horizon for the task is 1.2 second and $\mathrm{d}t = 0.02$. We sample 5 trajectories for each optimization stage. For optimization criterion we use both the expected (risk-neutral) cost and the risk-sensitive cost with $\epsilon = 0.2$. The CDIP task posture is shown in fig.3.2a.

PUMA-560 robotic arm reaching

PUMA-560 is a 3D robotic arm that has 12 state dimensions, 6 degrees of freedom with 6 actuators on the joints. The physical model parameters are the moments of inertia, masses, centers of gravity, lengths and offsets for 6 links. The task is to steer the end-effector to the desired position and orientation. See an illustration in fig.3.2b. The time horizon for the task is 2.0 second and $\mathrm{d}t = 0.02$. We sample 3 trajectories for each optimization stage. For optimization criterion we use the expected cost.

Quadrotor flight

Quadrotors are underactuated rotorcraft which rely on symmetry in order to fly in a conventional, stable flight regime. With 6 degrees of freedom and 4 rotors to control them, the systems attitude is highly coupled with its 3 dimensional movement. See fig.3.7a for an example. The objective is to start at (1, 1, 0.5) and finish at position (0.5, 1, 1.5) after 4 seconds. All velocities and angles begin at zero and should end at zero - thus the quadrotor begins at rest at the start position and should reach the goal position and stop there. The controls are thrust forces of the 4 rotors and we consider the control constraint $\mathbf{u}_{\text{min}} = 0.5, \mathbf{u}_{\text{max}} = 3$. A comparison between the constrained and unconstrained controllers is shown in fig.3.8. For optimization criterion we use the risk-sensitive cost with $\epsilon = 0.5$. We assume partial knowledge of the dynamics model is known, i.e., the structure of the transition dynamics with the following unknown parameters: moments of inertia about X,Y,Z axis, the distance of rotor to the center of mass and the mass of the quadrotor. Physical model parameters were learned using 5 sample rollouts and another 5 rollouts were used for GP model learning. See sec.3.3.3 for details regarding semiparametric model learning and inference.


(a)

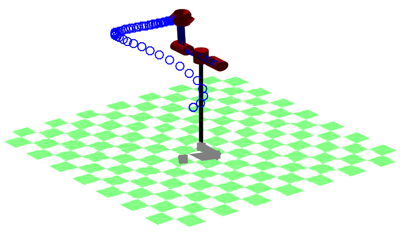

(b)

Figure 3.2: CDIP and Puma-560 tasks.

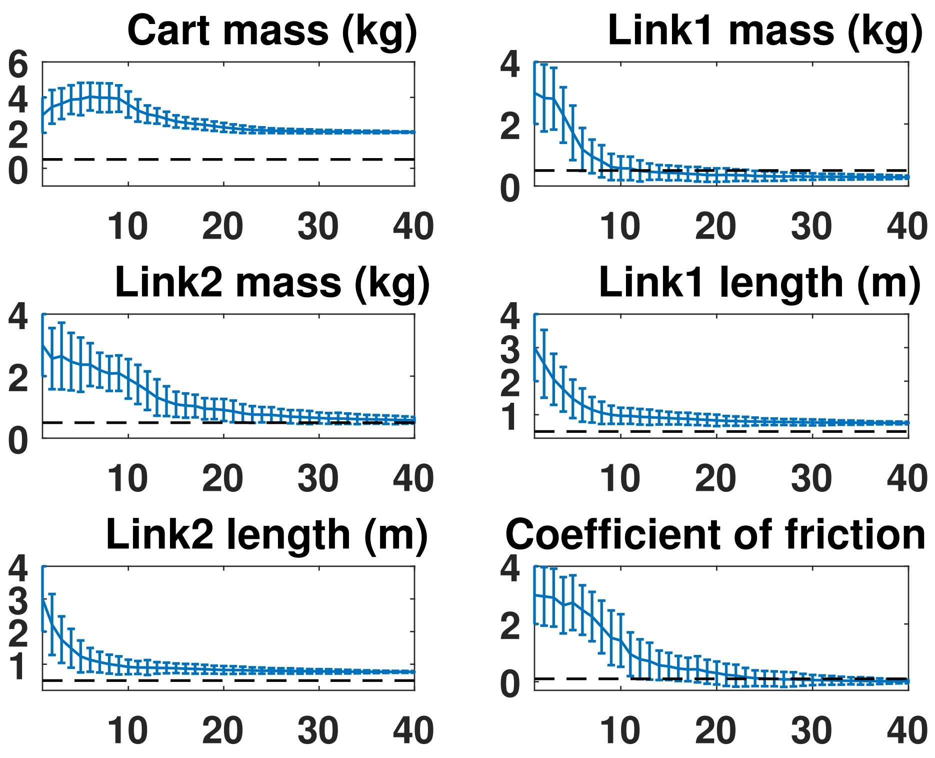
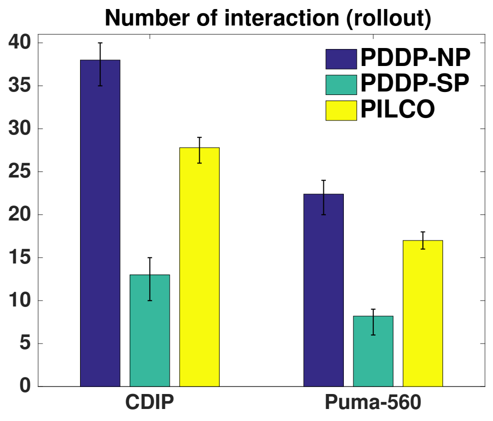


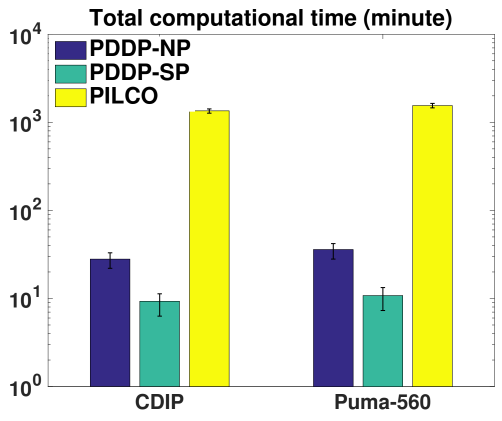
Figure 3.3: Cross-entropy parameter optimization for CDIP dynamics model with bad initial guesses. The horizontal axis is iteration number and vertical axis is parameter value. Dash lines are the true parameter values. Error bars show the mean and variance of sampling distributions at each iteration. Note that some parameters converge to the true values while some others converge to local minima.

3.7.2 Data efficiency

For the case of unknown dynamics, both PDDP and PILCO are based on nonparametric GP dynamics models. As shown in fig.3.4a, PDDP performs slightly worse than PILCO in terms of data efficiency based on the number of interactions required to learn the same tasks. The number of interactions indicates the total amount of sample rollouts required from the physical systems. Possible reasons for the slightly worse performances are: i) PDDP's policy is linear which might be restrictive, while PILCO yields nonlinear policy parameterizations; ii) As a global method, PILCO encourages more exploration especially in the early stages of learning. In contrast, PDDP is a local method, it searches for solution in the neighborhoods of trajectories. Despite these differences, PDDP offers close performances compared to PILCO in terms of data efficiency with less computational cost. According to the analysis in [50], PDDP would outperform most RL algorithms by at least an order of magnitude in terms of data efficiency.



(a)



(b)

Figure 3.4: Comparison between PDDP and PILCO in terms of the number of interactions with the physical system (a) and total computational time (b) required for learning the CDIP and Puma-560 tasks. PDDP-NP and PDDP-SP correspond to the nonparametric and semiparametric cases, respectively. The results were averaged over 5 independent trials.

3.7.3 Computational efficiency

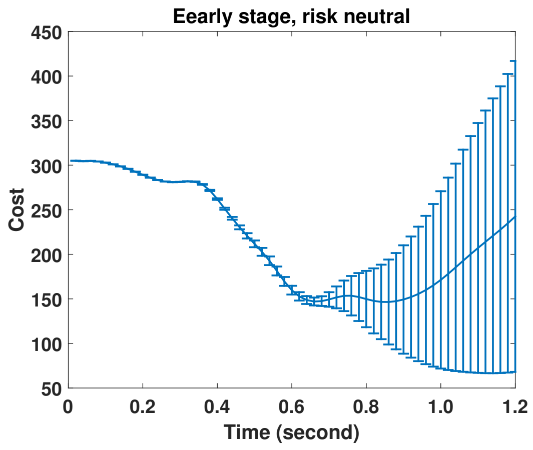
In terms of total computational time required to obtain the final controller, PDDP outperforms PILCO significantly as shown in fig.3.4b. PILCO requires an iterative method (e.g.,CG or BFGS) to solve global and high dimensional optimization problems (depending on the policy parameterization). In contrast, PDDP successively computes local optimal controls (3.26) without an extra optimizer and features fast convergence [79]. The major computational demand of PDDP comes from the approximate inference in GPs (moment matching). We did not use any approximate GP methods such as local GP [61, 90] or sparse GP [83, 91, 92, 93] approximations. These methods can be applied in PDDP to speed up computations when the training data size is large, e.g., [94].

3.7.4 Nonparametric vs. semiparametric learning

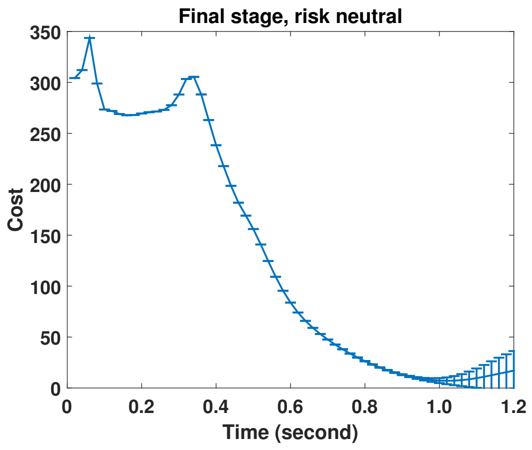
As shown in fig.3.4, PDDP based on semiparametric model learning and inference (with known basis function) outperforms both PDDP and PILCO under unknown dynamics in terms of data and computational efficiency. To distinguish the effect of parametric and nonparametric models, at each learning stage we use the old samples from previous stages for parameter estimation. The purpose of the parametric model is to generalize to the state-action space regions that are far away from the training data used for nonparametric model learning. This result is not surprising but it shows the potential of applying PDDP in a more practical way. In real world applications these basis functions are usually available for mechanical/robotics systems from physics-based models. Learning the model from scratch is conceptually appealing but practically challenging when the available data are not sufficiently informative. In addition, the semiparametric approach takes into account the uncertainty of parametric modeling error. For instance, fig.3.3 show the physical model parameter learning performance for the CDIP task using algorithm 1. Due to data insufficiency some parameter estimations deviate from the true values. This parametric error is learned as the GP error dynamics model (see section 3.3.3). The semiparametric PDDP could be applied to more challenging tasks compared with purely nonparametric methods. For example, the quadrotor system has highly coupled outputs. In contrast, the nonparametric GP assumed independent output for each dimension, therefore has limited modeling power in this case. We use the semiparametric PDDP to successfully learn a policy in a single optimization stage. Results for policy learning and testing are shown in fig.3.7. The optimization takes less than 20 iterations (see fig.3.7b).

3.7.5 Risk-sensitive vs. risk-neutral learning

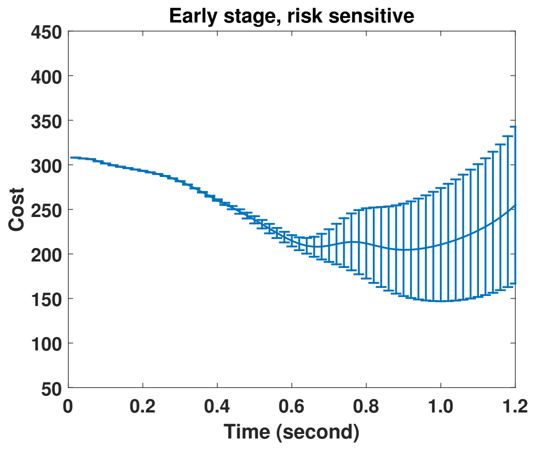
Another distinct feature for PDDP is that it incorporates the probability distribution over the cost as the performance criterion (see section 3.4.2). More precisely the mean and variance of the predicted cost. We evaluate the effect of the cost function by comparing the risk-sensitive and risk-neutral cases for the CDIP task. Fig.3.5 shows the predicted cost distributions during the early and final learning stages. The risk-sensitive learning leads to relatively less cost uncertainties due to the penalization on predicted cost variance. Fig.3.6 shows the learning curves for 4 cases on the same task. We use 3 and 5 rollouts at each optimization phase for both risk-neutral and risk-sensitive learning. An interesting phenomenon is that when using only 3 rollouts (180 data points) for learning. The risk-sensitive policy outperforms risk-neutral policy significantly. This is because when the sample data are sparse in state-action regions, the risk-sensitive criterion restricts exploration in the local region that are close to the sampled data and nominal trajectories. In contrast, in the risk-neutral case, applying PDDP policy results in higher costs because the cost variance is higher. When using 5 rollouts (300 data points) per stage, the risk-sensitive policy learns slower than risk-neutral in the early stages. But they perform similarly in the final stages. PDDP with both criteria are able to learn the task within about 7-8 optimization stages. Even in the risk-neutral case, PDDP would avoid explorations in highly uncertain regions because the expected cost (3.17) depends on predictive variance of the state. Risk-sensitive learning is more conservative and shows encouraging performance when using a very small amount of training data.



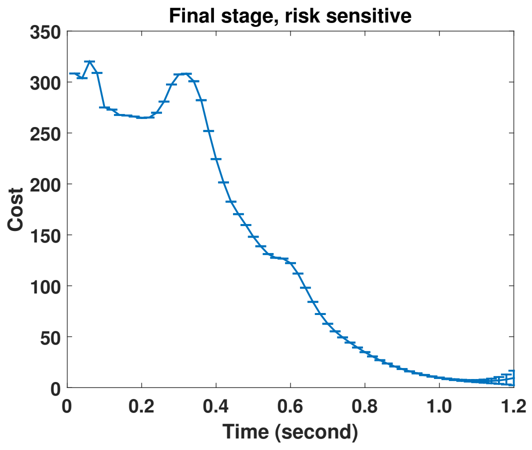
(a)



(b)



(c)



(d)

Figure 3.5: Comparison of the predicted trajectory cost distributions between risk-neutral (a,b) and risk-sensitive (c,d) learning via PDDP. Early stage and final stage correspond to the predictions after 1 and 8 optimization stages.

3.8 Discussion

In this work our goal is to address the principle challenges of applying reinforcement learning (RL) in complex real-world scenarios, namely data efficiency, computational cost and scalability. Thus we have introduced Probabilistic Differential Dynamic Programming (PDDP), a model-based RL framework for systems with unknown or partially known dynamics. PDDP is derived based on two main components: 1) local trajectory optimization

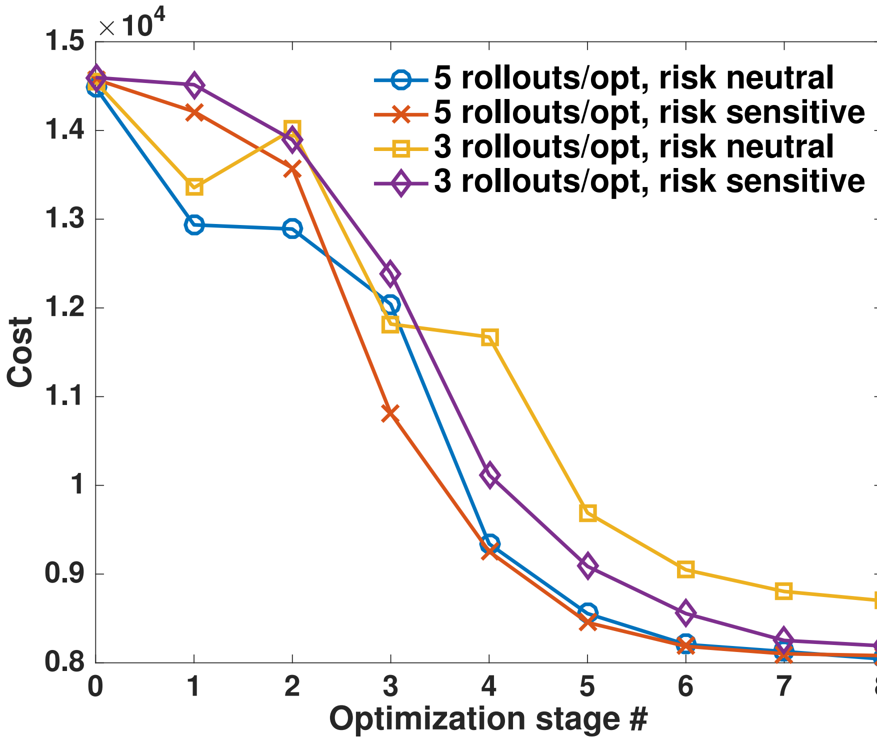
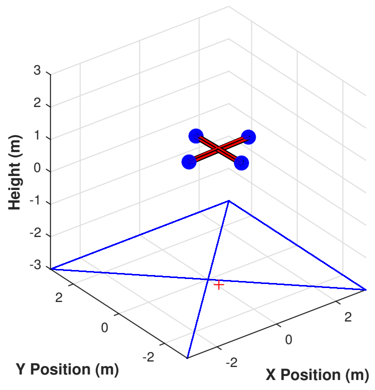
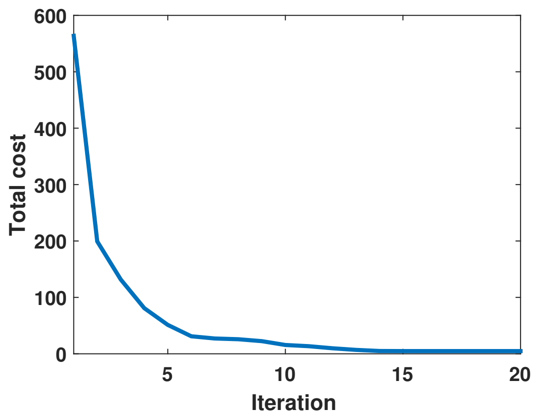


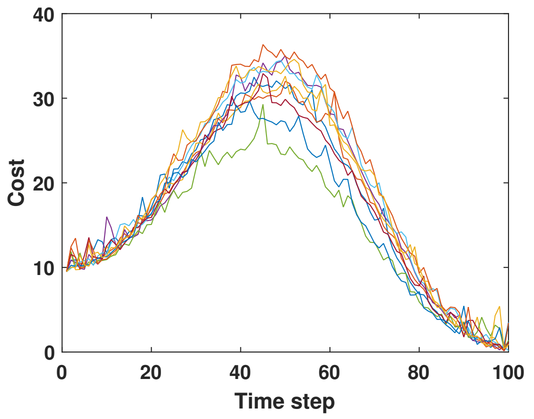
Figure 3.6: Total trajectory costs by applying PDDP policy after each optimization stage for risk-neutral ($\epsilon = 0$) and risk-sensitive learning ($\epsilon = 0.2$). 3 and 5 rollouts were used for learning in both cases.



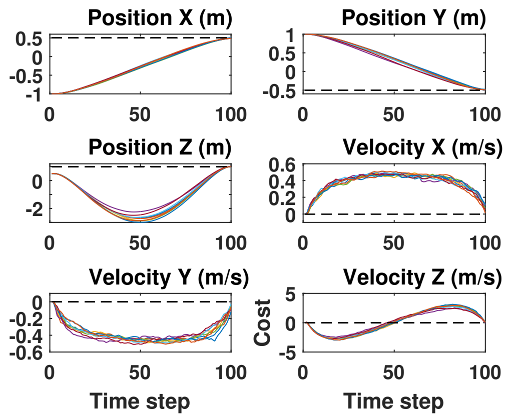
(a)



(b)



(c)



(d)

Figure 3.7: The quadrotor flight task. (a) Quadrotor simulation environment. (b) Expected trajectory cost reduction during PDDP optimization. (c) Trajectory costs over 10 independent trials using the optimized control policy. (d) State trajectories of the quadrotor task collected over 10 independent trials using the optimized controller. Dash lines are desired states.

via forward-backward sweeps, which is a classical method under the name Differential Dynamics Programming (DDP) [26]. 2) Probabilistic modeling and approximate inference with Gaussian processes (GPs). PDDP integrates the aforementioned components by representing the system dynamics using GPs and performing optimization in Gaussian belief spaces. To further improve its applicability, we explored the case of risk-sensitive trajectory optimization.

PDDP generalizes the deterministic [26] and stochastic [27, 28] trajectory optimization to a probabilistic setting, i.e. probabilistic trajectory optimization. By taking advantages

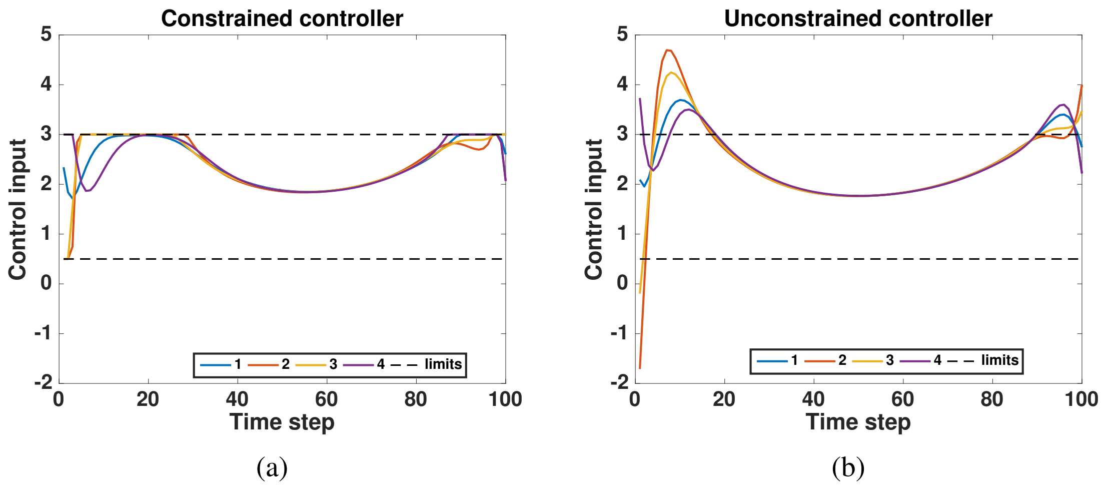


Figure 3.8: Constrained and unconstarined control inputs (4 motor torques) for the quadrotor task. The constraint is incorporated by solving a QP (sec.3.4.4)

of recent development in GPs, PDDP offers data efficiency superior to methods based on learned determinstic models, and comparable to the state of the art GP-based policy search method [50]. Compared with typical gradient-based policy search methods, PDDP features a more computationally efficient policy improvement scheme. PDDP yields local control policies and requires no a-priori policy parameterization. These strengths lead to a combination of data efficiency, computational efficiency and scalability. Theoretically, we provide analyses showing our algorithm converges to a stationary point globally. To our best knowledge, our analyses offer the most general convergence proof for DDP-related methods [26, 79, 80, 27, 29, 58, 68, 59]

The computational efficiency of probabilistic inference can be further improved. The possibility of using sampling-based methods (such as in PEGASUS [95]) instead of deterministic inference has been discussed in [55]. However, it may lead to derivatives with high variance and it is more suitable for sampling-based (gradient-free) methods, e.g.,[96]. Approximate GPs, such as local GP [61, 90] or sparse GP [83, 92], can be exploited to improve the computational efficiency. For instance, Sparse Spectrum GPs have been used for fast inference in trajectory optimization [94, 97, 98] when 1) the optimization needs to be performed in real-time and receding horizon fashion, 2) training dataset is big and 3) the task, dynamics and environment are time-varying.

Application of probabilistic trajectory optimization to real robotic systems is worth further investigation. In addition, our approximate inference method does not have an explicit error bound. Numerical methods such as Gaussian quadrature could provide more accurate approximations and an error bound [71]. However, such numerical methods are usually more computationally expensive than closed-form computation as we do here. Further investigation on combining the benefits of these methods with ours is left for future work.

CHAPTER 4
PATH INTEGRAL CONTROL UNDER UNCERTAIN DYNAMICS

In this chapter we present two stochastic optimal control (SOC) frameworks for systems with unknown or partially known dynamics based on the path integral formulation and probabilistic inference. We show that the optimal control can be computed in a sampling-based and gradient-based fashion without replying on local approximation used in PDDP (see chapter 3).

4.1 Introduction

Stochastic optimal control (SOC) is a general and powerful framework with applications in many areas of science and engineering. However, despite broad applicability, solving SOC problems remains challenging for systems in high-dimensional continuous state-action spaces [56, 57]. Over the last decade, SOC based on exponential transformation of the value function has demonstrated remarkable applicability in solving real world control and planning problems. In control theory, the exponential transformation of the value function was introduced in [32, 33]. In the past decade it has been explored in terms of path integral interpretations and theoretical generalizations [34, 35, 36, 37], discrete time formulations [38], and scalable reinforcement learning/control algorithms [39, 40, 41, 42, 43, 44]. The resulting stochastic optimal control frameworks are known as Path Integral (PI) control for continuous time, Kullback Leibler (KL) control for discrete time, or more generally Linearly Solvable Optimal Control [38, 45]. In this work we focus on the PI control framework.

One of the most attractive characteristics of PI control is that optimal control problems can be solved with forward sampling of Stochastic Differential Equations (SDEs). However, for model-based PI frameworks [34, 35, 36, 43, 37], full knowledge of a dynamics model is required to perform sampling. And these samples do not take into account any model uncertainty. In addition, for PI-based model-free policy search frameworks [39, 40], an intelligent choice of policy parameterization is required and expert demonstrations are usually necessary for policy initialization. In the following we develop two data-driven PI control methods to cope with these issues. In section 4.2, we introduce a gradient-based algorithm where all gradients are computed analytically. In section 4.3, we present a sampling-based approach where each sample consists of predictive state distributions computed via sparse spectrum GP inference.

4.2 Gradient-based Approach

Motivated by the aforementioned limitations, in this chapter we introduce a sample efficient, gradient-based approach to PI control. Different from existing PI control approaches, our method combines the benefits of PI control theory [34, 35, 36] and probabilistic model-based reinforcement learning methodologies [55, 50]. The main characteristics of the our approach are summarized as follows

- It extends the PI control theory [34, 35, 36] to the case of uncertain systems. The structural constraint is enforced between the control cost and uncertainty of the learned dynamics, which can be viewed as a generalization of previous work [34, 35, 36].

- Different from parameterized PI controllers [39, 40, 43, 37], we find analytic control law without any policy parameterization.

- Rather than keeping a fixed control cost weight [34, 35, 36, 39, 99], or ignoring the constraint between control authority and noise level [40], in this work the control cost weight is adapted based on the explicit uncertainty of the learned dynamics model.

- The algorithm operates in a different manner compared to existing PI-related methods that perform forward sampling [34, 35, 36, 39, 99, 40, 41, 43, 37]. More precisely our method perform successive deterministic approximate inference and backward computation of optimal control law.

- The proposed model-based approach is significantly more sample efficient than sampling-based PI control [34, 35, 36, 99]. In RL setting our method is comparable to the state-of-the-art RL methods [50, 85] in terms of sample and computational efficiency.

- Thanks to the linearity of the backward Chapman-Kolmogorov PDE, the learned controllers can be generalized to new tasks without re-sampling by constructing composite controllers. In contrast, most policy search and trajectory optimization methods [39, 40, 43, 50, 85, 100, 101, 102] find policy parameters that can not be generalized.

4.2.1 Preliminaries

In contrast to the generic problem defined in chapter 1, we consider a slightly different class of nonlinear stochastic system described by the following differential equation

$$\mathrm{d}\mathbf{x} = \big(\mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u}\big)\mathrm{d}t + \mathbf{B}\mathrm{d}\boldsymbol{\omega}, \tag{4.1}$$

with state $\mathbf{x} \in \mathbb{R}^n$, control $\mathbf{u} \in \mathbb{R}^m$, and standard Brownian motion noise $\boldsymbol{\omega} \in \mathbb{R}^p$ with variance $\boldsymbol{\Sigma}_\omega$. $\mathbf{f}(\mathbf{x})$ is the unknown drift term (passive dynamics), $\mathbf{G}(\mathbf{x}) \in \mathbb{R}^{n\times m}$ is the control matrix and $\mathbf{B} \in \mathbb{R}^{n\times p}$ is the diffusion matrix. Given some previous control $\mathbf{u}^{old}$, we seek the optimal control correction term $\delta\mathbf{u}$ such that the total control $\mathbf{u} = \mathbf{u}^{old} + \delta\mathbf{u}$. The original system becomes

$$\mathrm{d}\mathbf{x} = \big(\mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})(\mathbf{u}^{old} + \delta\mathbf{u})\big)\mathrm{d}t + \mathbf{B}\mathrm{d}\boldsymbol{\omega} = \underbrace{\big(\mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u}^{old}\big)}_{\tilde{\mathbf{f}}(\mathbf{x},\mathbf{u}^{old})}\mathrm{d}t + \mathbf{G}(\mathbf{x})\delta\mathbf{u}\mathrm{d}t + \mathbf{B}\mathrm{d}\boldsymbol{\omega}.$$

In this work we assume the dynamics based on the previous control can be represented by Gaussian processes (GP) such that

$$\mathbf{f}_{\mathbb{GP}}(\mathbf{x}) = \tilde{\mathbf{f}}(\mathbf{x}, \mathbf{u}^{old})\mathrm{d}t, \tag{4.2}$$

where $\mathbf{f}_{\mathbb{GP}}$ is the GP representation of the biased drift term $\tilde{\mathbf{f}}$ under the previous control. Now the original dynamical system (4.20) can be represented as follow

$$\mathrm{d}\mathbf{x} = \mathbf{f}_{\mathbb{GP}} + \mathbf{G}\delta\mathbf{u}\mathrm{d}t, \qquad \mathbf{f}_{\mathbb{GP}} \sim \mathcal{GP}(0, \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)), \tag{4.3}$$

where we use a prior of zero mean and covariance function $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_s^2 \exp(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}\mathbf{W}(\mathbf{x}_i - \mathbf{x}_j)) + \delta_{ij}\sigma_\omega^2$, with $\sigma_s, \sigma_\omega, \mathbf{W}$ the hyper-parameters. δ_{ij} is the Kronecker symbol that is one iff $i = j$ and zero otherwise. Samples over $\mathbf{f}_{\mathbb{GP}}$ can be drawn using an vector of i.i.d. Gaussian variable Ω

$$\tilde{\mathbf{f}}_{\mathbb{GP}} = \mu_f + \mathbf{L}_f\Omega \tag{4.4}$$

where $\mathbf{L}_f$ is obtained using Cholesky factorization such that $\mathbf{\Sigma}_f = \mathbf{L}_f\mathbf{L}_f^{\mathrm{T}}$. Note that generally Ω is an infinite dimensional vector and we can use the same sample to represent uncertainty during learning [103]. Without loss of generality we assume Ω to be the standard zero-mean Brownian motion. For the rest of the chapter we use simplified notations with subscripts indicating the time step. The discrete-time representation of the system is $\mathbf{x}_{t+\mathrm{d}t} = \mathbf{x}_t + \boldsymbol{\mu}_{ft} + \mathbf{G}_t\delta\mathbf{u}_t\mathrm{d}t + \mathbf{L}_{ft}\Omega_t\sqrt{\mathrm{d}t}$, and the conditional probability of $\mathbf{x}_{t+\mathrm{d}t}$ given $\mathbf{x}_t$ and $\delta\mathbf{u}_t$ is a Gaussian $\mathrm{p}\big(\mathbf{x}_{t+\mathrm{d}t}|\mathbf{x}_t, \delta\mathbf{u}_t\big) = \mathcal{N}\big(\boldsymbol{\mu}_{t+\mathrm{d}t}, \mathbf{\Sigma}_{t+\mathrm{d}t}\big)$, where $\boldsymbol{\mu}_{t+\mathrm{d}t} = \mathbf{x}_t + \boldsymbol{\mu}_{ft} + \mathbf{G}_t\delta\mathbf{u}_t$ and $\mathbf{\Sigma}_{t+\mathrm{d}t} = \mathbf{\Sigma}_{ft}$. In this work we consider a finite-horizon stochastic optimal control problem

$$J(\mathbf{x}_0) = \mathbb{E}\Big[q(\mathbf{x}_T) + \int_{t=0}^{T} \mathcal{L}(\mathbf{x}_t, \delta\mathbf{u}_t)\mathrm{d}t\Big],$$

where the immediate cost is defined as $\mathcal{L}(\mathbf{x}_t, \mathbf{u}_t) = q(\mathbf{x}_t) + \frac{1}{2}\delta\mathbf{u}_t^{\mathrm{T}}\mathbf{R}_t\delta\mathbf{u}_t$, and $q(\mathbf{x}_t) = (\mathbf{x}_t - \mathbf{x}_t^d)^{\mathrm{T}}\mathbf{Q}(\mathbf{x}_t - \mathbf{x}_t^d)$ is a quadratic cost function where $\mathbf{x}_t^d$ is the desired state. $\mathbf{R}_t = \mathbf{R}(\mathbf{x}_t)$ is a state-dependent positive definite weight matrix. Next we show the linearized Hamilton-Jacobi-Bellman equation for this class of optimal control problems.

4.2.2 Linearized Hamilton-Jacobi-Bellman Equation for Uncertain Dynamics

At each iteration the goal is to find the optimal control update $\delta\mathbf{u}_t$ that minimizes the value function

$$V(\mathbf{x}_t, t) = \min_{\delta\mathbf{u}_t} \mathbb{E}\Big[\int_t^{t+\mathrm{d}t} \mathcal{L}(\mathbf{x}_t, \delta\mathbf{u}_t)\mathrm{d}t + V(\mathbf{x}_t + \mathrm{d}\mathbf{x}_t, t + \mathrm{d}t)\mathrm{d}t|\mathbf{x}_t\Big]. \tag{4.5}$$

(4.22) is the Bellman equation. By approximating the integral for a small $\mathrm{d}t$ and applying Itô's rule we obtain the Hamilton-Jacobi-Bellman (HJB) equation (detailed derivation is skipped):

$$-\partial_t V_t = \min_{\delta\mathbf{u}_t}(q_t + \frac{1}{2}\delta\mathbf{u}_t^{\mathrm{T}}\mathbf{R}_t\delta\mathbf{u}_t + (\boldsymbol{\mu}_{ft} + \mathbf{G}_t\delta\mathbf{u}_t)^{\mathrm{T}}\nabla_{\mathbf{x}}\mathbf{V}_t + \frac{1}{2}\mathbf{Tr}(\boldsymbol{\Sigma}_{ft}\nabla_{\mathbf{xx}}\mathbf{V}_t)).$$

To find the optimal control update, we take gradient of the above expression (inside the parentheses) with respect to $\delta\mathbf{u}_t$ and set to 0. This yields $\delta\mathbf{u}_t = -\mathbf{R}_t^{-1}\mathbf{G}_t^{\mathrm{T}}\nabla_{\mathbf{x}}\mathbf{V}_t$. Inserting this expression into the HJB equation yields the following nonlinear and second order PDE

$$-\partial_t V_t = q_t + (\nabla_{\mathbf{x}}\mathbf{V}_t)^{\mathrm{T}}\boldsymbol{\mu}_{ft} - \frac{1}{2}(\nabla_{\mathbf{x}}\mathbf{V}_t)^{\mathrm{T}}\mathbf{G}_t\mathbf{R}^{-1}\mathbf{G}_t^{\mathrm{T}}\nabla_{\mathbf{x}}\mathbf{V}_t + \frac{1}{2}\mathbf{Tr}(\boldsymbol{\Sigma}_{ft}\nabla_{\mathbf{xx}}\mathbf{V}_t). \tag{4.6}$$

In order to solve the above PDE we use the exponential transformation of the value function $V_t = -\lambda \log \Psi_t$, where $\Psi_t = \Psi(\mathbf{x}_t)$ is called the *desirability* of $\mathbf{x}_t$. The corresponding partial derivatives can be found as $\partial_t V_t = -\frac{\lambda}{\Psi_t}\partial_t\Psi_t$, $\nabla_{\mathbf{x}}\mathbf{V}_t = -\frac{\lambda}{\Psi_t}\nabla_{\mathbf{x}}\Psi_t$ and $\nabla_{\mathbf{xx}}\mathbf{V}_t =$

$\frac{\lambda}{\Psi_t^2}\nabla_{\mathbf{x}}\Psi_t\nabla_{\mathbf{x}}\Psi_t^{\mathrm{T}} - \frac{\lambda}{\Psi_t}\nabla_{\mathbf{xx}}\Psi_t$. Inserting these terms to (4.35) results in

$$\begin{aligned}\frac{\lambda}{\Psi_t}\partial_t\Psi_t = q_t - \frac{\lambda}{\Psi_t}(\nabla_{\mathbf{x}}\Psi_t)^{\mathrm{T}}\boldsymbol{\mu}_{ft} - \frac{\lambda^2}{2\Psi_t^2}(\nabla_{\mathbf{x}}\Psi_t)^{\mathrm{T}}\mathbf{G}_t\mathbf{R}_t^{-1}\mathbf{G}_t^{\mathrm{T}}\nabla_{\mathbf{x}}\Psi_t \\ + \frac{\lambda}{2\Psi_t^2}\,\mathbf{Tr}((\nabla_{\mathbf{x}}\Psi_t)^{\mathrm{T}}\boldsymbol{\Sigma}_{ft}\nabla_{\mathbf{x}}\Psi_t) - \frac{\lambda}{2\Psi_t}\,\mathbf{Tr}(\nabla_{\mathbf{xx}}\Psi_t\boldsymbol{\Sigma}_{ft}).\end{aligned} \tag{4.7}$$

The quadratic terms $\nabla_{\mathbf{x}}\Psi_t$ will cancel out under the assumption of $\lambda\mathbf{G}_t\mathbf{R}_t^{-1}\mathbf{G}_t^{\mathrm{T}} = \boldsymbol{\Sigma}_{ft}$. This constraint is different from existing works in path integral control [34, 35, 36, 39, 99, 37] where the constraint is enforced between the additive noise covariance and control authority, more precisely $\lambda\mathbf{G}_t\mathbf{R}_t^{-1}\mathbf{G}_t^{\mathrm{T}} = \mathbf{B}\boldsymbol{\Sigma}_\omega\mathbf{B}^{\mathrm{T}}$. The new constraint enables an adaptive update of control cost weight based on explicit uncertainty of the learned dynamics. In contrast, most existing works use a fixed control cost weight [34, 35, 36, 39, 99, 41, 43, 37]. This condition also leads to more exploration (more aggressive control) under high uncertainty and less exploration with more certain dynamics. Given the aforementioned assumption, the above PDE is simplified as

$$\partial_t\Psi_t = \frac{1}{\lambda}q_t\Psi_t - \boldsymbol{\mu}_{ft}^{\mathrm{T}}\nabla_{\mathbf{x}}\Psi_t - \frac{1}{2}\,\mathbf{Tr}(\nabla_{\mathbf{xx}}\Psi_t\boldsymbol{\Sigma}_{ft}), \tag{4.8}$$

subject to the terminal condition $\Psi_T = \exp(-\frac{1}{\lambda}q_T)$. The resulting Chapman-Kolmogorov PDE (4.36) is linear. In general, solving (4.36) analytically is intractable for nonlinear systems and cost functions. We apply the Feynman-Kac formula which gives a probabilistic representation of the solution of the linear PDE (4.36)

$$\Psi_t = \lim_{\mathrm{d}t\to 0}\int \mathrm{p}(\tau_t|\mathbf{x}_t)\exp\big(-\frac{1}{\lambda}(\sum_{j=t}^{T-\mathrm{d}t}q_j\mathrm{d}t)\big)\Psi_T\mathrm{d}\tau_t, \tag{4.9}$$

where τ_t is the state trajectory from time t to T. The optimal control is obtained as

$$\begin{aligned}&\mathbf{G}_t\delta\hat{\mathbf{u}}_t = -\mathbf{G}_t\mathbf{R}_t^{-1}\mathbf{G}_t^{\mathrm{T}}(\nabla_{\mathbf{x}}V_t) = \lambda\mathbf{G}_t\mathbf{R}_t^{-1}\mathbf{G}_t^{\mathrm{T}}\Big(\frac{\nabla_{\mathbf{x}}\Psi_t}{\Psi_t}\Big) = \mathbf{\Sigma}_{ft}\Big(\frac{\nabla_{\mathbf{x}}\Psi_t}{\Psi_t}\Big)\\ \Longrightarrow&\hat{\mathbf{u}}_t = \mathbf{u}_t^{old} + \delta\hat{\mathbf{u}}_t = \mathbf{u}_t^{old} + \mathbf{G}_t^{-1}\mathbf{\Sigma}_{ft}\Big(\frac{\nabla_{\mathbf{x}}\Psi_t}{\Psi_t}\Big).\end{aligned} \tag{4.10}$$

Rather than computing $\nabla_{\mathbf{x}}\Psi_t$ and Ψ_t, the optimal control $\hat{\mathbf{u}}_t$ can be approximated based on path costs of sampled trajectories. Next we briefly review some of the existing approaches.

4.2.3 Relation to Existing Works

According to the path integral control theory [34, 35, 36, 39, 99, 37], the stochastic optimal control problem becomes an approximation problem of a path integral (4.37). This problem can be solved by forward sampling of the uncontrolled ($\mathbf{u} = 0$) SDE (4.20). The optimal control $\hat{\mathbf{u}}_t$ is approximated based on path costs of sampled trajectories. Therefore the computation of optimal controls becomes a forward process. More precisely, when the control and noise act in the same subspace, the optimal control can be evaluated as the weighted average of the noise $\hat{\mathbf{u}}_t = \mathbb{E}_{\mathrm{p}(\tau_t|\mathbf{x}_t)}\big[\mathrm{d}\boldsymbol{\omega}_t\big]$, where the probability of a trajectory is $\mathrm{p}(\tau_t|\mathbf{x}_t) = \frac{\exp(-\frac{1}{\lambda}S(\tau_t|\mathbf{x}_t))}{\int\exp(-\frac{1}{\lambda}S(\tau_t|\mathbf{x}_t))\mathrm{d}\tau}$, and $S(\tau_t|\mathbf{x}_t)$ is defined as the path cost computed by performing forward sampling. However, these approaches require a large amount of samples from a given dynamics model, or extensive trials on physical systems when applied in model-free reinforcement learning settings. In order to improve sample efficiency, a nonparametric approach was developed by representing the desirability Ψ_t in terms of linear operators in a reproducing kernel Hilbert space (RKHS) [41]. As a model-free approach, it allows sample re-use but relies on numerical methods to estimate the gradient of desirability, i.e., $\nabla_{\mathbf{x}}\Psi_t$, which can be computationally expensive. On the other hand, computing the analytic expressions of the path integral embedding is intractable and requires exact knowledge of the system dynamics. Furthermore, the control approximation is based on samples from the uncontrolled dynamics, which is usually not sufficient for highly nonlinear or underactuated systems.

Another class of PI-related method is based on policy parameterization. Notable approaches include PI^2 [39], PI^2-CMA [40], PI-REPS[43] and recently developed state-dependent PI[37]. The limitations of these methods are: 1) They do not take into account model uncertainty in the passive dynamics $\mathbf{f}(\mathbf{x})$. 2) The imposed policy parameterizations restrict optimal control solutions. 3) The optimized policy parameters can not be generalized to new tasks. A brief comparison of some of these methods can be found in Table 1. Motivated by the challenge of combining sample efficiency and generalizability, next we introduce a probabilistic model-based approach to compute the optimal control (4.10) analytically.

	PI [34, 35], iterative PI [99]	PI^2[39], PI^2-CMA [40]	PI-REPS[43]	State feedback PI[37]	Our method
Structural constraint	$\lambda \mathbf{G}_t \mathbf{R}_t^{-1} \mathbf{G}_t^{\mathrm{T}} = \mathbf{B} \boldsymbol{\Sigma}_\omega \mathbf{B}^{\mathrm{T}}$	same as PI	same as PI	same as PI	$\lambda \mathbf{G} \mathbf{R}^{-1} \mathbf{G}^{\mathrm{T}} = \boldsymbol{\Sigma}_f$
Dynamics model	model-based	model-free	model-based	model-based	GP model-based
Policy parameterization	No	Yes	Yes	Yes	No

Table 4.1: Comparison with some notable and recent path integral-related approaches.

4.2.4 Analytic Path Integral Control: a Forward-Backward Scheme

In order to derive the proposed framework, firstly we learn the function $\mathbf{f}_{\mathbb{GP}}(\mathbf{x}_t) = \tilde{\mathbf{f}}(\mathbf{x}, \mathbf{u}^{old})\mathrm{d}t + \mathbf{B}\mathrm{d}\boldsymbol{\omega}$ from sampled data. Learning the continuous mapping from state to state transition can be viewed as an inference with the goal of inferring the state transition $\mathrm{d}\tilde{\mathbf{x}}_t = \mathbf{f}_{\mathbb{GP}}(\mathbf{x}_t)$. The kernel function has been defined in Sec.4.2.1, which can be interpreted as a similarity measure of random variables. More specifically, if the training input $\mathbf{x}_i$ and $\mathbf{x}_j$ are close to each other in the kernel space, their outputs $\mathrm{d}\mathbf{x}_i$ and $\mathrm{d}\mathbf{x}_j$ are highly correlated. Given a sequence of states $\{\mathbf{x}_0, \dots \mathbf{x}_T\}$, and the corresponding state transition $\{\mathrm{d}\tilde{\mathbf{x}}_0, \dots, \mathrm{d}\tilde{\mathbf{x}}_T\}$, the posterior distribution can be obtained by conditioning the joint prior distribution on the observations. In this work we make the standard assumption of independent outputs (no correlation between each output dimension).

To propagate the GP-based dynamics over a trajectory of time horizon T we employ the moment matching approach [48, 50] to compute the predictive distribution. Given an input distribution over the state $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, the predictive distribution over the state at $t + \mathrm{d}t$ can be approximated as a Gaussian $\mathrm{p}(\mathbf{x}_{t+\mathrm{d}t}) \approx \mathcal{N}(\boldsymbol{\mu}_{t+\mathrm{d}t}, \boldsymbol{\Sigma}_{t+\mathrm{d}t})$ such that

$$\boldsymbol{\mu}_{t+\mathrm{d}t} = \boldsymbol{\mu}_t + \boldsymbol{\mu}_{ft}, \quad \boldsymbol{\Sigma}_{t+\mathrm{d}t} = \boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_{ft} + \mathbb{COV}[\mathbf{x}_t, \mathrm{d}\tilde{\mathbf{x}}_t] + \mathbb{COV}[\mathrm{d}\tilde{\mathbf{x}}_t, \mathbf{x}_t]. \tag{4.11}$$

The above formulation is used to approximate one-step transition probabilities over the trajectory. Details regarding the moment matching method can be found in [48, 50]. All mean and variance terms can be computed analytically. The hyper-parameters $\sigma_s, \sigma_\omega, \mathbf{W}$ are learned by maximizing the log-likelihood of the training outputs given the inputs [47]. Given the approximation of transition probability (4.27), we now introduce a Bayesian non-parametric formulation of path integral control based on probabilistic representation of the dynamics. Firstly we perform approximate inference (forward propagation) to obtain the Gaussian belief (predictive mean and covariance of the state) over the trajectory. Since the exponential transformation of the state cost $\exp(-\frac{1}{\lambda}q(\mathbf{x})\mathrm{d}t)$ is an unnormalized Gaussian $\mathcal{N}(\mathbf{x}^d, \frac{2\lambda}{\mathrm{d}t}\mathbf{Q}^{-1})$. We can evaluate the following integral analytically

$$\int \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \exp\big(-\frac{1}{\lambda}q_j\mathrm{d}t\big)\mathrm{d}\mathbf{x}_j = \Big|\mathbf{I} + \frac{\mathrm{d}t}{2\lambda}\boldsymbol{\Sigma}_j\mathbf{Q}\Big|^{-\frac{1}{2}} \exp\Big(-\frac{1}{2}(\boldsymbol{\mu}_j - \mathbf{x}_j^d)^{\mathrm{T}}\frac{\mathrm{d}t}{2\lambda}\mathbf{Q}(\mathbf{I} + \frac{\mathrm{d}t}{2\lambda}\lambda\boldsymbol{\Sigma}_j\mathbf{Q})^{-1}(\boldsymbol{\mu}_j - \mathbf{x}_j^d)\Big), \tag{4.12}$$

for $j = t + \mathrm{d}t, ..., T$. Thus given a boundary condition $\Psi_T = \exp(-\frac{1}{\lambda}q_T)$ and predictive distribution at the final step $\mathcal{N}(\boldsymbol{\mu}_T, \boldsymbol{\Sigma}_T)$, we can evaluate the one-step backward desirability $\Psi_{T-\mathrm{d}t}$ analytically using the above expression (4.12). More generally we use the following recursive rule

$$\Psi_{j-\mathrm{d}t} = \Phi(\mathbf{x}_j, \Psi_j) = \int \mathcal{N}\big(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\big) \exp\big(-\frac{1}{\lambda}q_j\mathrm{d}t\big)\Psi_j\mathrm{d}\mathbf{x}_j, \tag{4.13}$$

for $j = t+\mathrm{d}t, ..., T-\mathrm{d}t$. Since we use deterministic approximate inference based on (4.27) instead of explicitly sampling from the corresponding SDE, we approximate the conditional distribution $\mathrm{p}(\mathbf{x}_j|\mathbf{x}_{j-\mathrm{d}t})$ by the Gaussian predictive distribution $\mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$. Therefore the path integral

$$\begin{aligned}\Psi_t &= \int \mathrm{p}\Big(\tau_t|\mathbf{x}_t\Big)\exp\Big(-\frac{1}{\lambda}(\sum_{j=t}^{T-\mathrm{d}t} q_j \mathrm{d}t)\Big)\Psi_T \mathrm{d}\tau_t \\ &\approx \int ... \underbrace{\int \mathcal{N}\Big(\boldsymbol{\mu}_{T-\mathrm{d}t}, \boldsymbol{\Sigma}_{T-\mathrm{d}t}\Big)\exp\Big(-\frac{1}{\lambda}q_{T-\mathrm{d}t}\mathrm{d}t\Big)\underbrace{\int \mathcal{N}\Big(\boldsymbol{\mu}_T, \Sigma_T\Big)\underbrace{\exp\Big(-\frac{1}{\lambda}q_T\Big)}_{\Psi_T}\mathrm{d}\mathbf{x}_T}_{\Psi_{T-\mathrm{d}t}}\mathrm{d}\mathbf{x}_{T-\mathrm{d}t}}_{\Psi_{T-2\mathrm{d}t}} ...\mathrm{d}\mathbf{x}_{t+\mathrm{d}t} \\ &= \int \mathcal{N}\Big(\boldsymbol{\mu}_{t+\mathrm{d}t}, \boldsymbol{\Sigma}_{t+\mathrm{d}t}\Big)\exp\Big(-\frac{1}{\lambda}q_{t+\mathrm{d}t}\mathrm{d}t\Big)\Psi_{t+\mathrm{d}t}\mathrm{d}\mathbf{x}_{t+\mathrm{d}t} = \Phi(\mathbf{x}_{t+\mathrm{d}t}, \Psi_{t+\mathrm{d}t}). \end{aligned} \tag{4.14}$$

We evaluate the desirability Ψ_t backward in time by successive computation using the above recursive expression. The optimal control law $\hat{\mathbf{u}}_t$ (4.10) requires gradients of the desirability function with respect to the state, which can be computed backward in time as well. For simplicity we denote the function $\Phi(\mathbf{x}_j, \Psi_j)$ by Φ_j. Thus we compute the gradient of the recursive expression (4.14)

$$\nabla_{\mathbf{x}}\Psi_{j-\mathrm{d}t} = \Psi_j \nabla_{\mathbf{x}}\Phi_j + \Phi_j \nabla_{\mathbf{x}}\Psi_j, \tag{4.15}$$

where $j = t+\mathrm{d}t, ..., T-\mathrm{d}t$. Given the expression in (4.12) we compute the gradient terms in (4.15) as

$$\nabla_{\mathbf{x}}\Phi_j = \frac{\mathrm{d}\Phi_j}{\mathrm{dp}(\mathbf{x}_j)}\frac{\mathrm{dp}(\mathbf{x}_j)}{\mathrm{d}\mathbf{x}_t} = \frac{\partial\Phi_j}{\partial\boldsymbol{\mu}_j}\frac{\mathrm{d}\boldsymbol{\mu}_j}{\mathrm{d}\mathbf{x}_t} + \frac{\partial\Phi_j}{\partial\boldsymbol{\Sigma}_j}\frac{\mathrm{d}\boldsymbol{\Sigma}_j}{\mathrm{d}\mathbf{x}_t}, \text{ where } \frac{\partial\Phi_j}{\partial\boldsymbol{\mu}_j} = \Phi_j(\boldsymbol{\mu}_j - \mathbf{x}_j^d)^T\frac{\mathrm{d}t}{2\lambda}\mathbf{Q}(\mathbf{I} + \frac{\mathrm{d}t}{2\lambda}\lambda\boldsymbol{\Sigma}_j\mathbf{Q})^{-1},$$
$$\frac{\partial\Phi_j}{\partial\boldsymbol{\Sigma}_j} = \frac{\Phi_j}{2}\Big(\frac{\mathrm{d}t}{2\lambda}\mathbf{Q}(\mathbf{I} + \frac{\mathrm{d}t}{2\lambda}\lambda\boldsymbol{\Sigma}_j\mathbf{Q})^{-1}(\boldsymbol{\mu}_j - \mathbf{x}_j^d)(\boldsymbol{\mu}_j - \mathbf{x}_j^d)^T - \mathbf{I}\Big)\frac{\mathrm{d}t}{2\lambda}\mathbf{Q}(\mathbf{I} + \frac{\mathrm{d}t}{2\lambda}\lambda\boldsymbol{\Sigma}_j\mathbf{Q})^{-1}, \text{ and}$$
$$\frac{\mathrm{d}\{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}}{\mathrm{d}\mathbf{x}_t} = \Big\{\frac{\partial\boldsymbol{\mu}_j}{\partial\boldsymbol{\mu}_{j-\mathrm{d}t}}\frac{\mathrm{d}\boldsymbol{\mu}_{j-\mathrm{d}t}}{\mathrm{d}\mathbf{x}_t} + \frac{\partial\boldsymbol{\mu}_j}{\partial\boldsymbol{\Sigma}_{j-\mathrm{d}t}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{j-\mathrm{d}t}}{\mathrm{d}\mathbf{x}_t}, \frac{\partial\boldsymbol{\Sigma}_j}{\partial\boldsymbol{\mu}_{j-\mathrm{d}t}}\frac{\mathrm{d}\boldsymbol{\mu}_{j-\mathrm{d}t}}{\mathrm{d}\mathbf{x}_t} + \frac{\partial\boldsymbol{\Sigma}_j}{\partial\boldsymbol{\Sigma}_{j-\mathrm{d}t}}\frac{\mathrm{d}\boldsymbol{\Sigma}_{j-\mathrm{d}t}}{\mathrm{d}\mathbf{x}_t}\Big\}.$$

The term $\nabla_{\mathbf{x}}\Psi_{T-\mathrm{d}t}$ is compute similarly. The partial derivatives $\frac{\partial\boldsymbol{\mu}_j}{\partial\boldsymbol{\mu}_{j-\mathrm{d}t}}, \frac{\partial\boldsymbol{\mu}_j}{\partial\boldsymbol{\Sigma}_{j-\mathrm{d}t}}, \frac{\partial\boldsymbol{\Sigma}_j}{\partial\boldsymbol{\mu}_{j-\mathrm{d}t}}, \frac{\partial\boldsymbol{\Sigma}_j}{\partial\boldsymbol{\Sigma}_{j-\mathrm{d}t}}$ can be computed analytically as in [50]. We compute all gradients using this scheme without any numerical method (finite differences, etc.). Given Ψ_t and $\nabla_{\mathbf{x}}\Psi_t$, the optimal control takes a analytic form as in eq.(4.10). Since Ψ_t and $\nabla_{\mathbf{x}}\Psi_t$ are explicit functions of $\mathbf{x}_t$, the resulting control law is essentially different from the feedforward control in sampling-based path integral control frameworks [34, 35, 36, 39, 99] as well as the parameterized state feedback PI control policies [43, 37]. Notice that at current time step t, we update the control sequence $\hat{\mathbf{u}}_{t,...,T}$ using the presented forward-backward scheme. Only $\hat{\mathbf{u}}_t$ is applied to the system to move to the next step, while the controls $\hat{\mathbf{u}}_{t+\mathrm{d}t,...,T}$ are used for control update at future steps. The transition sample recorded at each time step is incorporated to update the GP model of the dynamics. A summary of the proposed algorithm is shown in **Algorithm** 3.

Algorithm 3 Sample efficient path integral control under uncertain dynamics

```
 1: Initialization: Apply random controls $\hat{\mathbf{u}}_{0,..,T}$ to the physical system (4.20), record data.
 2: repeat
 3:     for t=0:T do
 4:         Incorporate transition sample to learn GP dynamics model.
 5:         repeat
 6:             Approximate inference for predictive distributions using $\mathbf{u}^{old}_{t,..,T} = \hat{\mathbf{u}}_{t,..,T}$,
    see (4.27).
 7:             Backward computation of optimal control updates $\delta\hat{\mathbf{u}}_{t,..,T}$, see
    (4.14)(4.15)(4.10).
 8:             Update optimal controls $\hat{\mathbf{u}}_{t,..,T} = \mathbf{u}^{old}_{t,..,T} + \delta\hat{\mathbf{u}}_{t,..,T}$.
 9:         until Convergence.
10:         Apply optimal control $\hat{\mathbf{u}}_t$ to the system. Move one step forward and record
    data.
11:     end for
12: until Task learned.
```

4.2.5 Generalization to Unlearned Tasks without Sampling

In this section we describe how to generalize the learned controllers for new (unlearned) tasks without any interaction with the real system. The proposed approach is based on the compositionality theory [104] in linearly solvable optimal control (LSOC). We use superscripts to denote previously learned task indexes. Firstly we define a distance measure between the new target $\bar{\mathbf{x}}^d$ and old targets $\mathbf{x}^{dk}$, $k = 1, .., K$, i.e., a Gaussian kernel

$$\omega^k = \exp\Big(- \frac{1}{2}(\bar{\mathbf{x}}^d - \mathbf{x}^{dk})^{\mathrm{T}}\mathbf{P}(\bar{\mathbf{x}}^d - \mathbf{x}^{dk})\Big), \tag{4.16}$$

where $\mathbf{P}$ is a diagonal matrix (kernel width). The composite terminal cost $\bar{q}(\mathbf{x}_T)$ for the new task becomes

$$\bar{q}(\mathbf{x}_T) = -\lambda \log\left(\frac{\sum_{k=1}^{K} \omega^k \exp(-\frac{1}{\lambda} q^k(\mathbf{x}_T))}{\sum_{k=1}^{K} \omega^k}\right), \tag{4.17}$$

where $q^k(\mathbf{x}_T)$ is the terminal cost for old tasks. For conciseness we define a normalized distance measure $\tilde{\omega}^k = \frac{\omega^k}{\sum_{k=1}^{K} \omega^k}$, which can be interpreted as a probability weight. Based on (4.17) we have the composite terminal desirability for the new task which is a linear combination of Ψ_T^k

$$\bar{\Psi}_T = \exp\Big(-\frac{1}{\lambda}\bar{q}(\mathbf{x}_T)\Big) = \sum_{k=1}^{K} \tilde{\omega}^k \Psi_T^k. \tag{4.18}$$

Since Ψ_t^k is the solution to the linear Chapman-Kolmogorov PDE (4.36), the linear combination of desirability (4.18) holds everywhere from t to T as long as it holds on the boundary (terminal time step). Therefore we obtain the composite control

$$\bar{\mathbf{u}}_t = \sum_{k=1}^{K} \frac{\tilde{\omega}^k \Psi_t^k}{\sum_{k=1}^{K} \tilde{\omega}^k \Psi_t^k} \hat{\mathbf{u}}_t^k. \tag{4.19}$$

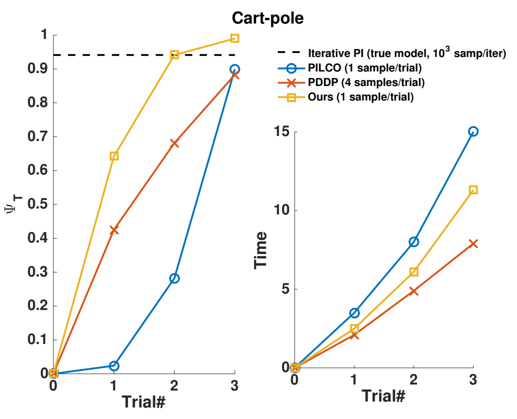
The composite control law in (4.19) is essentially different from an interpolating control law[104]. It enables sample-free controllers that constructed from learned controllers for different tasks. This scheme can not be adopted in policy search or trajectory optimization methods such as [39, 40, 43, 50, 85, 100, 101, 102]. Alternatively, generalization can be achieved by imposing task-dependent policies [105]. However, this approach might restrict the choice of optimal controls given the assumed structure of control policy.

4.2.6 Experiments and Analysis

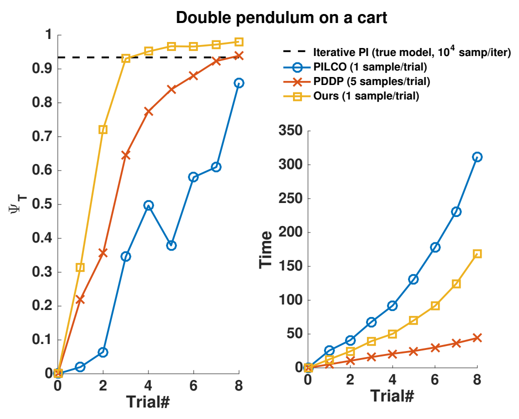
We consider 3 simulated RL tasks: cart-pole (CP) swing up, double pendulum on a cart (DPC) swing up, and PUMA-560 robotic arm reaching. The CP and DPC systems consist of a cart and a single/double-link pendulum. The tasks are to swing-up the single/double-link pendulum from the initial position (point down). Both CP and DPC are under-actuated systems with only one control acting on the cart. PUMA-560 is a 3D robotic arm that has 12 state dimensions, 6 degrees of freedom with 6 actuators on the joints. The task is to steer the end-effector to the desired position and orientation.

In order to demonstrate the performance, we compare the proposed control framework with three related methods: iterative path integral control [99] with known dynamics model, PILCO [50] and PDDP [85]. Iterative path integral control is a sampling-based stochastic control method. It is based on importance sampling using controlled diffusion process rather than passive dynamics used in standard path integral control [34, 35, 36]. Iterative PI control is used as a baseline with a given dynamics model. PILCO is a model-based policy search method that features state-of-the-art data efficiency in terms of number of trials required to learn a task. PILCO requires an extra optimizer (such as BFGS) for policy improvement. PDDP is a Gaussian belief space trajectory optimization approach. It performs dynamic programming based on local approximation of the learned dynamics and value function. Both PILCO and PDDP are applied with unknown dynamics. In this work we do not compare our method with model-free PI-related approaches such as [39, 40, 41, 43] since these methods would certainly cost more samples than model-based methods such as PILCO and PDDP. The reason for choosing these two methods for comparison is that our method adopts a similar model learning scheme while other state-of-the-art methods, such as [100] is based on a different model.

In **experiment 1** we demonstrate the sample efficiency of our method using the CP and DPC tasks. For both tasks we choose $T = 1.2$ and $\mathrm{d}t = 0.02$ (60 time steps per rollout). The iterative PI [99] with a given dynamics model uses 10^3/10^4 (CP/DPC) sample rollouts per iteration and 500 iterations at each time step. We initialize PILCO and the proposed method by collecting 2/6 sample rollouts (corresponding to 120/360 transition samples) for CP/DPC tasks respectively. At each trial (on the true dynamics model), we use 1 sample rollout for PILCO and our method. PDDP uses 4/5 rollouts (corresponding to 240/300 transition samples) for initialization as well as at each trial for the CP/DPC tasks. Fig. 4.1 shows the results in terms of Ψ_T and computational time. For both tasks our method shows higher desirability (lower terminal state cost) at each trial, which indicates higher sample efficiency for task learning. This is mainly because our method performs online re-optimization at each time step. In contrast, the other two methods do not use this scheme. However we assume partial information of the dynamics ($\mathbf{G}$ matrix) is given. PILCO and PDDP perform optimization on entirely unknown dynamics. In many robotic systems $\mathbf{G}$ corresponds to the inverse of the inertia matrix, which can be identified based on data as well. In terms of computational efficiency, our method outperforms PILCO since we compute the optimal control update analytically, while PILCO solves large scale nonlinear optimization problems to obtain policy parameters. Our method is more computational expensive than PDDP because PDDP seeks local optimal controls that rely on linear approximations, while our method is a global optimal control approach. Despite the relatively higher computational burden than PDDP, our method offers reasonable efficiency in terms of the time required to reach the baseline performance.



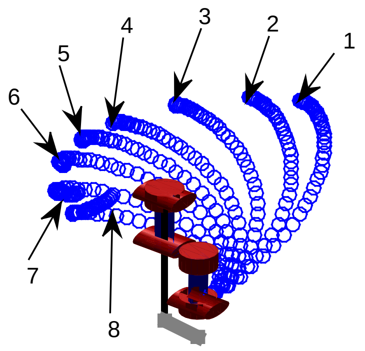
(a)



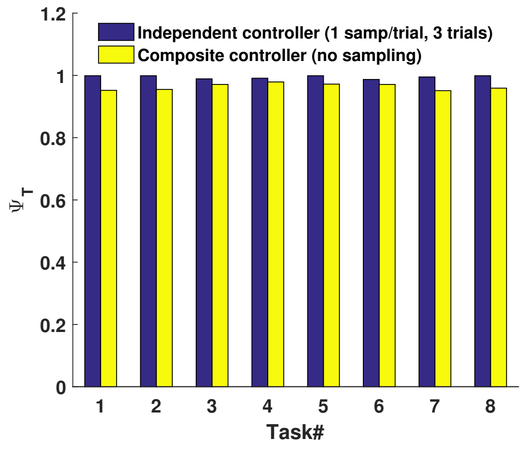
(b)

Figure 4.1: Comparison in terms of sample efficiency and computational efficiency for (a) cart-pole and (b) double pendulum on a cart swing-up tasks. Left subfigures show the terminal desirability Ψ_T (for PILCO and PDDP, Ψ_T is computed using terminal state costs) at each trial. Right subfigures show computational time (in minute) at each trial.

In **experiment 2** we demonstrate the generalizability of the learned controllers to new tasks using the composite control law (4.19) based on the PUMA-560 system. We use $T = 2$ and $\mathrm{d}t = 0.02$ (100 time steps per rollout). First we learn 8 independent controllers using **Algorithm** 3. The target postures are shown in Fig. 4.2. For all tasks we initialize with 3 sample rollouts and 1 sample at each trial. Blue bars in Fig. 4.2b shows the desirabilities Ψ_T after 3 trials. Next we use the composite law (4.19) to construct controllers without re-sampling using 7 other controllers learned using **Algorithm** 3. For instance the composite controller for task#1 is found as $\bar{\mathbf{u}}_t^1 = \sum_{k=2}^{8} \frac{\tilde{\omega}^k \Psi_t^k}{\sum_{k=2}^{8} \tilde{\omega}^k \Psi_t^k} \hat{\mathbf{u}}_t^k$. The performance comparison of the composite controllers with controllers learned from trials is shown in Fig. 4.2. It can be seen that the composite controllers give close performance as independently learned controllers. The compositionality theory [104] generally does not apply to policy search methods and trajectory optimizers such as PILCO, PDDP, and other recent methods [100, 101, 102]. Our method benefits from the compositionality of control laws that can be applied for multi-task control without re-sampling.



(a)



(b)

Figure 4.2: Resutls for the PUMA-560 tasks. (a) 8 tasks tested in this experiment. Each number indicates a corresponding target posture. (b) Comparison of the controllers learned independently from trials and the composite controllers without sampling. Each composite controller is obtained (4.19) from 7 other independent controllers learned from trials.

4.2.7 Summary and Discussion

We presented an iterative learning control framework that can find optimal controllers under uncertain dynamics using very small number of samples. This approach is closely related to the family of path integral (PI) control algorithms. Our method is based on a forward-backward optimization scheme, which differs significantly from current PI-related approaches. Moreover, it combines the attractive characteristics of probabilistic model-based reinforcement learning and linearly solvable optimal control theory. These characteristics include sample efficiency, optimality and generalizability. By iteratively updating the control laws based on probabilistic representation of the learned dynamics, our method demonstrated encouraging performance compared to the state-of-the-art model-based methods. In addition, our method showed promising potential in performing multi-task control based on the compositionality of learned controllers. Besides the assumed structural constraint between control cost weight and uncertainty of the passive dynamics, the major limitation is that we have not taken into account the uncertainty in the control matrix $\mathbf{G}$.

4.3 Sampling-based Approach

In this section we develop a sampling-based method that is slightly more general than the gradient-based method introduced previously. Instead of using full GPs for dynamics modeling and prediction, we leverage the Sparse Spectrum Gaussian Process (SSGP) [91], which is based on kernel function approximation using finite dimensional random feature mappings [106]. Algorithms for SSGP regression have demonstrated a superior combination of computational efficiency and predictive accuracy compared to approximation strategies such as the Fully Independent Training Conditional (FITC) model [91] and Locally Weighted Projection Regression (LWPR) [107]. In this work we use SSGP regression to generate sample rollouts in the belief space, therefore model uncertainty can be incorporated explicitly. In order to improve convergence speed, we propose a covariance matrix adaptation scheme to update the exploration magnitude automatically within the probability-weighted averaging framework.

In the following, we summarize the distinct characteristics of our method:

- The proposed method is data-driven and no prior knowledge of the system dynamics is required.
- Sampling is performed in belief space. Each sample takes into account predictive model uncertainty.
- The exploration noise covariance matrix is adapted via probability-weighted averaging.

4.3.1 Preliminaries

In contrast to the special class of systems considered in the previous method, here we consider a general class of nonlinear stochastic system described by the differential equation

$$\mathrm{d}\mathbf{x} = f(\mathbf{x},\mathbf{u})\mathrm{d}t + \mathrm{d}\boldsymbol{\omega}, \tag{4.20}$$

with state $\mathbf{x} \in \mathbb{R}^n$, control $\mathbf{u} \in \mathbb{R}^m$, and standard Brownian motion noise $\boldsymbol{\omega} \in \mathbb{R}^p$. $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is the unknown transition function. In this section we consider a finite-horizon stochastic optimal control problem

$$\min_{\mathbf{u}} \underbrace{\mathbb{E}\Big[q\big(\mathbf{x}(T)\big) + \int_{t=0}^{T} \mathcal{L}\big(\mathbf{x}(t),\mathbf{u}(t)\big)\mathrm{d}t\Big]}_{J(\mathbf{x}(0))}, \tag{4.21}$$

where the immediate cost is defined as $\mathcal{L}(\mathbf{x},\mathbf{u}) = q(\mathbf{x}) + \frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u}$. $\mathbf{R}$ is a positive definite weight matrix. For the rest of our analysis, we discretize the time using the Euler scheme as $k = 1, 2, ..., H$ with time step $\Delta t = \frac{T}{H-1}$ and denote $\mathbf{x}_k = \mathbf{x}(t_k)$. We use this subscript rule for other time-varying variables as well. The discretized system dynamics can be written as $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k + \sqrt{\Delta t}\xi_k$ where $\Delta\mathbf{x}_k = \Delta t f(\mathbf{x}_k, \mathbf{u}_k)$ and ξ_k is IID with $\mathcal{N}(0, I)$. To simplify notation we define $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \Delta t f(\mathbf{x}_k, \mathbf{u}_k)$. The goal is to find the optimal control $\mathbf{u}_k$ at each time step that minimizes the value function

$$V(\mathbf{x}_k, k) = \min_{\mathbf{u}_k} \mathbb{E}\Big[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) + V(\mathbf{x}_{k+1}, k+1)|\mathbf{x}_k\Big]. \tag{4.22}$$

where V is called value function. The above equation is known as the Bellman equation. Note that the dynamics model $\mathbf{f}$ is unknown and needs to be learned from data. In the next section, we present a probabilistic scheme to learn the dynamics.

4.3.2 Model Learning via Sparse Spectrum Gaussian Processes

In standard GP regression (GPR), we assume the transition function has a prior distribution $\mathbf{f}(\tilde{\mathbf{x}}) \sim \mathcal{GP}(m, k)$, where $m : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ and $k : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ are the mean and covariance functions, respectively. Without loss of generality, we consider zero mean $m = 0$ and the popular Squared Exponential (SE) covariance function with Automatic Relevance Determination (ARD) distance measure

$$k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) = \sigma_f^2 \exp(-\frac{1}{2}(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j)^{\mathsf{T}}\mathbf{P}^{-1}(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j)) + \sigma_n^2\delta_{ij}, \tag{4.23}$$

where δ_{ij} is a Kronecker delta which is one iff $i = j$ and zero otherwise. $\mathbf{P} = \text{diag}([\ l_1^2 \ \ ... \ \ l_{n+m}^2\ ])$. The hyperparameters of the kernel consist of the signal variance σ_f^2, the noise variance σ_n^2 and the length scales for input space $\mathbf{l} = [\ l_1 \ ... \ l_{n+m}\ ]$. Given a dataset of state-control pairs and the corresponding state transition $\mathcal{D} = \{(\mathbf{x}_i; \mathbf{u}_i), \Delta\mathbf{x}_i\}_{i=1}^N$, the posterior distribution is exactly Gaussian and can be computed in closed-form. Unfortunately, GPR exhibits significant practical limitations for learning and inference on large datasets due to its $O(N^3)$ computation and $O(N^2)$ space complexity, which is a direct consequence of having to store and invert a $N \times N$ matrix [47]. This computational inefficiency is a bottleneck for applying

GP-based method in real-time.

In order to scale up kernel methods such as GPs, random features can be used to approximate kernel functions. Based on *Bochner's theorem* [108], any shift-invariant kernel function can be represented as the Fourier transform of a unique measure

$$k(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j) = \int_{\mathbb{R}^{n+m}} e^{i\boldsymbol{\omega}^\mathsf{T}(\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j)} p(\boldsymbol{\omega}) d\boldsymbol{\omega} = \mathbb{E}_{\boldsymbol{\omega}}[\mathbf{z}_{\boldsymbol{\omega}}(\tilde{\mathbf{x}}_i)^\mathsf{T}\mathbf{z}_{\boldsymbol{\omega}}(\tilde{\mathbf{x}}_j)],$$

where $\mathbf{z}_{\boldsymbol{\omega}}(\tilde{\mathbf{x}}) = [\cos(\boldsymbol{\omega}^\mathsf{T}\tilde{\mathbf{x}}) \;\; \sin(\boldsymbol{\omega}^\mathsf{T}\tilde{\mathbf{x}})]^\mathsf{T}$. We can approximate the SE kernel function by drawing r random samples from the distribution $\mathrm{p}(\boldsymbol{\omega}) = \mathcal{N}(0, \mathbf{P}^{-1})$

$$k(\tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_k) \approx \sum_{i=1}^{r} \boldsymbol{\phi}_{\boldsymbol{\omega}_i}(\tilde{\mathbf{x}}_j)^\mathsf{T}\boldsymbol{\phi}_{\boldsymbol{\omega}_i}(\tilde{\mathbf{x}}_k) = \boldsymbol{\phi}(\tilde{\mathbf{x}}_j)^\mathsf{T}\boldsymbol{\phi}(\tilde{\mathbf{x}}_k), \tag{4.24}$$

where $\boldsymbol{\phi}_{\boldsymbol{\omega}_i}(\tilde{\mathbf{x}}) = \frac{\sigma_f}{\sqrt{r}}[\; \cos(\boldsymbol{\omega}_i^\mathsf{T}\tilde{\mathbf{x}}) \;\; \sin(\boldsymbol{\omega}_i^\mathsf{T}\tilde{\mathbf{x}}) \;]^\mathsf{T}$. is a *feature mapping*. Therefore the state transition function can be represented as a weighted sum of the feature functions

$$\Delta\mathbf{x} = \mathbf{f}(\tilde{\mathbf{x}}) = \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\tilde{\mathbf{x}}), \quad \boldsymbol{\phi}(\tilde{\mathbf{x}}) = \begin{bmatrix} \frac{\sigma_f}{\sqrt{r}}\cos(\boldsymbol{\Omega}^\mathsf{T}\tilde{\mathbf{x}}) \\ \frac{\sigma_f}{\sqrt{r}}\sin(\boldsymbol{\Omega}^\mathsf{T}\tilde{\mathbf{x}}) \end{bmatrix}, \tag{4.25}$$

where $\boldsymbol{\Omega} = [\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_r]^\mathsf{T}$. Assuming the prior distribution of feature weights $\mathbf{w} \sim \mathcal{N}(0, \boldsymbol{\Sigma}_p)$, the posterior distribution of $\Delta\mathbf{x}$ can be derived as in the standard Bayesian linear regression

$$\Delta\mathbf{x}|\mathcal{D}, \tilde{\mathbf{x}} \sim \mathcal{N}(\mathbf{w}^\mathsf{T}\boldsymbol{\phi}, \sigma_n^2(1 + \boldsymbol{\phi}^\mathsf{T}\mathbf{A}^{-1}\boldsymbol{\phi})), \tag{4.26}$$

$$\boldsymbol{\phi} = \boldsymbol{\phi}(\tilde{\mathbf{x}}), \; \mathbf{w} = \mathbf{A}^{-1}\boldsymbol{\Phi}\Delta\mathbf{X}, \; \mathbf{A} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma_n^2\boldsymbol{\Sigma}_p^{-1},$$

$$\boldsymbol{\Phi} = [\; \boldsymbol{\phi}(\tilde{\mathbf{x}}_1) \;\; ... \;\; \boldsymbol{\phi}(\tilde{\mathbf{x}}_N) \;].$$

The above formulation has been derived in SSGP regression [91].

The computational complexity becomes $O(Nr^2 + r^3)$, which is significantly more efficient than GPR with $O(N^3)$ time complexity when $r \ll N$. The hyper-parameters are learned by maximizing the log-marginal likelihood of the training outputs given the inputs using numerical methods [47]. To update the weights $\mathbf{w}$ incrementally given a new sample, we do not need to store or invert $\mathbf{A}$ explicitly. Instead, we keep track of its upper triangular Cholesky factor $\mathbf{A} = \mathbf{R}^{\mathsf{T}}\mathbf{R}$ [107]. Given a new sample, a rank-1 update is applied to the Cholesky factor $\mathbf{R}$, which requires $O(r^2)$ time. In contrast, model update in GPs requires $O(N^3)$. For the rest of the section, we use the SE kernel and linear Bayesian regression as in [91]. However, our proposed control framework is not tied to these choices. More precisely, the posterior can be computed by other methods, and other continuous shift-invariant kernels can be used instead of the SE kernel.

Our goal is to generate trajectory rollouts of probability distributions by applying multi-step SSGP inference. When propagating the predictive distributions over a time horizon H, the input state-control pair $\tilde{\mathrm{x}}$ becomes uncertain and we need to compute $\mathrm{p}(\Delta\mathbf{x}) = \int \mathrm{p}(\Delta\mathbf{x}|\tilde{\mathrm{x}})\mathrm{p}(\tilde{\mathrm{x}})\mathrm{d}\tilde{\mathrm{x}}$. As mentioned previously, this predictive distribution cannot be computed analytically. We will compute the Gaussian predictive distribution by linearizing the posterior mean function w.r.t. the input. We skip the details in this chapter and we refer the reader to chapter 5 for prediction under uncertain inputs in SSGPs. Here we assume we have computed the predictive Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_{\mathbf{f}_k}, \boldsymbol{\Sigma}_{\mathbf{f}_k}$ and cross-covariance $\boldsymbol{\Sigma}_{\mathbf{f}_k,\tilde{\mathbf{x}}_k}$. Then we obtain the state space representation of the learned dynamics

$$\begin{aligned} \boldsymbol{\mu}_{k+1} &= \boldsymbol{\mu}_k + \boldsymbol{\mu}_{\mathbf{f}_k}, \\ \boldsymbol{\Sigma}_{k+1} &= \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_{\mathbf{f}_k} + \boldsymbol{\Sigma}_{\tilde{\mathbf{x}}_k,\mathbf{f}_k} + \boldsymbol{\Sigma}_{\mathbf{f}_k,\tilde{\mathbf{x}}_k}. \end{aligned} \tag{4.27}$$

(4.27) is a general representation of dynamical systems. Note that $\boldsymbol{\mu}_{\mathbf{f}_k}, \boldsymbol{\Sigma}_k$ and $\boldsymbol{\Sigma}_{\tilde{\mathbf{x}}_k,\mathbf{f}_k}$ are nonlinear functions of $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. Similar formulation can be found in GP-based reinforcement learning methods such as [50]. Next we derive a stochastic optimal control scheme based on this expression.

4.3.3 A Path Integral Control Approach with Covariance Adaptation

In this section we derive an iterative stochastic optimal control algorithm based on 1) the probabilistic representation of the dynamics in (4.27), 2) the path integral (PI) control theory and 3) covariance matrix adaptation.

Augmentation of the belief dynamics

In order to incorporate uncertainty explicitly in our control framework, we introduce the Gaussian *belief* $\mathbf{v}_k = [\boldsymbol{\mu}_k \ \mathrm{vec}(\boldsymbol{\Sigma}_k)]^{\mathrm{T}}$ which is the predictive distribution over state $\mathbf{x}_k$, where $\mathrm{vec}(\boldsymbol{\Sigma}_k)$ is the vectorization of $\boldsymbol{\Sigma}_k$. Based on eq.(4.27), the belief dynamics model can be written as

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \mathcal{F}(\mathbf{v}_k, \mathbf{u}_k). \tag{4.28}$$

Where $\mathcal{F}$ is the belief transition function obtained from (4.27). Note that the belief dynamics is deterministic. Next we define an auxiliary control variable $\mathbf{u}^a$. The difference between the actual control and auxiliary control $\Delta\mathbf{u}^a = \mathbf{u}^a - \mathbf{u}$ is expressed by the differential equation

$$\frac{\mathrm{d}\Delta\mathbf{u}^a}{\mathrm{d}t} = \mathbf{G}\Delta\mathbf{u}^a. \tag{4.29}$$

Based on the stability theory for linear systems, this system has a constant solution $\Delta\mathbf{u}^a = 0 \in \mathbb{R}^m$. This solution is asymptotically stable as $t \to \infty$ if the real parts of all eigenvalues of the transition matrix $\mathbf{G}$ are negative. In other words, we can design the matrix $\mathbf{G}$ so that $\Delta\mathbf{u}^a \to 0$. In this case the auxiliary control $\mathbf{u}^a$ is approximately equivalent to the actual control $\mathbf{u}$. In other words, optimizing $\mathbf{u}^a$ is approximately equivalent to optimizing $\mathbf{u}$. In order to perform random exploration we add Gaussian noise $\varepsilon \in \mathbb{R}^m$ with $\varepsilon \sim \mathcal{N}(0, \boldsymbol{\Sigma}_\varepsilon)$ to the system (4.29), therefore we have a stochastic dynamical system represented by SDE

$\mathrm{d}\Delta\mathbf{u}^a = \mathbf{G}\Delta\mathbf{u}^a\mathrm{d}t + \mathbf{B}\mathrm{d}\varepsilon$. Its discrete-time representation becomes

$$\Delta\mathbf{u}^a_{k+1} = \Delta\mathbf{u}^a_k + \mathbf{G}\Delta\mathbf{u}^a_k\Delta t + \mathbf{B}\boldsymbol{\varepsilon}_k\sqrt{\Delta t}, \quad \boldsymbol{\varepsilon}_k \sim \mathcal{N}(0, \boldsymbol{\Sigma}_\varepsilon), \tag{4.30}$$

where $\mathbf{B} \in \mathbb{R}^{m\times m}$ is the user-designed diffusion matrix, and $\boldsymbol{\Sigma}_\varepsilon$ is the exploration noise covariance matrix. Based on (4.30) and (4.28) we obtain the augmented belief system

$$\begin{aligned}\begin{pmatrix}\mathbf{v}_{k+1}\\ \Delta\mathbf{u}^a_{k+1}\end{pmatrix} &= \begin{pmatrix}\mathbf{v}_k\\ \Delta\mathbf{u}^a_k\end{pmatrix} + \underbrace{\begin{pmatrix}\mathcal{F}(\mathbf{v}_k, \mathbf{u}_k)/\Delta t\\ -\mathbf{G}\mathbf{u}_k\end{pmatrix}}_{\tilde{\mathbf{f}}(\mathbf{v}_k)}\Delta t\\ &+ \underbrace{\begin{pmatrix}0\\ \mathbf{G}\end{pmatrix}}_{\tilde{\mathbf{G}}}\mathbf{u}^a_k\Delta t + \underbrace{\begin{pmatrix}0\\ \mathbf{B}\end{pmatrix}}_{\tilde{\mathbf{B}}}\boldsymbol{\varepsilon}_k\sqrt{\Delta t}.\end{aligned} \tag{4.31}$$

We can rewrite the above augmented belief dynamics in a concise form

$$\mathbf{t}_{k+1} = \mathbf{t}_k + \tilde{\mathbf{f}}(\mathbf{t}_k)\Delta t + \tilde{\mathbf{G}}\mathbf{u}^a_k\Delta t + \tilde{\mathbf{B}}\boldsymbol{\varepsilon}_k\sqrt{\Delta t}, \tag{4.32}$$

where the augmented belief $\mathbf{t} = [\mathbf{v} \;\; \Delta\mathbf{u}]^\mathsf{T}$. The actual control $\mathbf{u}$ can be viewed as a varying parameter of the new system. The control and diffusion matrices $\mathbf{G}, \mathbf{B}$ can be state (belief) or time dependent. For simplicity of notation we assume they are constant for the rest of this section. The augmented belief dynamics (4.32) is affine in the auxiliary control $\mathbf{u}^a$ as well as the exploration noise $\boldsymbol{\varepsilon}$. For this type of systems, we can apply the path integral control theory [36] to obtain a sampling-based control law.

Stochastic control law via the path integral formulation

Based on (4.32), the conditional probability of $\mathbf{t}_{k+1}$ given $\mathbf{t}_k$ and $\mathbf{u}_k$ is normally distributed. More precisely we have

$$\mathrm{p}(\mathbf{t}_{k+1}) = \mathcal{N}(\mathbf{v}_k + \tilde{\mathbf{f}}(\mathbf{t}_k)\Delta t + \tilde{\mathbf{G}}\mathbf{u}_k^a\Delta t, \tilde{\mathbf{B}}\boldsymbol{\Sigma}_\varepsilon\tilde{\mathbf{B}}^{\mathsf{T}}\Delta t). \tag{4.33}$$

Given the Bellman equation in (4.22), the Hamilton-Jacobi-Bellman (HJB) equation in continuous-time is obtained by approximating the integral for a small Δt and applying Itô's rule:

$$\begin{aligned} -\partial_t V_k = &\min_{\mathbf{u}_k^a}(\tilde{q}_k + \frac{1}{2}\mathbf{u}_k^{a\mathrm{T}}\mathbf{R}\mathbf{u}_k^a + (\tilde{\mathbf{f}}_k + \tilde{\mathbf{G}}\mathbf{u}_k)^{\mathrm{T}}\nabla_{\mathbf{x}}\mathbf{V}_k \\ &+\frac{1}{2}\,\mathbf{Tr}(\tilde{\mathbf{B}}\boldsymbol{\Sigma}_\varepsilon\tilde{\mathbf{B}}^{\mathsf{T}}\nabla_{\mathbf{xx}}\mathbf{V}_k)), \end{aligned} \tag{4.34}$$

where $\tilde{\mathbf{f}}_k = \tilde{\mathbf{f}}(\mathbf{t}_k)$, $\mathbf{V}_k = \mathbf{V}(\mathbf{t}_k, k)$ is the value function, and $\tilde{q}_k$ is the augmented state cost, which can be defined as

$$\tilde{q}_k = \mathbb{E}_{\mathbf{x}}[q_k] + (\Delta\mathbf{u}^a)^{\mathrm{T}}\mathbf{R}\Delta\mathbf{u}^a.$$

Note that this cost penalizes both the expectation of the state cost and the difference between the auxiliary control and the actual control. This cost is risk-neutral, and a risk-sensitive criterion can be incorporated by adding the cost variance term $\mathbb{VAR}_{\mathbf{x}}[q_k]$. Now we seek the optimal auxiliary control $\mathbf{u}^a$ for the augmented system, which is an approximation of the actual optimal control for the original system. To find the optimal control, we take the gradient of the above expression (inside the parentheses) with respect to $\mathbf{u}_k$ and set to 0. This yields $\mathbf{u}_k = -\mathbf{R}^{-1}\tilde{\mathbf{G}}^{\mathrm{T}}\nabla_{\mathbf{x}}\mathbf{V}_t$. Inserting this expression into the HJB equation yields the following nonlinear and second order PDE

$$\begin{aligned} -\partial_t V_k = &\tilde{q}_k + (\nabla_{\mathbf{x}}\mathbf{V}_k)^{\mathrm{T}}\tilde{\mathbf{f}}_k - \frac{1}{2}(\nabla_{\mathbf{x}}\mathbf{V}_k)^{\mathrm{T}}\tilde{\mathbf{G}}\mathbf{R}^{-1}\tilde{\mathbf{G}}^{\mathrm{T}}\nabla_{\mathbf{x}}\mathbf{V}_k \\ &+ \frac{1}{2}\,\mathbf{Tr}(\tilde{\mathbf{B}}\boldsymbol{\Sigma}_\varepsilon\tilde{\mathbf{B}}^{\mathsf{T}}\nabla_{\mathbf{xx}}\mathbf{V}_k). \end{aligned}$$

In order to solve the above PDE we use the exponential transformation of the value function $V_k = -\lambda \log \Psi_t$, where $\Psi_t = \Psi(\mathbf{t}_k)$ is called the *desirability* of $\mathbf{t}_k$. Inserting the partial derivative terms to the above equation results in

$$\begin{aligned} \frac{\lambda}{\Psi_k}\partial_t \Psi_k = \tilde{q}_k - \frac{\lambda}{\Psi_k}(\nabla_{\mathbf{x}}\Psi_k)^{\mathrm{T}}\tilde{\mathbf{f}}_k - \frac{\lambda^2}{2\Psi_t^2}(\nabla_{\mathbf{x}}\Psi_t)^{\mathrm{T}}\tilde{\mathbf{G}}\mathbf{R}^{-1}\tilde{\mathbf{G}}^{\mathrm{T}}\nabla_{\mathbf{x}}\Psi_k \\ +\frac{\lambda}{2\Psi_k^2}\mathbf{Tr}((\nabla_{\mathbf{x}}\Psi_k)^{\mathrm{T}}\tilde{\mathbf{B}}\mathbf{\Sigma}_\varepsilon\tilde{\mathbf{B}}^{\mathsf{T}}\nabla_{\mathbf{x}}\Psi_k) - \frac{\lambda}{2\Psi_t}\mathbf{Tr}(\nabla_{\mathbf{xx}}\Psi_k\tilde{\mathbf{B}}\mathbf{\Sigma}_\varepsilon\tilde{\mathbf{B}}^{\mathsf{T}}). \end{aligned} \tag{4.35}$$

The quadratic terms $\nabla_{\mathbf{x}}\Psi_k$ will cancel out under the assumption of $\lambda\tilde{\mathbf{G}}\mathbf{R}^{-1}\tilde{\mathbf{G}}^{\mathrm{T}} = \tilde{\mathbf{B}}\mathbf{\Sigma}_\varepsilon\tilde{\mathbf{B}}^{\mathsf{T}}$. This assumption was first proposed in [34]. It is straightforward to show that this is equivalent to $\lambda\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}} = \mathbf{B}\mathbf{\Sigma}_\varepsilon\mathbf{B}^{\mathsf{T}}$. This assumption can be easily satisfied in our case since $\mathbf{G}$ and $\mathbf{B}$ are designed by the user. Given this assumption, the above PDE is simplified as

$$\partial_t \Psi_k = \frac{1}{\lambda}\tilde{q}_k\Psi_k - \tilde{\mathbf{f}}_k^{\mathrm{T}}\nabla_{\mathbf{x}}\Psi_k - \frac{1}{2}\mathbf{Tr}(\nabla_{\mathbf{xx}}\Psi_k\tilde{\mathbf{B}}\mathbf{\Sigma}_\varepsilon\tilde{\mathbf{B}}^{\mathsf{T}}), \tag{4.36}$$

subject to the terminal condition $\Psi_H = \exp(-\frac{1}{\lambda}\tilde{q}_H)$. The resulting Chapman-Kolmogorov PDE (4.36) is linear. In general, solving (4.36) analytically is intractable for nonlinear systems and cost functions. Following the path integral control theory, we apply the Feynman-Kac formula which gives a probabilistic representation of the solution to the linear PDE (4.36)

$$\Psi_k = \int \mathrm{p}(\tau_k|\mathbf{t}_k)\exp\big(-\frac{1}{\lambda}(\sum_{j=k}^{H}\tilde{q}_j\Delta t)\big)\Psi_H \mathrm{d}\tau_k, \tag{4.37}$$

where τ_k is the belief trajectory from time k to H. Now we define the path cost

$$S(\tau_k|\mathbf{t}_k) = \tilde{q}_H + \sum_{j=k}^{H}\tilde{q}_j\Delta t. \tag{4.38}$$

The optimal control can be obtained as (derivation is omitted)

$$\mathbf{u}_k^a = \int P(\tau_k|\mathbf{t}_k)\mathbf{u}_L(\tau_k)\mathrm{d}\tau, \tag{4.39}$$

where the probability of trajectory is defined as

$$P(\tau_k|\mathbf{t}_k) = \frac{\exp(-\frac{1}{\lambda}S(\tau_k|\mathbf{t}_k)}{\int \exp(-\frac{1}{\lambda}S(\tau_k|\mathbf{t}_k)\mathrm{d}\tau}, \tag{4.40}$$

and the local control for each sample is

$$\mathbf{u}_L(\tau_k) = \mathbf{R}^{-1}\mathbf{G}^{\mathsf{T}}(\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathsf{T}})^{-1}\mathbf{B}\boldsymbol{\varepsilon}_k. \tag{4.41}$$

Note that the optimal control (4.39) is a weighted average of local controls which depend on the exploration noises $\boldsymbol{\varepsilon}$. This probability-weighted averaging scheme appears in many path integral control related algorithms, e.g., [34, 35, 36, 43, 39, 40]. Our method differs from these methods because we incorporate predictive uncertainty into sampling and averaging, e.g., highly uncertain samples have low probabilities. The control law is based on sampling from uncontrolled process. This exploration scheme is inefficient for high dimensional or highly nonlinear systems since uncontrolled trajectories usually have low probabilities (in high cost region) [34]. In the next section we present an efficient iterative scheme with covariance adaptation.

Iterative control with covariance adaptation

In order to improve sample efficiency, an importance sampling scheme based on controlled process has been proposed in [34] and an iterative algorithm has been derived in [99]. The iterative control law can be expressed as

$$\mathbf{u}_k^{a,new} = \mathbf{u}_k^{a,old} + \delta\mathbf{u}_k, \tag{4.42}$$

where the control correction term and the probability of trajectory are defined as

$$\begin{aligned} \delta \mathbf{u}_k &= \int \tilde{P}(\tau_k|\mathbf{t}_k)\mathbf{u}_L(\tau_k)\mathrm{d}\tau_k, \\ \tilde{P}(\tau_k|\mathbf{t}_k) &= \frac{\exp(-\frac{1}{\lambda}\tilde{S}(\tau_k|\mathbf{t}_k)}{\int \exp(-\frac{1}{\lambda}\tilde{S}(\tau_k|\mathbf{t}_k)\mathrm{d}\tau_k}, \end{aligned} \tag{4.43}$$

where

$$\tilde{S}(\tau_k|\mathbf{t}_k) = S(\tau_k|\mathbf{t}_k) + \sum_{j=k}^{H} \Big((\mathbf{u}_j^a)^{\mathsf{T}} \mathbf{u}_j^a \Delta t + 2(\mathbf{u}_j^a)^{\mathsf{T}} \boldsymbol{\varepsilon}_j \Big)$$

is the new path cost which has a coupling term between the control and noise added to the original path cost $S(\tau_k|\mathbf{t}_k)$ [99]. The derivation is omitted. In the finite sample case we can rewrite the control law as

$$\begin{aligned} \mathbf{u}_k^{a,new} &= \mathbf{u}_k^{a,old} + \sum_{i=1}^{N} \tilde{P}_i(\tau_k|\mathbf{t}_k)\mathbf{u}_{L,i}(\tau_k), \\ \tilde{P}_i(\tau_k|\mathbf{t}_k) &= \frac{\exp(-\frac{1}{\lambda}\tilde{S}_i(\tau_k|\mathbf{t}_k)}{\sum_{i=1}^{N} \exp(-\frac{1}{\lambda}\tilde{S}_i(\tau_k|\mathbf{t}_k)\mathrm{d}\tau_k}, \end{aligned} \tag{4.44}$$

where N is the total number of samples. Intuitively, the local control associated with each sample is weighted by the probability of the path. And the optimal control update is the probability-weighted average of the local controls. The iterative scheme improves the scalability of the batch model PI control [109]. However, the exploration noise has a constant covariance during learning and the trade-off between exploration and exploitation is omitted.

Now we propose a method for updating the exploration noise covariance matrix via probability-weighted averaging. The basic idea is similar to the Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [110]. More precisely after each control update, we update the covariance of the exploration noise

$$\boldsymbol{\Sigma}_{\varepsilon,k}^{new} = \sum_{i=1}^{N} \tilde{P}_i(\tau_k|\mathbf{t}_k)(\boldsymbol{\varepsilon}_{k,i} - \bar{\boldsymbol{\varepsilon}}_k)(\boldsymbol{\varepsilon}_{k,i} - \bar{\boldsymbol{\varepsilon}}_k)^{\mathsf{T}}, \tag{4.45}$$

where $\bar{\varepsilon}_k$ is the mean of sampled ε_k. W.l.o.g. we use zero-mean $\bar{\varepsilon}_k = 0$. By implementing this scheme, we not only update the direction of exploration via the iterative PI control law (4.44), but also the magnitude of the exploration via covariance adaptation (4.45). The covariance adaptation scheme has shown significant improvement in terms of convergence speed in policy search [111]. More technical details and interpretations can be found in [110]. However, our covariance adaptation method is different from previous related works such as [111]. In our case the noise covariance is time-dependent, i.e., $\mathbf{\Sigma}_{\varepsilon,k}^{new}$ depends on the time step k. More precisely, the exploration magnitude is different and updated at each time step along trajectories. At the next iteration, the exploration noise $\varepsilon_1, ..., \varepsilon_H$ will be sampled from the updated distribution $\mathcal{N}(\bar{\varepsilon}_1, \mathbf{\Sigma}_{\varepsilon,1}^{new}), ..., \mathcal{N}(\bar{\varepsilon}_H, \mathbf{\Sigma}_{\varepsilon,H}^{new})$. In contrast, [111] uses the same noise covariance at each time step to generate trajectories. An algorithm for episodic reinforcement learning (RL) is summarized in **Algorithm** 4. This algorithm can be easily extended to perform model predictive control (MPC).

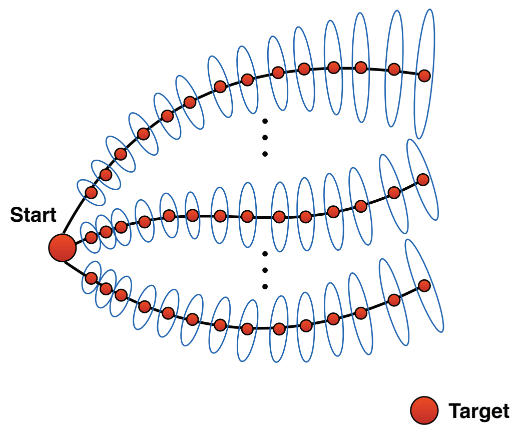
4.3.4 Relation to Existing Works

In the following we summarize the novel features of the proposed framework. Our method is derived from the original PI and iterative PI control paradigm [34, 35, 36, 99]. It differs form these works in three major ways: 1) these methods require full knowledge of the dynamics model. In contrast, our method is data-driven. 2) Our method performs sampling in belief space and each sample takes into account model uncertainty. 3) Our method uses covariance adaptation in probability-weighted averaging, which lead to significantly improved convergence speed. In contrast [34, 35, 36, 99] keep a fixed noise covariance. Compared with PI-based policy search approaches [39, 40, 43], our method does not require policy parameterization or demonstration for policy initialization. In particular, the covariance adaptation scheme has been used in [40] for policy search. Our method differs from [40] because 1) our method is probabilistic model-based, while [40] is model-free. The benefits of probabilistic model-based method was discussed in [55]. 2) Our control

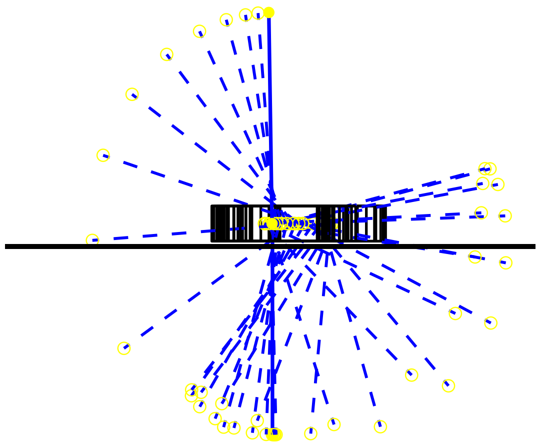
Algorithm 4 Path Integral Control with Covariance Adaptation under Unknown Dynamics

```
1: Given: The number of sample $N$, number of random feature $r$, parameters $\mathbf{B}$, $\mathbf{G}$ and $\mathbf{R}$
2: Initialization: Collect data from the physical model (4.20) using random controls.
3: Model learning: Train GP hyperparameters. Sample random features and compute their weights (sec.4.3.2).
4: repeat
5:     for k = 1:H do
6:         repeat
7:             Sample: Generate $N$ trajectories in the augmented belief space $\mathbf{t}_{k:H,1}, ..., \mathbf{t}_{k:H,N}$ using
               exploration noises $\boldsymbol{\varepsilon}_{k:H,1}, ..., \boldsymbol{\varepsilon}_{k:H,N}$ sampled from $\mathcal{N}(0, \boldsymbol{\Sigma}_{\varepsilon,k:H})$.
8:             Evaluate: Compute trajectory cost for each sample $\tilde{S}_1(\tau|\mathbf{t}_{k:H}), ..., \tilde{S}_N(\tau|\mathbf{t}_{k:H})$.
9:             Probability: Compute probability for each sample $\tilde{P}_1(\tau|\mathbf{t}_{k:H}), ..., \tilde{P}_N(\tau|\mathbf{t}_{k:H})$, see (4.44).
10:            Control update: Update control via probability-weighted averaging $\mathbf{u}^{a,new}_{k:H} = \mathbf{u}^{a,old}_{k:H} + \delta\mathbf{u}_{k:H}$, see (4.44).
11:            Covariance adaptation: Update exploration noise covariance matrices $\boldsymbol{\Sigma}_{\varepsilon,k:H} = \sum_{i=1}^{N} \tilde{P}_i(\tau|\mathbf{t}_{k:H})$
               $(\boldsymbol{\varepsilon}_{k:H,i} - \bar{\boldsymbol{\varepsilon}}_{k:H})(\boldsymbol{\varepsilon}_{k:H,i} - \bar{\boldsymbol{\varepsilon}}_{k:H})^{\mathsf{T}}$.
12:        until Convergence
13:        Execution: Apply $\mathbf{u}^{a,new}_{k}$ to the original system (4.20). Move one step forward.
           Incorporate new data
           and update model (see section 4.3.2).
14:    end for
15: until Task learned
16: return Optimal control sequence.
```

policy is time-varing, in contrast to time-invariant parameters in [40]. Recently a gradient-based PI control method based on GPs was introduced in [44]. This method is sample efficient but requires partial model knowledge and the system has to be affine in control. In contrast, our method has no such assumption. Furthermore, our method takes advantage of fast probabilistic inference in SSGPs which is much more computationally efficient than full GP inference. Another key aspect of the proposed approach is that computation for probabilistic inference can be done in parallel (see fig. 4.3a). Real-time implementation requires a Graphic Processing Unit (GPU).



(a)



(b)

Figure 4.3: (a) Sampling of belief trajectories via probabilistic inference. The solid line and error ellipse represent predictive mean and variance of the state, respectively. The computation of probabilistic inference can be distributed in parallel. (b) Cart-pole swing up task.

4.3.5 Experiments and Analysis

In this section we evaluate the performance of the proposed framework using a simulated cart-pole swing up task, see fig. 4.3b. The cart-pole is an under-actuated systems consist of a cart and a single pendulum (4 states and 1 control). The tasks is to swing-up the pendulum from the initial position (point down). We consider a quadratic state cost function

Method	Optimized average optimized total cost	Samples from the true model	Samples from the learned model	Number of iteration per time step
Our method	3.57×10^4	8 total	300 per iteration	20
Iterative PI [109]	3.62×10^4	300 per iteration	N/A	20

Table 4.2: Comparison between our method under unknown dynamics and the iterative PI control with known dynamics model for the cart-pole swing up task.

$q(\mathbf{x}) = (\mathbf{x} - \mathbf{x}^{target})^{\mathsf{T}} Q(\mathbf{x} - \mathbf{x}^{target})$, and therefore

$$\mathbb{E}_{\mathbf{x}}[q(\mathbf{x})] = (\boldsymbol{\mu} - \mathbf{x}^{target})^{\mathsf{T}} Q(\boldsymbol{\mu} - \mathbf{x}^{target}) + \mathbf{Tr}(\boldsymbol{\Sigma} Q),$$

where $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. We initialized our algorithm by 8 trajectories sampled from the true dynamics model using random controls. In order to demonstrate the effect of covariance adaptation, we also implemented our algorithm with constant exploration noise covariance. We performed 5 independent experiments for both cases and the comparison is shown in fig.4.4. Our covariance adaptation scheme significantly improves the convergence performance. This is because exploration magnitude decreased quickly after a few iterations. The trajectory costs by executing the optimized controls are shown in fig. 4.5. Table 4.2 compares the performance of our method and the iterative PI controller [109] with known dynamics model. For both methods we used 300 samples and performed 20 iterations per time step during optimization. Note that our method only uses 8 samples from the true systems for model learning, and the 300 samples were generated from the learned SSGP model. Our data-driven method performed well in terms of cost reduction. The iterative PI controller generally requires more samples and iterations to achieve optimal performance. In contrast, our method is more efficient and robust to modeling errors because 1) each sample is generated by probabilistic inference and weighted by its predictive uncertainty, 2) covariance adaptation improves convergence speed.

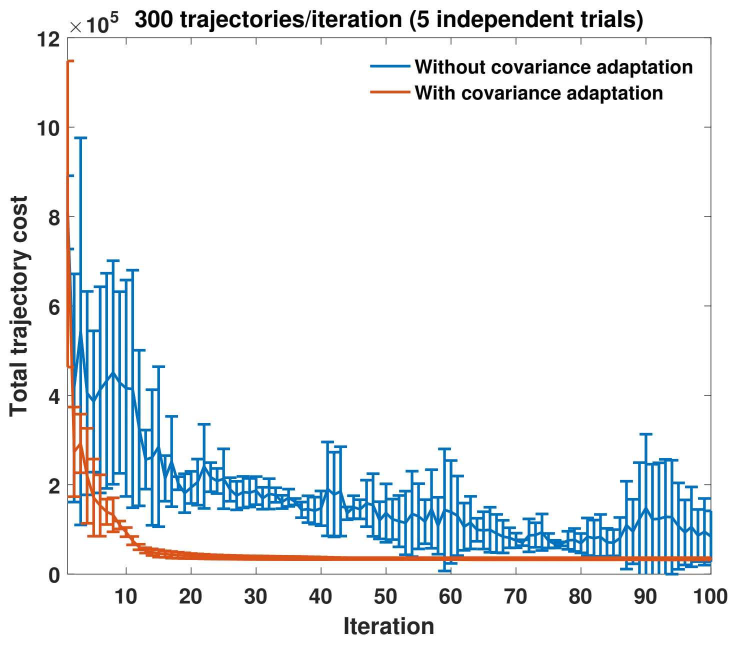


Figure 4.4: Comparison in terms of total trajectory cost reduction at each iteration during optimization.

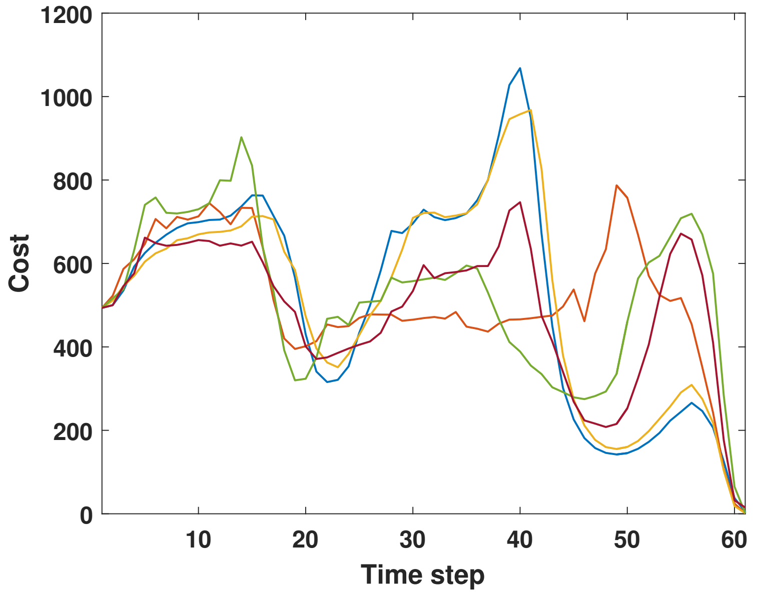


Figure 4.5: Trajetory costs collected by executing optimized controls on the true dynamics model (5 independent trials).

4.3.6 Summary and Discussion

We have introduced a data-driven stochastic control method based on the path integral (PI) control formulation and probabilistic inference. Similar to related PI control methods [34, 43, 39, 40, 109], we perform sampling and probability-weighted averaging to obtain optimal controls. In contrast to these methods, our method does not rely on model knowledge or policy parameterization. A key feature of our method is that sampling is performed in the belief space and each sample takes into account model uncertainty. In addition, we leverage covariance adaptation in order to tune the exploration magnitude automatically. We presnted a numerical example demonstrating that 1) our method achieves good performance in terms of cost reduction using much less samples from the true system than [109], 2) our covariance adaptation scheme improves convergence speed. Our future work will focus on GPU implementation of our method and applications in robotics.

CHAPTER 5
PREDICTION UNDER UNCERTAINTY IN SPARSE SPECTRUM GAUSSIAN PROCESSES

Sparse Spectrum Gaussian Processes (SSGPs) are a powerful tool for scaling Gaussian processes (GPs) to large datasets. Existing SSGP algorithms for regression assume deterministic inputs, precluding their use in many real-world robotics and engineering applications where accounting for input uncertainty is crucial. We address this problem by proposing two analytic moment-based approaches with closed-form expressions for SSGP regression with uncertain inputs. Our methods are more general and scalable than their standard GP counterparts, and are naturally applicable to multi-step prediction or uncertainty propagation. We show that efficient algorithms for Bayesian filtering and stochastic model predictive control can use these methods, and we evaluate our algorithms with comparative analyses and both real-world and simulated experiments.

5.1 Introduction

The problem of *prediction under uncertainty*, appears in many fields of science and engineering that involve sequential prediction including state estimation [112, 113], time series prediction [49], stochastic process approximation [114], and planning and control [50, 44]. In these problems, uncertainty can be found in both the predictive models and the model's inputs. Formally, we are often interested in finding the probability density of a prediction y, given a distribution $p(x)$ and a probabilistic model $p(y|x)$. By marginalization,

$$p(y) = \int p(y|x)p(x)\,\mathrm{d}x. \tag{5.1}$$

Unfortunately, computing this integral exactly is often intractable. In this chapter, we tackle a subfamily of (5.1) where: 1) the probabilistic model is learned from data and specified by a sparse spectrum representation of a Gaussian process (SSGP); and 2) the input x is normally distributed. We show that analytic expressions of the moments of $p(y)$ can be derived and that these are directly applicable to sequential prediction problems like filtering and control.

Related work

Gaussian Process (GP) regression with uncertain inputs has been addressed by [48, 49], and extended to the multivariate outputs by [69]. These methods have led to the development of many algorithms in reinforcement learning [60, 50], Bayesian filtering [112, 70], and smoothing [113]. However, these approaches have two major limitations: 1) they are not directly applicable to large datasets, due to the polynomial time complexity for exact inference [47]; and 2) analytic moment expressions, when used, are restricted to squared exponential (SE) kernels [69] and cannot be generalized to other kernels in a straightforward way.

A common method for approximating large-scale kernel machines is through random Fourier features [106]. The key idea is to map the input to a low-dimensional feature space yielding fast linear methods. In the context of GP regression (GPR), this idea leads to the sparse spectrum GPR (SSGPR) algorithm [91]. SSGP has been extended in a number of ways for, e.g. incremental model learning [107], and large-scale GPR [115, 116]. However, to the best of our knowledge, prediction under uncertainty for SSGPs has not been explored. Although there are several alternative approximations to exact GP inference including approximating the posterior distribution using inducing points, *e.g.*, [84, 117, 118], comparing different GP approximations is not the focus of this chapter.

Applications

We consider two key problems that are widely encountered in robotics and engineering: Bayesian filtering and stochastic model predictive control.

The goal of Bayesian filtering is to infer a hidden system state through the recursive application of Bayes' rule. Well-known frameworks for Bayesian filtering include unscented Kalman Filtering (UKF), particle filtering (PF), extended Kalman filtering (EKF), and assumed density filtering (ADF). GP-based Bayesian filtering with SE kernels has been developed for these frameworks by [112, 70]. We extend this work with highly efficient SSGP-based EKF and ADF algorithms.

The goal of stochastic model predictive control (MPC) is to find finite horizon optimal control at each time instant. Due to the high computational cost of GP inference and real-time optimization requirements in MPC, most GP-based control methods [50, 85, 119] are restricted to episodic reinforcement learning tasks. To cope with this challenge, we present an SSGP-based MPC algorithm that is fast enough to perform probabilistic trajectory optimization and model adaptation on-the-fly.

Our contributions

- We propose two approaches to prediction under uncertainty in SSGPs with closed-form expressions for the predictive distribution. Compared to previous GP counterparts, our methods: 1) are more scalable, and 2) can be generalized to any continuous shift-invariant kernels with a Fourier feature representation.

- We demonstrate successful applications of the proposed approaches by presenting scalable algorithms for 1) recursive Bayesian filtering and 2) stochastic model predictive control via probabilistic trajectory optimization.

The rest of the chapter is organized as follows. In §5.2, we give an introduction to SSGPs, which serves as our probabilistic model. Derivation and expressions of the two proposed prediction methods are detailed in §5.3. Applications to filtering and control, and experimental results are presented in §5.4 and §5.5 respectively. Finally §5.7 concludes the chapter.

5.2 Sparse Spectral Representation of GPs

Consider the task of learning the function $f : \mathbf{R}^d \to \mathbf{R}$, given IID data $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$, with each pair related by

$$y = f(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2), \tag{5.2}$$

where ϵ is IID additive Gaussian noise. Gaussian process regression (GPR) is a principled way of performing Bayesian inference in function space, assuming that function f has a prior distribution $f \sim \mathcal{GP}(m, k)$, with mean function $m : \mathbf{R}^d \to \mathbf{R}$ and kernel $k : \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$. Without loss of generality, we assume $m(x) = 0$. Exact GPR is challenging for large datasets due to its $O(n^3)$ time and $O(n^2)$ space complexity [47], which is a direct consequence of having to store and invert an $n \times n$ Gram matrix.

Random features can be used to form an unbiased approximation of continuous shift-invariant kernel functions, and are proposed as a general mechanism to accelerate large-scale kernel machines [106], via explicitly mapping inputs to low-dimensional feature space. Based on Bochner's theorem, the Fourier transform of a continuous shift-invariant positive definite kernel $k(x, x')$ is a proper probability distribution $p(\omega)$, assuming $k(x, x')$ is properly scaled [106]:

$$\begin{aligned} k(x, x') &= \int p(\omega) e^{j\omega^T (x - x')} \, \mathrm{d}\omega \\ &= \mathbb{E}(\phi_\omega(x) \phi_\omega(x')^*), \quad \omega \sim p(\omega), \end{aligned} \tag{5.3}$$

where $\phi_\omega(x) = e^{j\omega^T x}$, and we can see that $k(x, x')$ only depends on the lag vector separating x and x': $x - x'$. Equation (5.3) leads to an unbiased finite sample approximation of

k: $k(x, x') \approx \frac{1}{m} \sum \phi_{\omega_i}(x)\phi_{\omega_i}(x')^*$, where random frequencies $\{\omega_i\}_{i=1}^m$ are drawn IID from $p(\omega)$. Utilizing the fact that ϕ_ω can be replaced by sinusoidal functions since both $p(\omega)$ and $k(x, x')$ are reals, and concatenating features $\{\phi_{\omega_i}\}_{i=1}^m$ into a succinct vector form, an approximation for $k(x, x')$ is expressed as

$$k(x, x') \approx \phi(x)^T \phi(x'),\ \phi(x) = \begin{bmatrix} \phi^c(x) \\ \phi^s(x) \end{bmatrix}, \tag{5.4}$$
$$\phi_i^c(x) = \sigma_k \cos(\omega_i^T x),\ \phi_i^s(x) = \sigma_k \sin(\omega_i^T x),\ \omega_i \sim p(\omega),$$

where σ_k is a scaling coefficient. For the commonly used Squared Exponential (SE) kernel: $k(x, x') = \sigma_f^2 \exp(-\frac{1}{2}\|x - x'\|_{\Lambda^{-1}}^2)$, $p(\omega) = \mathcal{N}(0, \Lambda^{-1})$ and $\sigma_k = \frac{\sigma_f}{\sqrt{m}}$, where the coefficient σ_f and the diagonal matrix Λ are the hyperparameters, examples of kernels and corresponding spectral densities can be found in Table 5.1.

In accordance with this feature map (5.4), Sparse Spectrum GPs are defined as follows

Definition 2. *Sparse Spectrum GPs (SSGPs) are GPs with kernels defined on the finite-dimensional and randomized feature map ϕ (5.4):*

$$k(x, x') = \phi(x)^T \phi(x') + \sigma_n^2 \delta(x - x'), \tag{5.5}$$

where the function δ is the Kronecker delta function.

The second term in (5.5) accounts for the additive zero mean Gaussian noise in (5.2), if the goal is to learn the correlation between x and y directly as in our case of learning the probabilistic model $p(y|x)$, instead of learning the latent function f.

Because of the explicit finite-dimensional feature map (5.4), each SSGP is equivalent to a Gaussian distribution over the weights of features $w \in \mathbf{R}^{2m}$. Assuming that prior distribution of weights w is $\mathcal{N}(0, I)$ [1] and the feature map is fixed, after conditioning on

[1]I is the identity matrix with proper size. The prior covariance is identity since $\mathbb{E}(f(x)f(x)) =$

the data $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$, the posterior distribution of w is [2]

$$w \sim \mathcal{N}(\alpha,\ \sigma_n^2 A^{-1}), \tag{5.6}$$
$$\alpha = A^{-1}\Phi Y, \quad A = \Phi\Phi^T + \sigma_n^2 I,$$

which can be derived through Bayesian linear regression. In (5.6), the column vector Y and the matrix Φ are specified by the data $\mathcal{D}$: $Y = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^T$, $\Phi = \begin{bmatrix} \phi(x_1) & \dots & \phi(x_n) \end{bmatrix}$. Consequently, the posterior distribution over the output y in (5.2) at a test point x is *exactly Gaussian*, in which the posterior variance explicitly captures the model uncertainty in prediction with input x:

$$p(y|x) = \mathcal{N}(\alpha^T\phi(x),\ \sigma_n^2 + \sigma_n^2\|\phi(x)\|^2_{A^{-1}}). \tag{5.7}$$

This Bayesian linear regression method for SSGP is proposed in [91]. Its time complexity is $O(nm^2 + m^3)$, which is significantly more efficient than standard GPR's $O(n^3)$ when $m \ll n$.

Remark It's worth noting that the methods proposed in this chapter are not tied to specific algorithms for SSGP regression such as Bayesian linear regression [91], but able to account for any SSGP with specified feature weights distribution (5.6), where posterior α and A can be computed by any means. Variations on A include sparse approximations by a low rank plus diagonal matrix, or iterative solutions by optimization methods like doubly stochastic gradient descent [115].

$\mathbb{E}\left(\phi(x)^T ww^T \phi(x')\right) = \phi(x)^T\,\mathbb{E}(ww^T)\phi(x')$, and $\mathbb{E}\left(f(x)f(x')\right) = \phi(x)^T\phi(x')$ (see §2.2 in [60] for details.)

[2]Conditioning on data $\mathcal{D}$ is omitted, *e.g.*, in $w|\mathcal{D}$, for simplicity in notation.

5.3 Prediction under Uncertainty

Two methods for prediction under uncertainty are presented under two conditions: 1) the uncertain input is normally distributed: $x \sim \mathcal{N}(\mu, \Sigma)$, and 2) probabilistic models are in the form of (5.7) specified by SSGPs. Despite these conditions, evaluating the integral in (5.1) is still intractable. In this work, we approximate the true predictive distribution $p(y)$ by a Gaussian distribution with moments that are analytically computed through: 1) exact moment matching, and 2) linearization of posterior mean function. Closed-form expressions for predictive mean, variance, covariance, and input-prediction cross-covariance are derived. We consider multivariate outputs by utilizing conditionally independent scalar models for each output dimension, *i.e.*, assuming for outputs in different dimension y_a and y_b, $p(y_a, y_b|x) = p(y_a|x)p(y_b|x)$. For notational simplicity, we suppress the dependency of $\phi(x)$ on x, and treat y as a scalar by default.

5.3.1 Exact moment matching (SSGP-EMM)

We derive the closed-form expressions for exact moments: 1) the predictive mean $\mathbb{E}\, y$, 2) the predictive variance $\mathbb{VAR} y$ and covariance $\mathbb{COV}(y_a, y_b)$, which in the multivariate case correspond to the diagonal and off-diagonal entries of the predictive covariance matrix, and 3) the cross-covariance between input and prediction $\mathbb{COV}(x, y)$.

Using the expressions for SSGP (5.4), (5.7), and the law of total expectation, the predictive mean becomes

$$\mathbb{E}\, y = \mathbb{E}\,\mathbb{E}(y|x) = \mathbb{E}\left(\alpha^T \phi\right) = \alpha^T\, \mathbb{E} \begin{bmatrix} \phi^c \\ \phi^s \end{bmatrix}, \tag{5.8}$$

$$\mathbb{E}\, \phi_i^c = \sigma_k\, \mathbb{E} \cos(\omega_i^T x), \quad \mathbb{E}\, \phi_i^s = \sigma_k\, \mathbb{E} \sin(\omega_i^T x),$$

where $i = 1, \ldots, m$, and in the nested expectation $\mathbb{E}\,\mathbb{E}(y|x)$, the outer expectation is over the input distribution $p(x) = \mathcal{N}(\mu, \Sigma)$, and the inner expectation is over the conditional distribution $p(y|x)$ (5.7).

By observing (5.8), we see that the expectation of sinusoids under the Gaussian distribution is the key to computing the predictive mean. Thus, we state the following proposition:

Proposition 1. *The expectation of sinusoids over multivariate Gaussian distributions: $x \sim \mathcal{N}(\mu, \Sigma)$, $x \in \mathbf{R}^d$, i.e., $p(x) = (2\pi)^{-\frac{d}{2}}(\det \Sigma)^{-\frac{1}{2}} \exp(-\frac{1}{2}\|x-\mu\|^2_{\Sigma^{-1}})$, can be computed analytically:*

$$\begin{aligned}\mathbb{E}\cos(\omega^T x) &= \exp(-\frac{1}{2}\|\omega\|^2_\Sigma)\cos(\omega^T\mu),\\ \mathbb{E}\sin(\omega^T x) &= \exp(-\frac{1}{2}\|\omega\|^2_\Sigma)\sin(\omega^T\mu).\end{aligned}$$

To prove it, we invoke Euler's formula to transform the left-hand-side to complex domain, apply identities involving quadratic exponentials, and then convert back to real numbers. In Proposition 1, the expectations depend on the mean and variance of the input Gaussian distribution. Intuitively, after passing a Gaussian distributed input through a sinusoidal function, the expectation of the output is equal to passing the mean of the input through the sinusoid, and then scaling it by a constant $\exp(-\frac{1}{2}\|\omega\|^2_\Sigma)$, which depends on the variance of the input. Expectations are smaller with larger input variance due to the periodicity of sinusoids.

The exact moments are then derived using Proposition 1. By the law of total variance, the predictive variance is

$$\begin{aligned}\mathbb{VAR} y &= \mathbb{E}\,\mathbb{VAR}(y|x) + \mathbb{VAR}\,\mathbb{E}(y|x)\\ &= \sigma_n^2 + \sigma_n^2 \,\mathbf{Tr}\left(A^{-1}\Psi\right) + \alpha^T\Psi\alpha - (\mathbb{E}\,y)^2,\end{aligned} \tag{5.9}$$

where Ψ is defined as the expectation of the outer product of feature vectors over input distribution $p(x)$. Specifically, we compute Ψ by applying the product-to-sum trigonometric identities:

$$\begin{aligned}
\mathbb{E}\left(\phi\phi^T\right) = \Psi &= \begin{bmatrix} \Psi^{cc} & \Psi^{cs} \\ \Psi^{sc} & \Psi^{ss} \end{bmatrix}, \\
\Psi_{ij}^{cc} &= \frac{\sigma_k^2}{2}\left(\mathbb{E}\left(\cos(\omega_i+\omega_j)^T x\right) + \mathbb{E}\left(\cos(\omega_i-\omega_j)^T x\right)\right), \\
\Psi_{ij}^{ss} &= \frac{\sigma_k^2}{2}\left(\mathbb{E}\left(\cos(\omega_i-\omega_j)^T x\right) - \mathbb{E}\left(\cos(\omega_i+\omega_j)^T x\right)\right), \\
\Psi_{ij}^{cs} &= \frac{\sigma_k^2}{2}\left(\mathbb{E}\left(\sin(\omega_i+\omega_j)^T x\right) - \mathbb{E}\left(\sin(\omega_i-\omega_j)^T x\right)\right),
\end{aligned}$$

where Ψ^{cc}, Ψ^{ss}, Ψ^{cs} are $m \times m$ matrices, and $i, j = 1, \ldots, m$, on whose terms Proposition 1 can be directly applied.

Next, we derive the covariance for different output dimensions for multivariate prediction. These correspond to the off-diagonal entries of the predictive covariance matrix. We show that, despite the conditional independence assumption for different outputs given a deterministic input, outputs become coupled with uncertain inputs. Using the law of total covariance, the covariance is

$$\begin{aligned}
\mathbb{COV}(y_a, y_b) &= \mathbb{COV}\left(\mathbb{E}(y_a|x), \mathbb{E}(y_b|x)\right) \\
&= \mathbb{E}\left(\mathbb{E}(y_a|x), \mathbb{E}(y_b|x)\right) - (\mathbb{E}\, y_a)(\mathbb{E}\, y_b) \\
&= \alpha_a^T \Psi_{ab} \alpha_b - (\alpha_a^T\, \mathbb{E}\, \phi_a)(\alpha_b^T\, \mathbb{E}\, \phi_b),
\end{aligned} \tag{5.10}$$

where matrix Ψ_{ab} is the expectation of the outer product of feature vectors corresponding to different feature maps ϕ_a, ϕ_b for outputs y_a, y_b, computed similarly as in (5.3.1) with corresponding random frequencies $\{\omega_i\}$, and the scaling coefficient σ_k (5.4). Vectors α_a and α_b are the corresponding weight vectors for y_a and y_b (5.7). Compared to the expression for the variance of a single output in (5.9), the term $\mathbb{E}\left(\mathbb{COV}(y_a|x)\mathbb{COV}(y_b|x)\right)$ that is included in the law of total covariance is neglected due to the assumption of conditional independence of different outputs (§5.2), so (5.10) does not have the corresponding first two terms in (5.9).

Finally, we compute the cross-covariance between input and each output dimension. Invoking the law of total covariance:

$$\begin{aligned}
\mathbb{COV}(x, y) &= \mathbb{COV}(x, \mathbb{E}(y|x)) \\
&= \mathbb{E}\left(x\, \mathbb{E}(y|x)\right) - (\mathbb{E}\, x)(\mathbb{E}\, y) \\
&= \Upsilon\alpha - (\mathbb{E}\, y)\mu,
\end{aligned} \tag{5.11}$$

where matrix Υ is the expectation of the outer product of the input x and the feature vector $\phi(x)$ over input distribution $x \sim \mathcal{N}(\mu, \Sigma)$:

$$\mathbb{E}(x\phi^T) = \Upsilon = \begin{bmatrix} \Upsilon_1^c & \dots & \Upsilon_m^c & \Upsilon_1^s & \dots & \Upsilon_m^s \end{bmatrix},$$
$$\Upsilon_i^c = \sigma_k\, \mathbb{E}\left(\cos(\omega_i^T x)x\right), \quad \Upsilon_i^s = \sigma_k\, \mathbb{E}\left(\cos(\omega_i^T x)x\right),$$

where $i = 1, \dots, m$. We state the following proposition to compute each column in Υ consisting of expectations of the product sinusoidal functions and inputs.

Proposition 2. *The expectation of the multiplication of sinusoids and linear functions over multivariate Gaussian distributions: $x \sim \mathcal{N}(\mu, \Sigma)$, can be computed analytically:*

$$\begin{aligned}
\mathbb{E}\left(\cos(\omega^T x)x\right) &= \left(\mathbb{E}\cos(\omega^T x)\right)\mu - (\mathbb{E}(\sin(\omega^T x))\Sigma\omega, \\
\mathbb{E}\left(\sin(\omega^T x)x\right) &= \left(\mathbb{E}\sin(\omega^T x)\right)\mu + \left(\mathbb{E}\cos(\omega^T x)\right)\Sigma\omega,
\end{aligned}$$

where the right-hand-side expectations have analytical expressions (Proposition 1).

To prove it, we find an expression for $\mathbb{E}\left(a^T x \cos(\omega^T x)\right)$, for any a, through the complex domain trick used to prove Proposition 1. Next, the result is extended to $\mathbb{E}\left(x \cos(\omega^T x)\right)$, by setting a to consist of indicator vectors. Applying Proposition 1 and 2, we complete the derivation of $\mathbb{COV}(x, y)$ in (5.11).

Kernel	$k(x,x')$	$p(\omega)$
Gaussian	$\exp(-\frac{1}{2}\|x-x'\|^2_{\Lambda^{-1}})$	$\mathcal{N}(0,\Lambda^{-1})$
Laplacian	$\exp(-\|x-x'\|_1)$	$\prod_{i=1}^d \frac{1}{\pi(1+\omega_i)}$
Matérn	$\frac{2^{1-\nu}}{\Gamma(\nu)} r^\nu K_\nu(r)$	$h(\frac{2\nu}{\ell^2}+4\pi^2\|\omega\|^2_2)^{\nu+\frac{d}{2}}$

Table 5.1: Examples of continuous shift-invariant positive-definite kernels and their corresponding spectral densities, where $r = \frac{\sqrt{2\nu}\|x-x'\|_2}{\ell}$, K_ν is a modified Bessel function, and $h = \frac{2^d \pi^{\frac{d}{2}} \Gamma(\nu+\frac{d}{2})(2\nu)^\nu}{\Gamma(\nu)\ell^{2\nu}}$.

Remark In summary, SSGP-EMM computes the exact posterior moments. This is equivalent to expectation propagation [120] by minimizing the Kullback-Leibler divergence between the true distribution and its Gaussian approximation with respect to the natural parameters. SSGP-EMM's computation complexity is $O\left(m^2k^2d^2\right)$, where m is the number of features, k is the output dimension, and d is the input dimension. The most computationally demanding part is constructing matrices Ψ_{ab} (5.10) for each output pair, where each requires $O\left(m^2d^2\right)$.

Compared to the multivariate moment-matching approach for GPs (GP-EMM) [49, 69] with $O\left(n^2k^2d^2\right)$ time complexity, SSGP-EMM is more efficient when $m \ll n$. Moreover, our approach is applicable to any positive-definite continuous shift-invariant kernel with different spectral densities (see examples in Table 5.1), while previous approaches like GP-EMM [69] are only derived for squared exponential (SE) or polynomial kernels. Next we introduce a more computationally efficient but less accurate approach that avoids the computation of Ψ_{ab}'s.

5.3.2 Linearization (SSGP-Lin)

An alternative approach to computing the exact moments of the predictive distribution is based on the linearization of the posterior mean function in (5.7) at the input mean μ:

$$m(x) = \alpha^T\phi(x) \approx m(\mu) + \alpha^T \underbrace{D\phi(\mu)}_{M}(x-\mu), \tag{5.12}$$

where $D\phi(\mu)$ denotes taking the derivative of function ϕ at μ. Given the definition of ϕ in (5.4), $D\phi$ can be found by chain rule: $D\phi_i^c(x) = -\sigma_k \sin(\omega_i^T x)\omega_i^T$, $D\phi_i^s(x) = \sigma_k \cos(\omega_i^T x)\omega_i^T$.

Utilizing the linearized posterior mean function (5.12), the predictive moments can be approximated. The predictive mean approximation is

$$\mathbb{E}\, y = \mathbb{E}\,\mathbb{E}(y|x) \approx m(\mu), \tag{5.13}$$

and the predictive variance approximation is

$$\begin{aligned} \mathbb{VAR} y &= \mathbb{E}\,\mathbb{VAR}(y|x) + \mathbb{VAR}\,\mathbb{E}(y|x) \\ &\approx \mathbb{VAR}(y|\mu) + \mathbb{VAR}(\alpha^T M x) \\ &= \sigma_n^2 + \sigma_n^2 \|\phi(\mu)\|_{A^{-1}}^2 + \alpha^T M \Sigma M^T \alpha. \end{aligned} \tag{5.14}$$

and the approximate covariance between output dimension a and b is

$$\begin{aligned} \mathbb{COV}(y_a, y_b) &= \mathbb{COV}\left(\mathbb{E}(y_a|x), \mathbb{E}(y_b|x)\right) \\ &= \mathbb{E}\left(\alpha_a^T M_a (x-\mu)(x-\mu)^T M_b^T \alpha_b\right) \\ &\approx \alpha_a^T M_a \Sigma M_b^T \alpha_b, \end{aligned} \tag{5.15}$$

where M_a and M_b are defined as M in (5.12), except that they correspond to feature maps ϕ_a and ϕ_b. Notice that the assumption of conditional independence between different outputs is invoked here again, *cf.*, (5.10).

Finally, the cross-covariance between the input and output can be approximated as

$$\begin{aligned} \mathbb{COV}(x, y) &= \mathbb{COV}(x, \mathbb{E}(y|x)) \\ &\approx \mathbb{E}\left((x-\mu)(\alpha^T M (x-\mu))\right) \\ &= \alpha^T M \Sigma \end{aligned} \tag{5.16}$$

Method	SSGP-EMM	SSGP-Lin	GP-EMM
Time	$O(m^2k^2d^2)$	$O(m^2k + mk^2d)$	$O(n^2k^2d^2)$
Applicable kernels	continuous shift-invariant kernels	continuous shift-invariant kernels	SE or polynomial kernels

Table 5.2: Comparison of our proposed methods and GP-EMM [49, 69] in terms of computational complexity and generalizability.

Unlike SSGP-EMM, which computes exact moments (§5.3.1), this linearization-based approach SSGP-Lin computes an *approximation* of the predictive moments. In contrast to SSGP-EMM's $O\big(m^2k^2d\big)$ computational complexity, the computation time of SSGP-Lin is reduced to $O\big(m^2kd\big)$, as a direct consequence of avoiding the construction of Ψ (5.3.1) in SSGP-EMM (5.10), which makes SSGP-Lin more efficient than SSGP-EMM, especially when the output dimension is high.

Both SSGP-EMM and SSGP-Lin are applicable to a general family of kernels. See Table 5.2 for a comparison between our methods and GP-EMM [49, 69]. In the next section, we compare these approaches in applications of filtering and control.

5.4 Applications

We focus on the application of the proposed methods to Bayesian filtering and predictive control. We begin by introducing Gauss-Markov models, which can be expressed by the following discrete-time nonlinear dynamical system:

$$x_{t+1} = f(x_t, u_t) + \epsilon_t^x, \qquad \epsilon_t^x \sim \mathcal{N}(0, \Sigma_{\epsilon^x}), \tag{5.17}$$

$$y_t = g(x_t) + \epsilon_t^y, \qquad \epsilon_t^y \sim \mathcal{N}(0, \Sigma_{\epsilon^y}), \tag{5.18}$$

where $x_t \in \mathbf{R}^d$ is state, $u_t \in \mathbf{R}^r$ is control, $y_t \in \mathbf{R}^k$ is observation or measurement, $\epsilon_t^x \in \mathbf{R}^d$ is IID process noise, $\epsilon_t^y \in \mathbf{R}^k$ is IID measurement noise, and subscript t denotes discrete time index. We call the probabilistic models (5.17) and (5.18) the dynamics and observation models, and the corresponding deterministic functions f and g the dynamics and observation functions.

We consider scenarios where f and g are unknown but a dataset $\mathcal{D} = \left(\{(x_t, u_t), x_{t+1}\}_{t=1}^{n-1}, \{x_t, y_t\}_{t=1}^{n}\right)$ is provided. The probabilistic models specified by SSGPs can be learned from the dataset, and then used to model the dynamics and observation (5.17) (5.18). More concretely, the dynamics model $p(x_{t+1}|x_t, u_t)$ is learned using state transition pairs $\{(x_t, u_t), x_{t+1}\}_{t=1}^{n-1}$, and the observation model $p(y_t|x_t)$ is learned separately from state-observation pairs $\{x_t, y_t\}_{t=1}^{n}$.

5.4.1 Bayesian filtering

The task of Bayesian filtering is to infer the posterior distribution of the current state of a dynamical system based on the current and past noisy observations, *i.e.*, finding $p(x_{t|t})$, where the notation $x_{t|s}$ denotes the random variable $x_t|y_0, \ldots, y_s$. Due to the Markov property of the process x, *i.e.*, $x_t|x_0, \ldots, x_{t-1} = x_t|x_{t-1}$, in Gauss-Markov models, $p(x_{t|t})$ can be computed *recursively* through alternating *prediction step* and *correction step*.

Prediction step ($x_{t-1|t-1} \rightarrow x_{t|t-1}$)

In the prediction step, $x_{t-1|t-1}$ is propagated through the dynamics model $p(x_t|x_{t-1}, u_{t-1})$:

$$p(x_{t|t-1}) = \int p(x_t|x_{t-1}, u_{t-1}) p(x_{t-1|t-1}) \, \mathrm{d}x_{t-1},$$

which can be viewed as prediction under uncertainty (5.1). Suppose that $p(x_{t-1|t-1}) = \mathcal{N}(\hat{\mu}_{t-1|t-1}, \hat{\Sigma}_{t-1|t-1})$, with learned SSGP representation for the dynamics, Gaussian approximations of the output: $p(x_{t|t-1}) \approx \mathcal{N}(\hat{\mu}_{t|t-1}, \hat{\Sigma}_{t|t-1})$ can be obtained by either SSGP-EMM (§5.3.1) using (5.8), (5.9) and (5.10), or SSGP-Lin (§5.3.2) using (5.13), (5.14) and (5.15).

Correction step ($x_{t|t-1} \to x_{t|t}$)

The correction step conditions $x_{t|t-1}$ on the current observation y_t using Bayes' rule:

$$p(x_{t|t}) = \frac{p(y_t|x_{t|t-1})p(x_{t|t-1})}{\int p(y_t|x_{t|t-1})p(x_{t|t-1})\,\mathrm{d}x_t}. \tag{5.19}$$

In the preceding prediction step, we obtain $p(x_{t|t-1}) \approx \mathcal{N}(\hat{\mu}_{t|t-1}, \hat{\Sigma}_{t-1|t-1})$, which serves as a prior on x_t in this correction step. Due to the intractability of the integral in the denominator, to apply Bayes' rule we first seek Gaussian approximations for the joint distribution, as in the previous work on Bayesian filtering relying on GPs [70, 112]:

$$\begin{bmatrix} x_{t|t-1} \\ y_{t|t-1} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \hat{\mu}_{t|t-1} \\ \hat{\mu}_y \end{bmatrix}, \begin{bmatrix} \hat{\Sigma}_{t|t-1} & \hat{\Sigma}_{xy} \\ \hat{\Sigma}_{xy}^T & \hat{\Sigma}_y \end{bmatrix} \right), \tag{5.20}$$

Invoking $p(y_{t|t-1}) = \int p(y_t|x_{t|t-1})p(x_{t|t-1})\,\mathrm{d}x_t$, the moments $\hat{\mu}_y$, $\hat{\Sigma}_y$, and $\hat{\Sigma}_{xy}$ in the joint Gaussian approximation can be computed as the predictive mean, predictive covariance, and input-prediction cross-covariance, for the observation model $p(y_t|x_t)$ with input $p(x_{t|t-1})$, using SSGP-EMM or SSGP-Lin. Having all terms in (5.20) determined, we condition $x_{t|t-1}$ exactly on current observation y_t:

$$\begin{aligned} \hat{\mu}_{t|t} &= \hat{\mu}_{t|t-1} + \hat{\Sigma}_{xy}\hat{\Sigma}_y^{-1}(y - \hat{\mu}_y), \\ \hat{\Sigma}_{t|t} &= \hat{\Sigma}_{t|t-1} - \hat{\Sigma}_{xy}\hat{\Sigma}_y^{-1}\hat{\Sigma}_{xy}. \end{aligned} \tag{5.21}$$

This Gaussian approximation $p(x_{t|t}) \approx \mathcal{N}(\hat{\mu}_{t|t}, \hat{\Sigma}_{t|t})$ is then used as input to the prediction step. Thus, we have shown that starting from $p(x_0) = \mathcal{N}(\mu_0, \Sigma_0)$, by consecutively applying prediction and correction steps presented above, we recursively obtain state estimates for $x_{t|t-1}$ and $x_{t|t}$. Rather than using a finite sample-based approximation such as in the GP-UKF [112], the Gaussian approximations of the full densities $p(x_{t|t})$ and $p(x_{t|t-1})$ are propagated.

Algorithm 5 SSGP-ADF and SSGP-EKF

```
1: Model learning: collect dataset $\mathcal{D}$, and learn SSGP dynamics and observations models (§5.2.)
2: Initialization: set prior $p(x_0)$.
3: for $t = 1, \ldots$ do
4:     Prediction: compute $\hat{\mu}_{t|t-1}$ and $\hat{\Sigma}_{t|t-1}$ . by either SSGP-EMM (§5.3.1) or SSGP-Lin (§5.3.2).
5:     Measurement: make an observation $y_t$.
6:     Correction: compute $\hat{\mu}_{t|t}$ and $\hat{\Sigma}_{t|t}$ according to (5.21) by either SSGP-EMM (§5.3.1) or SSGP-Lin (§5.3.2).
7: end for
```

We summarize the resulting filtering algorithm SSGP-ADF (assumed density filtering) and SSGP-EKF (extended Kalman filtering), based on SSGP-EMM and SSGP-Lin, respectively, in Algorithm 5. These are analogs of GP-ADF [70] and GP-EKF [112].

5.4.2 Stochastic Model Predictive Control

The stochastic model predictive control (MPC) problem is to choose a control sequence that minimizes the expected cost, provided $p(x_t)$:

$$u^{\star}_{t+1:t+T} = \underset{u_{t+1:t+T}}{\operatorname{argmin}} \, \mathbb{E}\left(h(x_{t+T}) + \sum_{i}^{i+T} l(x_{t+i}, u_{t+i})\right),$$

at each time step, subject to stochastic system dynamics (5.17), where function $h : \mathbf{R}^d \to \mathbf{R}$ and $l : \mathbf{R}^d \times \mathbf{R}^r \to \mathbf{R}$ are the final and running cost respectively.

There are two main challenges to applying MPC in practice: 1) MPC requires an accurate dynamics model for multi-step prediction, and 2) online optimization is very computationally expensive. For clarity in presentation, we will assume that the state is fully observable henceforth.

Algorithm 6 MPC via probabilistic trajectory optimization (1-3: offline optimization, 4-8: online optimization)

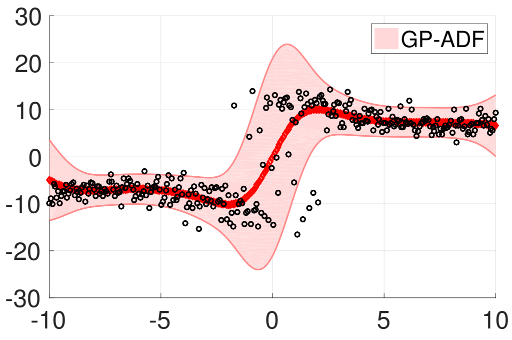
```
1: Model learning: collect dataset $\mathcal{D}$, and learn SSGP dynamics model (§5.2).
2: Initialization: set $t = 0$, and estimate $p(x_0)$.
3: Trajectory optimization: perform trajectory optimization in belief space, obtain $u^\star_{t+1:t+T}$.
4: **repeat**
5:     Policy execution: apply one-step control $u^\star_{t+1}$ to the system and move one step forward, update $t = t + 1$.
6:     Model adaptation: incorporate new data and update SSGP dynamics model.
7:     Trajectory optimization: perform re-optimization with the updated model. Initialize with the previously optimized trajectory and obtain new $u^\star_{t+1:t+T}$.
8: **until** Task terminated
```

MPC via probabilistic trajectory optimization

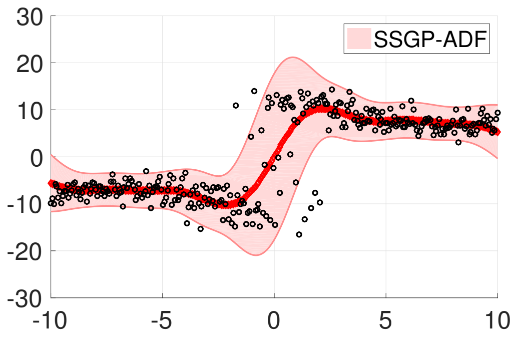
We address the aforementioned challenges by employing a combination of prediction under uncertainty and trajectory optimization. More precisely, we use SSGP-EMM or SSGP-Lin to efficiently obtain approximate Gaussian distribution over trajectory of states and perform trajectory optimization in the resultant *Gaussian belief space* based on differential dynamic programming (DDP) [58, 29]. Note that DDP-related methods require computation of first and second order derivatives of the dynamics and cost. Our analytic moment expressions provide a robust and efficient way to compute these derivatives.

Within the SSGP framework, we may incrementally *update* the posterior distribution over the feature weights w (5.6) given a new sample without storing or inverting the matrix A explicitly, Instead we keep track of its upper triangular Cholesky factor $A = R^T R$ [107]. Given a new sample, a rank-1 update is applied to the Cholesky factor R, which requires $O(m^2)$ time. To cope with time-varying systems and to make the method more adaptive, we employ a forgetting factor $\lambda \in (0, 1)$, such that the impact of the previous samples decays exponentially in time [121].

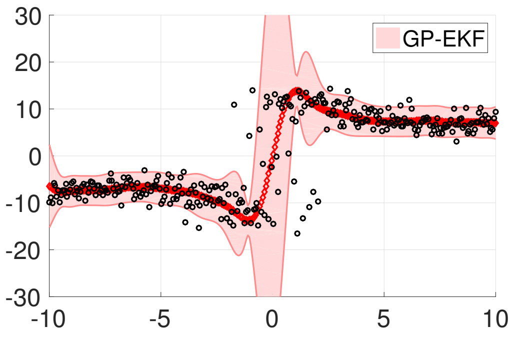
Our proposed MPC algorithm, summarized in Algorithm 6, is related to several algorithms and differs in both model and controller learning. First, SSGPs are more robust to modeling error than Locally Weighted Projection Regression (LWPR) used in iLQG-LD [59]. See a numerical comparison in [107]. Second, we efficiently propagate uncertainty in multi-step prediction which is crucial in MPC. In contrast, AGP-iLQR [68] drops the input uncertainty and uses subset of regressors (SoR-GP) which lacks a principled way to select reference points. In addition, PDDP [85] uses GPs which are computationally expensive for online optimization. Two deep neural networks are used for modeling in [122], which make it difficult to perform online incremental learning, as we do here.



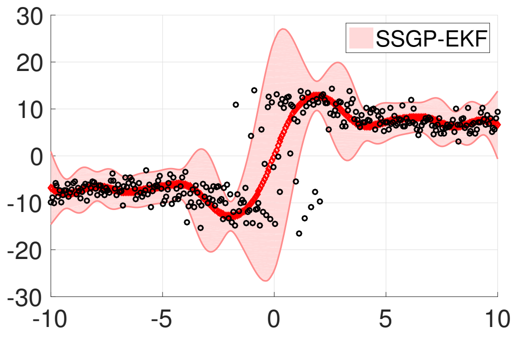
(a) GP-ADF (800 data points)



(b) SSGP-ADF (10 features)



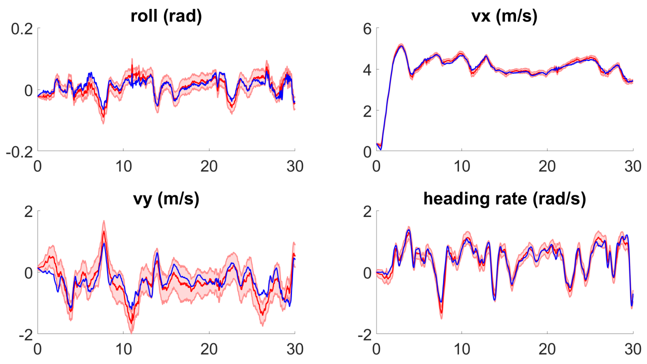
(c) GP-EKF (800 data points)



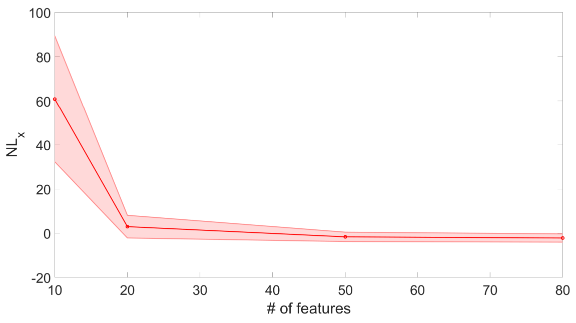
(d) SSGP-EKF (10 features)

Figure 5.1: Black points are ground truth states, red areas are filter distributions for (a) GP-ADF [70], (c) GP-EKF [112], our proposed methods (b) SSGP-ADF and (d) SSGP-EKF. The x-axis is the mean of initial belief $p(x_0)$, which is randomly distributed in $[-10, 10]$ and y-axis shows the mean and twice the standard deviation of filtered distribution $p(x_1|y_1)$ after observing y_1.

(a) Autonomous driving task



(b) Filtered distribution vs. ground truth



(c) NL_x vs. # of features

Figure 5.2: Recursive filtering task for high-speed autonomous driving. Figure (b) shows trajectories of all the states of a 30 seconds continuous driving (1,200 steps), where blue lines are the ground truth, and red lines and red areas are the mean and twice the standard deviation of the filtered distributions respectively. In (c), the red line and area are the mean and twice the standard deviation of NL_x over six 30 seconds driving with varying number of features.

5.5 Experimentals and Analysis

5.5.1 Bayesian filtering

1D One-step filtering

We consider a synthetic dynamical system with ground-truth dynamics $f(x) = \frac{1}{2}x + \frac{25x}{1+x^2}$ and observation $g(x) = 6\sin(2x)$ with $\Sigma_{\epsilon^x} = 1.5^2$ and $\Sigma_{\epsilon^y} = 1$ in (5.17,5.18), in a similar setting to [70]. We compare the performance of four filters, SSGP-ADF, SSGP-EKF, GP-ADF [70] and GP-EKF [112]. All models are trained using 800 samples. However, for SSGP models, only 10 random Fourier features of a SE kernel are used. Figure 5.1

Method	SSGP-ADF	SSGP-EKF	GP-ADF	GP-EKF
NL_x	2.5003	3.415467	2.489385	3.396343
$RMSE$	4.6822	5.1451	4.6854	5.1012

Table 5.3: Comparison of our methods with GP-ADF [70] and GP-EKF [112] in terms of average NL_x (negative log-likelihood) of the ground truth states given estimates and RMSE (root-mean-square error). Lower values are better. The results correspond to the filtering task in sec 5.5.1.

illustrates the comparison of filtered state distribution of a typical realization. We evaluate the methods by computing NL_x (the negative log-likelihood of the ground truth samples in the filtered distribution) and RMSE (root-mean-square error between filtered mean and ground truth samples). See Table 5.3 for a detailed comparison. Our methods SSGP-ADF and SSGP-EKF are able to offer close performance with their full GP counterparts but with greatly reduced computational cost. See the discussion section for further discussions on the comparison between SSGP-ADF and SSGP-EKF.

Recursive filtering

We next consider a state estimation task in high-speed autonomous driving on a dirt track (Figure 5.2a). The goal is to recursively estimate the state of an autonomous rallycar given noisy measurements. The vehicle state consists of linear velocities (x and y), heading rate, and roll angle, in body frame. Controls are steering and throttle. Measurements are collected by wheel speed sensors. This filtering task is challenging because of the complex nonlinear dynamics and the amount of noise in the measurements. We do not use any prior model of the car, but learn the model from ground truth estimates of vehicle state generated by integrating GPS and IMU data via iSAM2 [123]. 50,000 samples are collected from wheel speed sensors and ground truth state estimates from iSAM2 for training. Because of the sample size, it is too computationally expensive to use GP-based filter such as GP-ADF [70]. Instead, we use SSGP-ADF to perform 1,200 recursive filtering steps which correspond to 30 seconds of high-speed driving. Filtered distributions using 80 features are shown in Figure 5.2b, and Figure 5.2c shows the mean and twice the standard deviation of NL_x over six 30 seconds driving with different number of features. Surprisingly, only need a small number of features is necessary for satisfactory results.

5.5.2 Model Predictive Control

Tracking a moving target

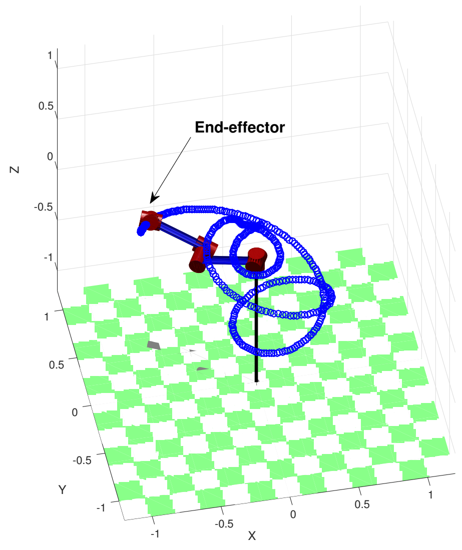
We consider the Puma-560 robotic arm and quadrotor systems with dynamics model specified by SSGPs. For both tasks the goal is to track a moving target. In addition, the true system dynamics vary online, which necessitates both online optimization and model update, as we do here. The tasks are described as follow:

PUMA-560 task: moving target and model parameter changes

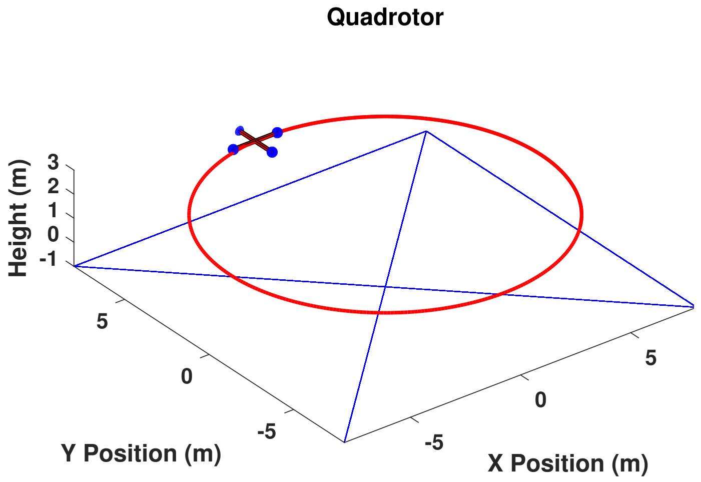
The task is to steer the end-effector to the desired position and orientation. The desired state is time-varying over 800 time steps as shown in fig.5.3a. We collected 1000 data points offline and sampled 50 random features for both of our methods. Similarly for AGP-iLQR we used 50 reference points. In order to show the effect of online adaptation, we increased the mass of the end-effector by 500$\%$ at the beginning of online learning (it is fixed during learning).

Quadrotor task: time-varying tasks and dynamics

The objective is to start at (-1, 1, 0.5) and track a moving target as shown in fig.5.3b for 400 steps. The mass of the quadrotor is decreasing at a rate of 0.02 kg/step. The controls are thrust forces of the 4 rotors and we consider the control constraint $u_{\min} = 0.5, u_{\max} = 3$. We collected 3000 data points offline, and sampled 100 and 400 features for online learning. The forgetting factor for online learning $\lambda = 0.992$. SSGP-Lin was used for approximate inference. The receding-horizon DDP (RH-DDP) [29] with full knowledge of the dynamics model was used as a baseline.



(a) Puma 560 task



(b) Quadrotor task

Figure 5.3: PUMA-560 and quadrotor tasks

Results in terms of cost $l(x_t, u_t)$ are shown in Figure 5.5. Figure 5.5a shows that our methods outperform iLQG-LD [59] and AGP-iLQR [68]. The similarities and differences between these methods have been discussed in §5.4.2. Figure 5.5b shows that model update is necessary and more features could improve performance.

We study the control of an autonomous car during extreme operating conditions (powerslide). The task is to stabilize the vehicle to a specified steady-state using purely longitudinal control during high-speed cornering. This problem has been studied in [124] where the authors developed a LQR control scheme based on a physics-based dynamics model. We apply our MPC algorithm to this task without any prior model knowledge and 2,500 data points generated by the model in [124]. SSGP-Lin is used for multi-step prediction. Results and comparison to [124] are illustrated in Figure 5.4.

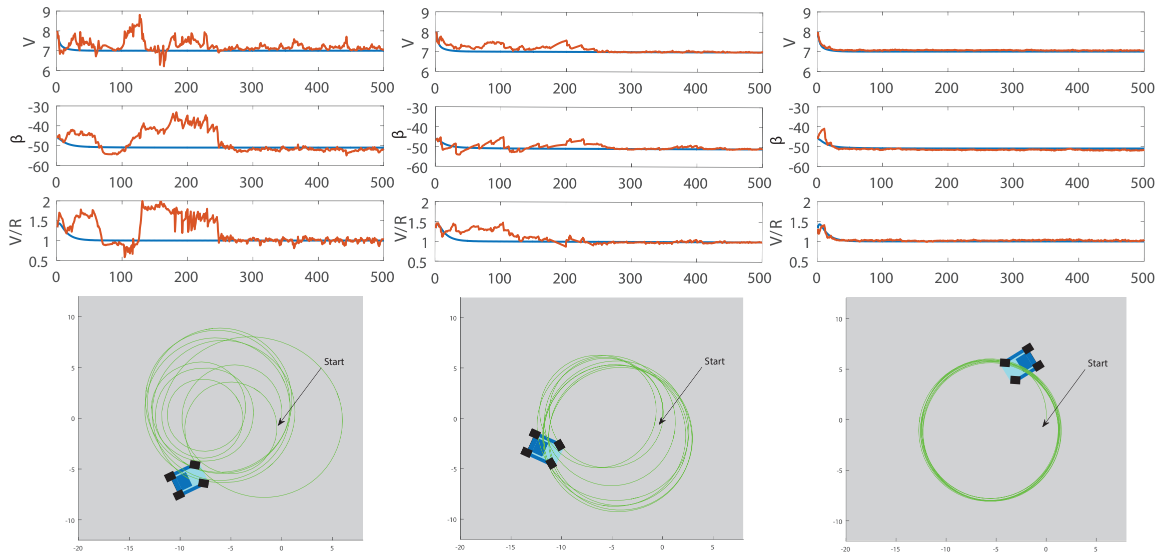
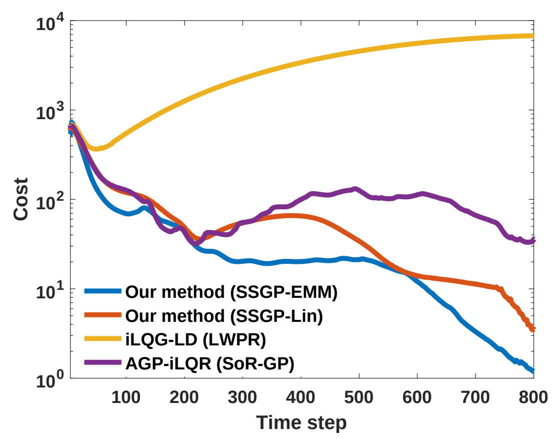


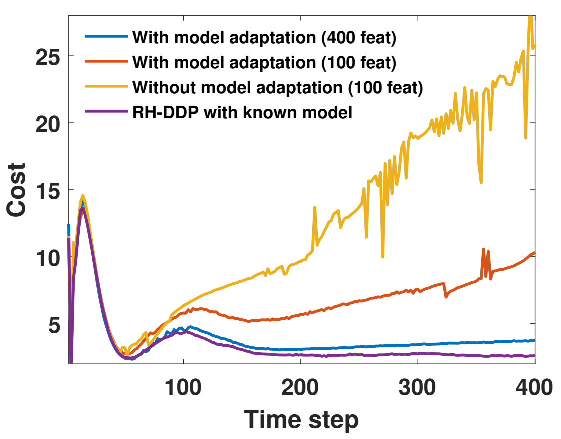
Figure 5.4: Comparison of the drifting performance using 50 (left), 150 (middle) and 400 (right) random features. Blue lines are the solution provided in [124]. Performance improves with a larger number of features, and with a moderate number of features, MPC with SSGP-Lin behaves very closely to the ground truth solution.

5.5.3 Additional experiments on approximate inference

We compare the proposed approximate inference methods with three existing approaches: the full GP exact moment matching (GP-EMM) approach [48, 49, 69], Subset of Regressors GP (SoR-GP) [47] used in AGP-iLQR [68], and LWPR [125] used in iLQG-LD [59]. Note that SoR-GP and LWPR do not take into account input uncertainty when performing



(a) Robotic arm task cost



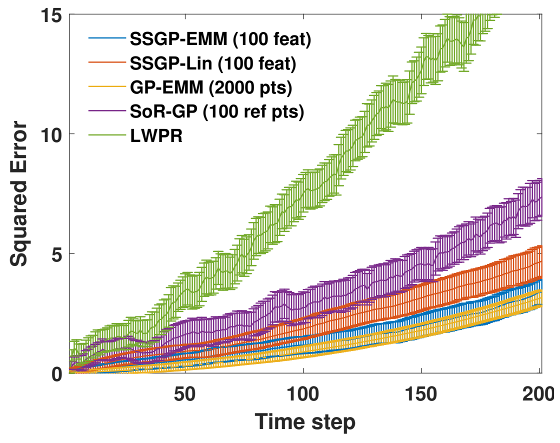
(b) Quadrotor task cost

Figure 5.5: Cost comparison for arm and quadrotor tasks.

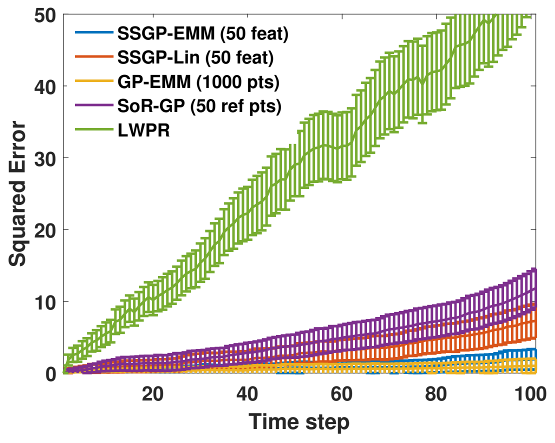
regressions. We consider two multi-step prediction tasks using the dynamics models of a quadrotor (16 state dimensions, 4 control dimensions) and a Puma-560 manipulator (12 state dimensions, 6 control dimensions).

Accuracy of multi-step prediction

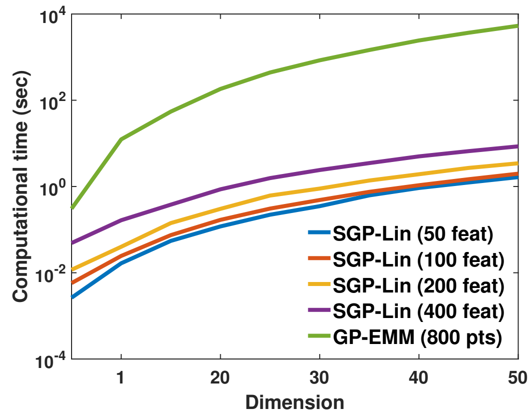
In the following, we evaluate the performance in terms of prediction accuracy. We collected training sets of 1000 and 2000 data points for the quadrotor and puma task, respectively. We used 100 and 50 random features for our methods. We used 100 and 50 reference points for SoR-GP. Based on the learned models, we used a set of 10 initial states and control sequences to perform rollouts (200 steps for quadrotor and 100 steps for Puma) and compute the cost expectations at each step. Fig.5.6(a)(b) shows the cost prediction errors, i.e.$(\mathcal{L}(x_k) - \mathbb{E}\,\mathcal{L}(x_k))^2$. It can be seen that SSGP-EMM is very close to GP-EMM and SSGP-EMM performs slightly better than SSGP-Lin in all cases. Since SoR-GP and LWPR do not take into account input uncertainty when performing regression, our methods outperform them consistently.



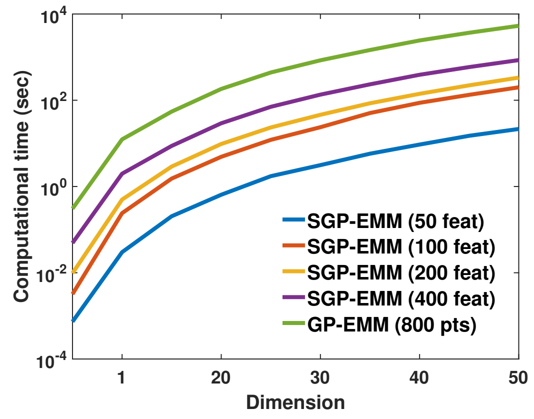
(a) Quadrotor



(b) Puma 560



(c) Computation time



(d) Computation time

Figure 5.6: (a)-(b): Approximate inference accuracy test. The vertical axis is the squared error of cost predictions for (a) quadrotor system and (b) Puma 560 system. Error bars represent standard deviations over 10 independent rollouts. (c)-(d): Comparison of computation time on a log scale between (c) SSGP-Lin and GP-EMM; (d) SSGP-EMM and GP-EMM. The horizontal axis is the input and output dimension (equal in this case). Vertical axis is the CPU time in seconds.

Computational efficiency

In terms of the computational demand, we tested the CPU time for one-step prediction using SSGP-EMM and SSGP-Lin and full GP-EMM. We used sets of 800 random data points of 1,10,20,30,40,50,60,70,80,90 and 100 dimensions to learn SSGP and GP models. The results are shown in fig.5.6c,5.6d. Both SSGP-EMM and SSGP-Lin show significantly less computational demand than GP-EMM with similar prediction performance (fig.5.6c,5.6d).

Our methods are more scalable than GP-EMM, which is the major computational bottleneck for probabilistic model-based RL approaches [50, 85].

5.6 Discussion

5.6.1 Conditional independence between outputs

To deal with multivariate outputs, the assumption of conditional independence between any two output dimensions is imposed, which implies that 1) the noise for different outputs are independent, *e.g.*, Gaussian noise with diagonal covariance matrix, and 2) there's no cross-dependence between channels in the prior, *e.g.*, a vector-valued Gaussian process (GP) prior with a matrix-valued kernel function that only has nonzero entries on the diagonal. These two conditions may be violated in practice. On one hand, the noise may not be independent in general, *e.g.*, wind blowing in some direction causes coupled noise on acceleration for aircraft. On the other hand, one may wish to exploit useful structure between different channels by incorporating them in the prior, *e.g.*, dependence of velocity and acceleration in learning dynamics, and the relation between inverse dynamics models for different loads [126]. But, nevertheless, conditional independence is assumed in most of the related work [50, 112]. And we made this assumption in our experiments as well.

To accommodate conditional dependence in a principled way, vector-valued Gaussian processes can be used [127, 128]. This generally results in a more complicated model with additional computational cost. Incorporating vector-valued GPs would be an interesting extension of this work.

5.6.2 SSGP-EKF vs. SSGP-ADF

Given the results shown earlier, SSGP-EKF seems to underestimate the variance of the filtered distribution when the input is around zero compared with GP-EKF. In this experiment, only 10 random Fourier features were used to approximate an SE kernel. As the number of features increases, the filtered distributions using GP-EKF and SSGP-EKF look more similar.

In general, we hypothesize that SSGP-Lin is more sensitive to a small number of features than SSGP-EMM. In SSGP-Lin we use a locally linear approximation of a nonlinear function, and map a Gaussian distribution through this linear function. Intuitively, this locally linear approximation may vary significantly with the number of features, especially when the number of features is small. In contrast, in SSGP-EMM we compute the moments of the predictive distribution without this locally linear approximation. In order to validate this hypothesis, we performed additional experiments on a one-step approximate inference task, shown in fig. 5.7. This exact phenomenon was observed in these experimental results. More precisely, when a small number of features are used (less than 20), the difference between SSGP-Lin and GP-Lin is greater than the difference between SSGP-EMM and GP-EMM, where the difference is measured by the KL divergence between the predictive distributions.

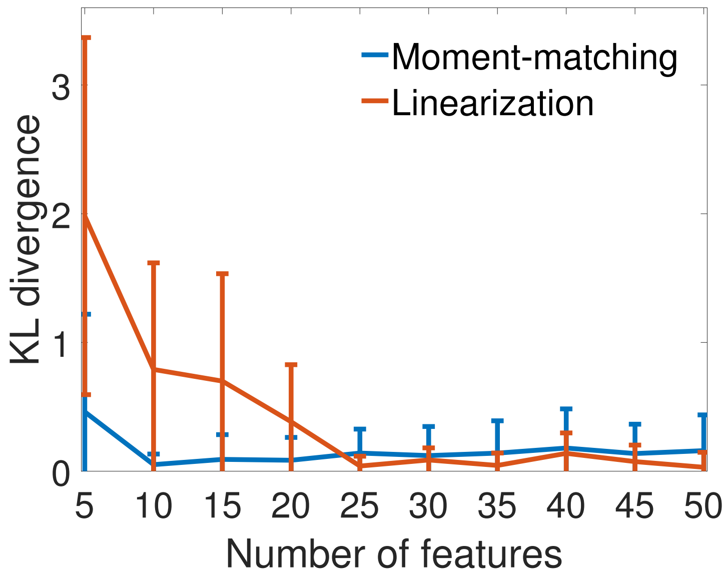


Figure 5.7: KL divergences between SSGP-EMM and GP-EMM, SSGP-Lin and GP-Lin

5.7 Summary

We introduced two analytic moment-based approaches to *prediction under uncertainty* in sparse spectrum Gaussian processes (SSGPs). Compared to their full GP counterparts, our methods are more general: they are applicable to any continuous shift-invariant kernel. They also scale to larger datasets by leveraging random features with frequencies sampled from the spectral density of a given kernel (see Table 5.1, 5.2). Although we adopt the name SSGP, our proposed methods are not tied to specific model learning methods such as linear Bayesian regression [91]. They can be applied to any SSGP with a specified feature weight distribution (5.6), and α and A can be computed via different approaches. For example, A can be iteratively computed by methods like doubly stochastic gradient descent [115].

We studied the application of the proposed methods to Bayesian filtering and model predictive control. Our methods directly address the challenging aspects of these problems: model uncertainty and real-time execution constraints. We evaluated our algorithms on real-world and simulated examples and showed that SSGP-EMM (§5.3.1) and SSGP-Lin (§5.3.2) are accurate alternatives to their full GP counterparts when learning from large amounts of data.

CHAPTER 6
DEEP IMITATION LEARNING FOR AGILE AUTONOMOUS DRIVING

In this chapter, we present an end-to-end imitation learning system for agile, off-road autonomous driving using only low-cost on-board sensors. By imitating an optimal controller, we train a deep neural network control policy to map raw, high-dimensional observations to continuous steering and throttle commands, the latter of which is essential to successfully drive on varied terrain at high speed. Compared with recent approaches to similar tasks, our method requires neither state estimation nor online planning to navigate the vehicle. Real-world experimental results demonstrate successful autonomous off-road driving, matching the state-of-the-art performance.

6.1 Introduction

High-speed autonomous off-road driving is a challenging robotics problem [19, 20, 21] (Fig. 6.1). To succeed in this task, a robot is required to perform both precise steering and throttle maneuvers in a physically-complex, uncertain environment by executing a series of high-frequency decisions. Compared with most previously studied autonomous driving tasks, the robot must reason about unstructured, stochastic natural environments and operate at high speed. Consequently, designing a control policy by following the traditional model-plan-then-act approach [19, 22] becomes challenging, as it is difficult to adequately characterize the robot's interaction with the environment *a priori*.

This task has been considered previously, for example, by Williams et al. [20, 21] using model-predictive control (MPC). While the authors demonstrate impressive results, their internal control scheme relies on expensive and accurate Global Positioning System (GPS) and Inertial Measurement Unit (IMU) for state estimation and demands high-frequency online replanning for generating control commands. Due to these costly hardware require-

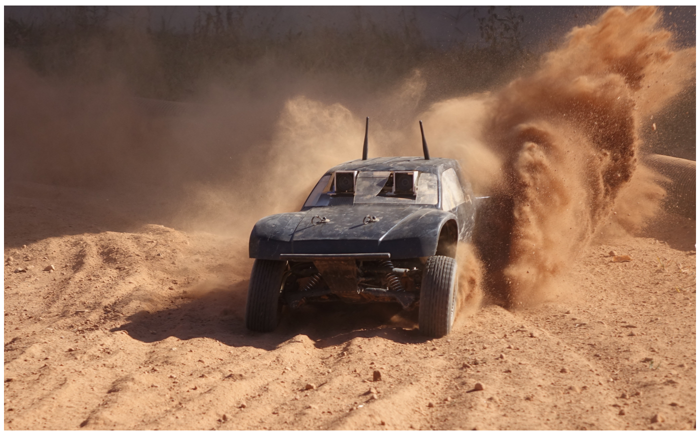


Figure 6.1: High-speed off-road driving task

ments, their robot can only operate in a rather controlled environment.

We aim to relax these requirements by designing a reflexive driving policy that uses only *low-cost, on-board* sensors (e.g. camera, wheel speed sensors). Building on the success of deep reinforcement learning (RL) [129, 4], we adopt deep neural networks (DNNs) to parametrize the control policy and learn the desired parameters from the robot's interaction with its environment. While the use of DNNs as policy representations for RL is not uncommon, in contrast to most previous work that showcases RL in simulated environments [4], our agent is a high-speed physical system that incurs real-world cost: collecting data is a cumbersome process, and a single poor decision can physically impair the robot and result in weeks of time lost while replacing parts and repairing the platform. Therefore, direct application of model-free RL techniques is not only sample inefficient, but costly and dangerous in our experiments.

These real-world factors motivate us to adopt *imitation learning* [131] to optimize the control policy instead. A major benefit of using imitation learning is that we can leverage domain knowledge through *expert* demonstrations.

In this work, we present an imitation learning system for real-world high-speed off-road driving tasks. By leveraging demonstrations from an algorithmic expert, our system can learn a control policy that achieves similar performance as the expert. The system was implemented on a 1/5-scale autonomous AutoRally car. In real-world experiments, we

Table 6.1: Comparison of our method to prior work on imitation learning for autonomous driving

Methods	Tasks	Observations	Action	Algorithm	Experiment
[130]	On-road low-speed	Single image	Steering	Batch	Real &simulated
[131]	On-road low-speed	Single image & laser	Steering	Batch	Real &simulated
[132]	On-road low-speed	Single image	Steering	Batch	Simulated
[133]	Off-road low-speed	Left & right images	Steering	Batch	Real
[134]	On-road unknown speed	Single image	Steering + break	Online	Simulated
Our Method	Off-road high-speed	Single image + wheel speeds	Steering + throttle	Batch & online	Real & simulated

show the AutoRally car—without any state estimator or online planning, but with a DNN policy that directly inputs sensor measurements from a low-cost monocular camera and wheel speed sensors—could learn to perform high-speed navigation at an average speed of $\sim$6 m/s and a top speed of $\sim$8 m/s, matching the state of the art [21].

6.2 Relation to Existing Works

End-to-end learning for self-driving cars has been explored since late 1980's. Autonomous Land Vehicle in a Neural Network (ALVINN) [131] was developed to learn steering angles directly from camera and laser range measurements using a neural network with a single hidden layer. Based on similar ideas, modern self-driving cars [133, 130, 132] have recently started to employ a batch imitation learning approach: parameterizing control policies with DNNs, these systems require only expert demonstrations during the training phase and on-board measurements during the testing phase. For example, Nvidia's PilotNet [130], a convolutional neural network that outputs steering angle given an image, was trained to mimic human drivers' reaction to visual input with demonstrations collected in real-world road tests. A Dataset Aggregation (DAgger) [135] related online imitation learning algorithm for autonomous driving was recently demonstrated in [134], but only considered simulated environments.

Our problem differs substantially from these previous on-road driving tasks. We study autonomous driving on a fixed set of dirt tracks, whereas on-road driving must perform well in a larger domain and contend with moving objects such as cars and pedestrians. While on-road driving in urban environments may seem more difficult, our agent must overcome challenges of a different nature. It is required to drive at high speed, and prominent visual features such as lane markers are absent. Compared with paved roads, the surface of our dirt tracks are constantly evolving and highly stochastic. As a result, to successfully perform high-speed driving in our task, high-frequency application of both steering and throttle commands are required. Previous work only focuses on steering commands [133, 130, 132]. A comparison of different imitation learning approaches to autonomous driving is presented in Table 6.1.

Our task is most similar to the task considered by Williams et al. [20, 21] and Drews et al. [136]. Compared with a DNN policy, their MPC approach has several drawbacks: computationally expensive optimization for planning is required to be performed online at high-frequency and the learning component is not end-to-end. In [20, 21], accurate GPS and IMU feedbacks are also required for state estimation, which may not contain sufficient information to contend with the changing environment in off-road driving tasks. While the requirement on GPS and IMU is relaxed by using a vision-based cost map in [136], a large dataset (300,000 images) was used to train the model, expensive on-the-fly planning is still required, and speed performance is compromised. In contrast to previous work, our approach off-loads the hardware requirements to an expert. While the expert may use high-quality sensors and more computational power, our agent only needs access to on-board sensors and its control policy can run reactively in high frequency, without on-the-fly planning and optimization. Additionally, our experimental results match that in [20, 21] and are faster and more data efficient than that in [136].

6.3 Imitation Learning for Autonomous Driving

To design a policy for off-road autonomous driving, we introduce a policy optimization problem and then show how a policy can be learned by imitation learning. We discuss the strengths and weakness of deploying a batch or online imitation learning algorithm to our task.

6.3.1 Problem Definition

To mathematically formulate the autonomous driving task, it is natural to consider a discrete-time continuous-valued RL problem. Let $\mathbb{S}$, $\mathbb{A}$, and $\mathbb{O}$ be the state, action, and the observation spaces. In our setting, the state space is unknown to the agent; observations consist of on-board measurements, including a monocular RGB image from the front-view camera and wheel speeds from Hall effect sensors; actions consist of continuous-valued steering and throttle commands.

The goal is to find a stationary deterministic policy $\pi : \mathbb{O} \mapsto \mathbb{A}$ (e.g. a DNN policy) such that π achieves low accumulated cost over a finite horizon of length T

$$\min_{\pi} J(\pi) := \min_{\pi} \mathbb{E}_{\rho_\pi} \left[\sum_{t=0}^{T-1} c(s_t, a_t) \right] \tag{6.1}$$

in which ρ_π is the distribution of $s_t \in \mathbb{S}$, $o_t \in \mathbb{O}$, and $a_t \in \mathbb{A}$ under policy $a_t = \pi(o_t)$, for $t = 1 \ldots T$. Here $c(s_t, a_t)$ is the instantaneous cost, which, for example, encourages maximal speed driving while staying on the track. For notations, we denote $Q_\pi^t(s, a)$ as the Q-function at time t under policy π and $V_\pi^t = \mathbb{E}_{a \sim \pi}[Q_\pi^t(s, a)]$ as its associated value function.

6.3.2 Imitation Learning

Directly optimizing (6.1) is challenging for high-speed off-road autonomous driving. Since our task involves a physical robot, model-free RL techniques are intolerably sample inefficient and have the risk of permanently damaging the car when applying a partially-optimized policy in exploration. Although model-based RL may require fewer samples, it can lead to suboptimal, potentially unstable, results because it is difficult for a model that uses only on-board measurements to fully capture the complex dynamics of off-road driving.

Considering these limitations, we propose to solve for the policy π by imitation learning. We assume the access to an oracle policy or *expert* π^* to generate demonstrations during the training phase, which relies on resources that are unavailable in the testing phase, e.g. additional sensors and computation. For example, the expert can be a computationally intensive optimal controller that relies on exteroceptive sensors not available at test time (e.g. GPS for state estimation), or an experienced human driver.

The goal of imitation learning is to perform as well as the expert with an error that has at most linear dependency on T. Formally, we introduce a lemma due to Kakade and Langford [137] and define what we mean by an *expert*.

Lemma 4. *Define* $d_\pi(s) = \sum_{t=0}^{T-1} d_\pi^t(s)$ *as an unnormalized stationary state distribution, where* d_π^t *is the distribution of state at time* t *when running policy* π*. Let* π *and* π' *be two policies. Then*

$$J(\pi) = J(\pi') + \mathop{\mathbb{E}}_{s \sim d_\pi} \mathop{\mathbb{E}}_{a \sim \pi} [A_{\pi'}^t(s,a)] \tag{6.2}$$

where $A_{\pi'}^t(s,a) = Q_{\pi'}^t(s,a) - V_{\pi'}^t(s)$ *is the advantage function at time* t *with respect to running* π'*.*

Definition 3. *A policy* π^* *is called an* expert *to problem* (6.1) *if* $C_{\pi^*} = \sup_{t \in [0,T-1], s \in \mathbb{S}}$ *Lip* $(Q_{\pi^*}^t(s,\cdot)) \in O(1)$ *independent of* T*, where Lip*$(f(\cdot))$ *denotes the Lipschitz constant of function* f *and*

$Q^t_{\pi^*}(s,a)$ *is the Q-function at time* t *of running policy* π^*.

The idea behind Definition 3 is that an expert policy π^* should perform stably under arbitrary action perturbation with respect to the cost function $c(\cdot,\cdot)$, regardless of where it starts.[1] As we will see in Section 6.3.3, this requirement provides guidance for whether to choose batch learning vs. online learning to train a policy by imitation.

Online Imitation Learning

We now present the objective function for the online learning [138] approach to imitation learning, which simplifies the derivation in [135] and extends it to continuous action spaces as required in the autonomous driving task. Although our goal here is not to introduce a new algorithm, but rather to give a concise introduction to online imitation learning, we found that a connection between online imitation learning and DAgger-like algorithms [135] in continuous domains has not been formally introduced. DAgger has only been used heuristically in these domains as in [139, 134].

Assume π^* is an expert to (6.1) and suppose $\mathbb{A}$ is a normed space with norm $\|\cdot\|$. Let $D_W(\cdot,\cdot)$ denote the Wasserstein metric [140]: for two probability distributions p and q defined on a metric space $\mathcal{M}$ with metric d,

$$\begin{aligned} D_W(p,q) &:= \sup_{f:\mathrm{Lip}(f(\cdot))\le 1} \mathop{\mathbb{E}}_{x\sim p}[f(x)] - \mathop{\mathbb{E}}_{x\sim q}[f(x)] && (6.3)\\ &= \inf_{\gamma\in\Gamma(p,q)} \int_{\mathcal{M}\times\mathcal{M}} d(x,y)d\gamma(x,y), && (6.4)\end{aligned}$$

where Γ denotes the family of distributions whose marginals are p and q. It can be shown by the Kantorovich-Rubinstein theorem that the above two definitions are equivalent [140]. These assumptions allow us to construct a surrogate problem, which is relatively easier to solve than (6.1). We achieve this by upper-bounding the difference between the perfor-

[1]We define the expert here using an uniform Lipschitz constant because the action space in our task is continuous; for discrete action spaces, $\mathrm{Lip}\,(Q^t_{\pi^*}(s,\cdot))$ can be replaced by $\sup_{a\in\mathbb{A}} Q^t_{\pi^*}(s,a)$ and the rest applies.

mance of π and π' given in Lemma 4:

$$\begin{aligned}
&J(\pi) - J(\pi^*) \\
&= \mathop{\mathbb{E}}_{s_t \sim d_\pi} \left[\mathop{\mathbb{E}}_{a_t \sim \pi} [Q^t_{\pi^*}(s_t, a_t)] - \mathop{\mathbb{E}}_{a_t^* \sim \pi^*} [Q^t_{\pi^*}(s_t, a_t^*)] \right] \\
&\leq C_{\pi^*} \mathop{\mathbb{E}}_{s_t \sim d_\pi} [D_W(\pi, \pi^*)] \\
&\leq C_{\pi^*} \mathop{\mathbb{E}}_{s_t \sim d_\pi} \mathop{\mathbb{E}}_{a_t \sim \pi} \mathop{\mathbb{E}}_{a_t^* \sim \pi^*} [\|a_t - a_t^*\|], \qquad (6.5)
\end{aligned}$$

where we invoke the definition of advantage function $A^t_{\pi^*}(s_t, a_t) = Q^t_{\pi^*}(s_t, a_t) - \mathbb{E}_{a_t^* \sim \pi^*}[Q^t_{\pi^*}(s_t, a_t^*)]$, the first and the second inequalities is due to (6.3) and (6.4), respectively.

Define $\hat{c}(s_t, a_t) = \mathbb{E}_{a_t^* \sim \pi^*}[\|a_t - a_t^*\|]$. Thus, to make π perform as well as π^*, we can minimize the upper bound, which is equivalent to solving a surrogate RL problem

$$\min_\pi \mathbb{E}_{\rho_\pi} \left[\sum_{t=1}^T \hat{c}(s_t, a_t) \right]. \qquad (6.6)$$

The optimization problem (6.6) is called the *online* imitation learning problem. This surrogate problem is comparatively more structured than the original RL problem (6.1), so we can adopt algorithms with provable performance guarantees. In this work, we use the meta-learning algorithm DAgger [135], which reduces (6.6) to a sequence of supervised learning problems: Let $\mathcal{D}$ be the training data. DAgger initializes $\mathcal{D}$ with samples gathered by running π^*. Then, in the ith iteration, it trains π_i by supervised learning,

$$\pi_i = \arg\min_\pi \mathbb{E}_{\mathcal{D}}[\hat{c}(s_t, a_t)], \qquad (6.7)$$

where subscript $\mathcal{D}$ denotes empirical data distribution. Next it runs π_i to collect more data, which is then added into $\mathcal{D}$ to train π_{i+1}. The procedure is repeated for $O(T)$ iterations and the best policy, in terms of (6.6), is returned. Suppose the policy is linearly parametrized and $o_t = x_t$. Since our instantaneous cost $\hat{c}(s_t, \cdot)$ is strongly convex, the theoretical analysis of DAgger applies. Therefore, together with the assumption that π^* is an expert, running DAgger to solve (6.6) finds a policy π with performance $J(\pi) \leq J(\pi^*) + O(T)$, achieving our initial goal.

We note here the instantaneous cost $\hat{c}(s_t, \cdot)$ can be selected to be any suitable norm according the problem's property. In our off-road autonomous driving task, we find $l1$-norm is preferable (e.g. over $l2$-norm) for its ability to filter outliers in a highly stochastic environment.

Batch Imitation Learning

By swapping the order of π and π^* in the above derivation in (6.5), we can derive another upper bound and use it to construct another surrogate problem: define $\tilde{c}_\pi(s_t^*, a_t^*) = \mathbb{E}_{a_t \sim \pi}[\|a_t - a_t^*\|]$ and $C_\pi^t(s_t^*) = \text{Lip}(Q_\pi^t(s_t^*, \cdot))$, then

$$\begin{aligned} & J(\pi) - J(\pi^*) \\ & = \mathop{\mathbb{E}}_{s_t^* \sim d_{\pi^*}} \left[\mathop{\mathbb{E}}_{a_t \sim \pi} [Q_\pi^t(s_t^*, a_t)] - \mathop{\mathbb{E}}_{a_t^* \sim \pi^*} [Q_\pi^t(s_t^*, a_t^*)] \right] \\ & \leq \mathop{\mathbb{E}}_{s_t^* \sim d_{\pi^*}} \mathop{\mathbb{E}}_{a_t^* \sim \pi^*} \left[C_\pi^t(s_t^*) \tilde{c}_\pi(s_t^*, a_t^*) \right] . \end{aligned} \tag{6.8}$$

where we use again Lemma 4 for the equality and the property of Wasserstein distance for inequality. The minimization of the upper-bound (6.8) is called *batch* imitation learning problem [130, 132]:

$$\min_\pi \mathbb{E}_{\rho_{\pi^*}} \left[\sum_{t=1}^{T} \tilde{c}_\pi(s_t^*, a_t^*) \right] , \tag{6.9}$$

In contrast to the surrogate problem in online imitation learning (6.6), batch imitation learning reduces to a supervised learning problem, because the expectation is defined by a fixed policy π^*.

6.3.3 Comparison of Imitation Learning Algorithms

Comparing (6.5) and (6.8), we observe that in batch imitation learning the Lipschitz constant $C_\pi^t(s_t^*)$, without π being an expert, can be on the order of $T - t$ in the worst case. Therefore, if we take a uniform bound and define $C_\pi = \sup_{t\in[0,T-1],s\in\mathbb{S}} C_\pi^t(s)$, we see $C_\pi \in O(T)$. In other words, under the same assumption in online imitation, i.e. (6.8) can be minimized to an error that depends linearly on T, the difference between $J(\pi)$ and $J(\pi^*)$ in batch imitation learning can actually grow quadratically in T due to error compounding. Therefore, in order to achieve the same level of performance as online imitation learning, batch imitation learning requires a more expressive policy class or more demonstration samples. As shown in [135], the quadratic bound is tight.

Therefore, if we can choose an expert policy π^* that is stable in the sense of Definition 3, then online imitation learning is preferred theoretically. This is satisfied, for example, when the expert policy is an algorithm with certain performance characteristics. On the contrary, if the expert is human, the assumptions required by online imitation learning becomes hard to realize in real-road off-road driving tasks. Because humans rely on real-time sensory feedback (as in sampling from ρ_{π^*} but not from ρ_π) to generate ideal expert actions, the action samples collected in the online learning approach using ρ_π are often biased and inconsistent [141]. This is especially true in off-road driving tasks, where the human driver depends heavily on instant feedback from the car to overcome stochastic disturbances. Therefore, the frame-by-frame labeling approach [139], for example, can lead to a very counter-intuitive, inefficient data collection process, because the required dynamics information is lost in a single image frame. Overall, when using human demonstrations, online imitation learning can be as bad as batch imitation learning [141], just due to inconsistencies introduced by human nature.

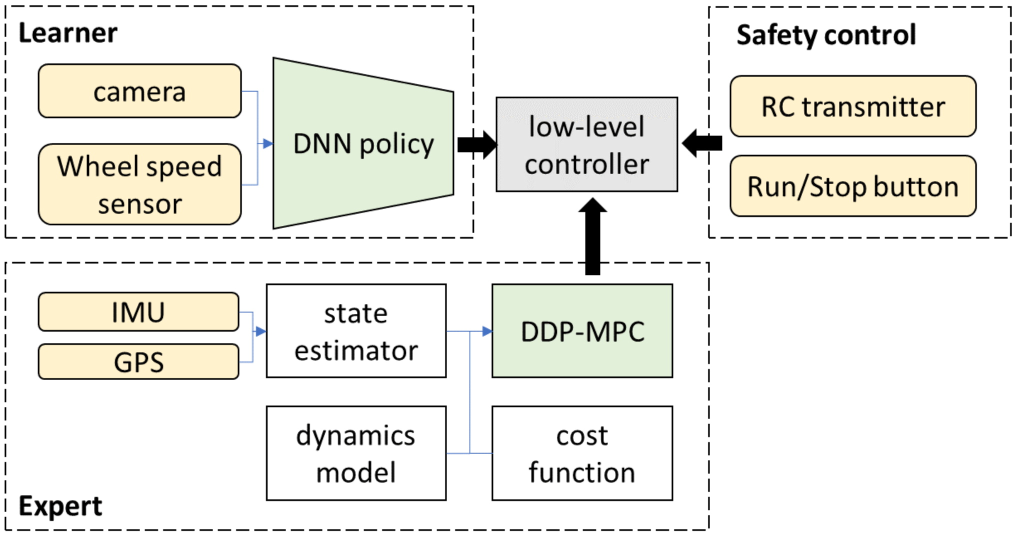


Figure 6.2: System diagram

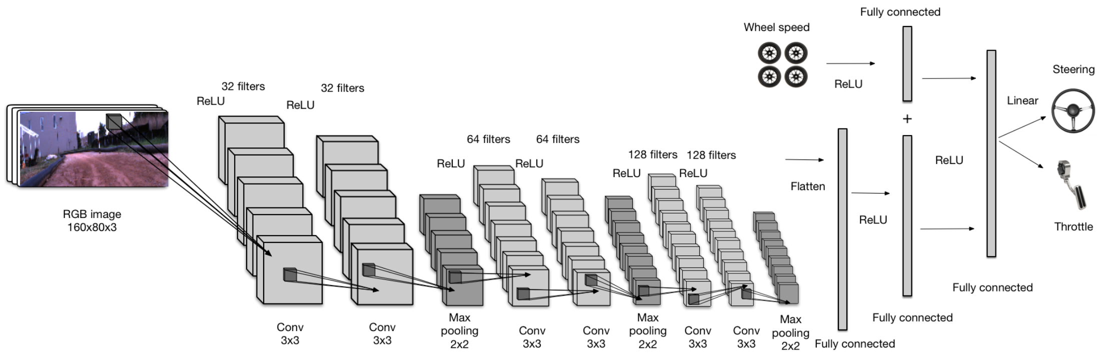


Figure 6.3: The DNN control policy

6.4 The Autonomous Driving System

Building on the analyses in the previous section, we design a system that can learn to perform fast off-road autonomous driving with only on-board measurements. The overall system architecture for learning end-to-end DNN driving policies is illustrated in Fig. 6.2. It consists of three high-level controllers (an expert, a learner, and a safety control module) and a low-level controller, which receives steering and throttle commands from the high-level controllers and translates them to pulse-width modulation (PWM) signals to drive the steering and throttle actuators of a vehicle.

On the basis of the analysis in Section 6.3.3, we assume the expert is algorithmic and has access to expensive sensors (GPS and IMU) for accurate global state estimates[2] and resourceful computational power. The expert is built on multiple hand-engineered components, including a state estimator, a dynamics model of the vehicle, a cost function of the task, and a trajectory optimization algorithm for planning (see Section 6.4.1). By contrast, the learner is a DNN policy that has access to only a monocular camera and wheel speed sensors and is required to output steering and throttle command directly (see Section 6.4.2). In this setting, the sensors that the learner uses can be significantly cheaper than that of the expert; specifically on our experimental platform, the AutoRally car (see Section 6.4.3), the IMU and the GPS sensors required by the expert in Section 6.4.1 together cost more than \$6,000, while the sensors used by the learner's DNN policy cost less than \$500. The safety control module has the highest priority among all three controllers and is used prevent the vehicle from high-speed crashing.

The software system was developed based on the Robot Operating System (ROS) in Ubuntu. In addition, a Gazebo-based simulation environment [142] was built using the same ROS interface but without the safety control module; the simulator was used to evaluate the performance of the system before real track tests.

6.4.1 Algorithmic Expert with Model-Predictive Control

We use an MPC expert [94] based on an incremental Sparse Spectrum Gaussian Process (SSGP) dynamics model (which was learned from 30 minute-long driving data) and an iSAM2 state estimator [123]. To generate actions, the MPC expert solves a finite horizon optimal control problem for every sampling time: at time t, the expert policy $\pi^*(a_t|s_t)$ is a locally optimal policy such that

$$\pi^*(a_t|s_t) \approx \arg\min_{\pi} \mathbb{E}_{\rho_\pi} \left[\sum_{\tau=t}^{t+T_h} c(s_\tau, a_\tau) | s_t \right] \tag{6.10}$$

where T_h is the length of horizon it previews.

[2]Global position, heading and roll angles, linear velocities, and heading angle rate.

The computation is realized by the trajectory optimization algorithm, Differential Dynamic Programming (DDP) [143]: in each iteration of DDP, the system dynamics and the cost function are approximated quadratically along a nominal trajectory; then the Bellman equation of the approximate problem is solved in a backward pass to compute the control law; finally, a new nominal trajectory is generated by applying the updated control law through the dynamics model in a forward pass. Upon convergence, DDP returns a locally optimal control sequence $\{\hat{a}_t^*, ..., \hat{a}_{t+T_h-1}^*\}$, and the MPC expert executes the first action in the sequence as the expert's action at time t (i.e. $a_t^* = \hat{a}_t^*$). This process is repeated at every sampling time.

In view of the analysis in Section 6.3.2, we can assume that the MPC expert satisfies Definition 3, because it updates the approximate solution to the original RL problem (6.1) in high-frequency using global state information. However, because the MPC requires replanning for every step, running the expert policy (6.10) online consumes significantly more computational power than what is required by the learner.

6.4.2 Learning a DNN Control Policy

The learner's control policy π is parametrized by a DNN containing $\sim$10 million parameters. As illustrated in Fig. 6.3, the DNN policy, consists of two sub-networks: a convolutional neural network (CNN) with 6 convolutional layers, 3 max-pooling layers and 2 fully-connected layers that takes 160×80 RGB monocular images as inputs[3], and a feedforward network with a fully-connected hidden layer, that takes wheel speeds as inputs. The convolutional and max-pooling layers are used to extract lower-dimensional features from images. The DNN policy uses 3×3 filters for all convolutonal layers, and rectified linear unit (ReLU) activation for all layers except the last one. Max-pooling layers with 2×2 filters are integrated to reduce the spatial size of the representation (and therefore reduce the number of parameters and computation loads). The two sub-networks are concatenated

[3]The raw images from the camera were re-scaled to 160×80.

and then followed by another fully-connected hidden layer. The structure of this DNN was selected empirically based on experimental studies of several different architectures.

In construction of the surrogate problem for imitation learning, the action space $\mathbb{A}$ is equipped with $\|\cdot\|_1$ for filtering outliers, and the optimization problem, (6.7) or (6.9), is solved using ADAM [144], which is a stochastic gradient descent algorithm with an adaptive learning rate. Note while s_t or s_t^* is used in (6.7) or (6.9), the neural network policy does not use the state, but rather the synchronized raw observation o_t, as input. Note that we did not perform any data selection or augmentation techniques in any of the experiments. The only pre-processing was scaling and cropping raw images.

6.4.3 The Autonomous Driving Platform

To validate our imitation learning approach to off-road autonomous driving, the system was implemented on a custom-built, 1/5-scale autonomous AutoRally car (weight 22 kg; LWH 1m$\times$0.6m$\times$0.4m), shown in the top figure in Fig. 6.4. The car was equipped with an ASUS mini-ITX motherboard, an Intel quad-core i7 CPU, 16GB RAM, a Nvidia GTX 750ti GPU, and a 11000mAh battery. For sensors, two forward facing machine vision cameras[4], a Hemisphere Eclipse P307 GPS module, a Lord Microstrain 3DM-GX4-25 IMU, and Hall effect wheel speed sensors were instrumented. In addition, an RC transmitter could be used to remotely control the vehicle by a human, and a physical run-stop button was installed to disable all motions in case of emergency.

In the experiments, all computation was executed on-board the vehicle in real-time. In addition, an external laptop was used to communicate with the on-board computer remotely via Wi-Fi to monitor the vehicle's status. The observations were sampled and action were executed at 50 Hz to account for the high-speed of the vehicle and the stochasticity of the environment. Note this control frequency is significantly higher than [130] (10 Hz), [132] (12 Hz), and [133] (15 Hz).

[4]In this work we only used one of the cameras.

6.5 Experimental Setup

6.5.1 High-speed Navigation Task

We tested the performance of the proposed imitation learning system in Section 6.4 in a high-speed navigation task with a desired speed of 7.5 m/s. The performance index of the task was formulated as the cost function in the finite-horizon RL problem (6.1) with

$$\begin{aligned} c(s_t, a_t) = {} & \alpha_1 \text{cost}_{\textbf{pos}}(s_t) + \alpha_2 \text{cost}_{\textbf{spd}}(s_t) \\ & + \alpha_3 \text{cost}_{\textbf{slip}}(s_t) + \alpha_3 \text{cost}_{\textbf{act}}(a_t), \end{aligned} \tag{6.11}$$

in which $cost_{\textbf{pos}}$ favors the vehicle to stay in the middle of the track, $cost_{\textbf{spd}}$ drives the vehicle to reach the desired speed, $cost_{\textbf{slip}}$ stabilizes the car from slipping, and $cost_{\textbf{act}}$ inhibits large control commands.

More precisely, the position cost $\text{cost}_{\textbf{pos}}(s)$ for the high-speed navigation task is a 16-term cubic function of the vehicle's global position (x, y):

$$\begin{aligned} \text{cost}_{\textbf{pos}}(s) = {} & c_0 + c_1 y + c_2 y^2 + c_3 y^3 + c_4 x + c_5 xy \\ & + c_6 xy^2 + c_7 xy^3 + c_8 x^2 + c_9 x^2 y + c_{10} x^2 y^2 + c_{11} x^2 y^3 \\ & + c_{12} x^3 + c_{13} x^3 y + c_{14} x^3 y^2 + c_{15} x^3 y^3. \end{aligned}$$

This coefficients in this cost function were identified by performing a regression to fit the track's boundary: First, a thorough GPS survey of the track was taken. Points along the inner and the outer boundaries were assigned values of -1 and $+1$, respectively, resulting in a zero-cost path along the center of the track. The coefficient values c_i were then determined by a least-squares regression of the polynomials in $\text{cost}_{\textbf{pos}}(s)$ to fit the boundary data.

The speed cost $cost_{\textbf{spd}} = \|v_x - v_{\textbf{desired}}\|^2$ is a quadratic function which penalizes the difference between the desired speed $v_{\textbf{desired}}$ and the longitudinal velocity v_x in the body frame. The side slip angle cost is defined as $\text{cost}_{\textbf{slip}}(s) = -\arctan^2(\frac{v_y}{\|v_x\|})$, where v_y is the lateral velocity in the body frame. The action cost is a quadratic function defined as $\text{cost}_{\textbf{act}}(a) = \gamma_1 a_1 + \gamma_2 a_2$, where a_1 and a_2 correspond to the steering and the throttle commands, respectively. In the experiments, $\gamma_1 = 1$ and $\gamma_2 = 1$ were selected.

The goal of the high-speed navigation task to minimize the accumulated cost function over one-minute continuous driving. That is, under the 50-Hz sampling rate, the task horizon was set $T = 3,000$. The cost information (6.11) was given to the MPC expert in Fig. 6.2 to perform online trajectory optimization with a two-second preview horizon (i.e. $T_h = 100$). In the experiments, the weighting in (6.11) were set as $\alpha_1 = 2.5$, $\alpha_2 = 1$, $\alpha_3 = 100$ and $\alpha_4 = 60$, so that the MPC expert in Section 6.4.1 could perform reasonably well. The learner's policy was tuned by online/batch imitation learning in attempts to match the expert's performance.

6.5.2 Test Track

All the experiments were performed on an elliptical dirt track, shown in the bottom figure of Fig. 6.4, with the AutoRally car described in Section 6.4.3. The test track was $\sim$3m wide and $\sim$30m long and built with fill dirt. Its boundaries were surrounded by soft HDPE tubes, which were detached from the ground, for safety during experimentation. Due to the changing dirt surface, debris from the track's natural surroundings, and the shifting track boundaries after car crashes, the track condition and vehicle dynamics can change from one experiment to the next, adding to the complexity of learning a robust policy.

6.5.3 Data Collection

Training data was collected in two ways. In batch imitation learning, the MPC expert was executed, and the camera images, wheel speed readings, and the corresponding steering and throttle commands were recorded. In online imitation learning, a mixture of the expert and learner's policy was used to collect training data (camera images, wheel speeds, and expert actions): in the ith iteration of DAgger, a mixed policy was executed at each time step $\hat{\pi}_i =$

$\beta^i \pi^* + (1-\beta^i)\pi_{i-1}$, where π_{i-1} is learner's DNN policy after $i-1$ DAgger iterations, and β^i is the probability of executing the expert policy. The use of a mixture policy was suggested in [135] for better stability. A mixing rate $\beta = 0.6$ was used in our experiments. Note that the probability of using the expert decayed exponentially as the number of DAgger iterations increased. Experimental data was collected on an outdoor track, and consisted on changing lighting conditions and environmental dynamics. In the experiments, the rollouts about to crash were terminated remotely by overwriting the autonomous control commands with the run-stop button or the RC transmitter in the safety control module; these rollouts were excluded from the data collection.

6.5.4 Policy Learning

In online imitation learning, three iterations of DAgger were performed. At each iteration, the robot executed one rollout using the mixed policy described above (the probabilities of executing the expert policy were 60%, 36%, and 21%, respectively). For a fair comparison, the amount of training data collected in batch imitation learning was the same as all of the data collected over the three iterations of online imitation learning.

At each training phase, the optimization problem (6.7) or (6.9) was solved by ADAM for 20 epochs, with mini-batch size 64, and a learning rate of 0.001. Dropouts were applied at all fully connected layers to avoid over-fitting (with probability 0.5 for the firstly fully connected layer and 0.25 for the rest). See Section 6.4.2 for details. Finally, after the entire learning session of a policy, three rollouts were performed using the learned policy for performance evaluation.

6.6 Experimental Results

6.6.1 Online vs Batch Learning

We first study the performance of training a control policy with online and batch imitation learning algorithms. Fig. 6.5 illustrates the vehicle trajectories of different policies.

Table 6.2: Test statistics. Total loss denotes the imitation loss in (6.6), which is the average of the steering and the throttle losses. Completion ratio is defined as the ratio of the traveled time steps to the targeted time steps (3,000). All results here represent the average performance over three independent evaluation trials.

Policy	Avg. speed	Top speed	Training data	Completion ratio	Total loss	Steering / Throttle loss
Expert	6.05 m/s	8.14 m/s	NA	100 %	0	0
Batch	4.97 m/s	5.51 m/s	3000	100 %	0.108	0.092/0.124
Batch	6.02 m/s	8.18 m/s	6000	51 %	0108	0.162/0.055
Batch	5.79 m/s	7.78 m/s	9000	53 %	0.123	0.193/0.071
Batch	5.95 m/s	8.01 m/s	12000	69 %	0.105	0.125/0.083
Online (1 iter)	6.02 m/s	7.88 m/s	6000	100 %	0.090	0.112/0.067
Online (2 iter)	5.89 m/s	8.02 m/s	9000	100 %	0.075	0.095/0.055
Online (3 iter)	6.07 m/s	8.06 m/s	12000	100 %	0.064	0.073/0.055

Due to accumulating errors, the policy trained with batch imitation learning crashed into the lower-left boundary, an area of the state space-action rarely explored in the expert's demonstrations. On the contrary, online imitation learning let the policy learn to successfully cope with corner cases as the learned policy occasionally ventured into new areas of the state-action space.

Fig. 6.6 shows the performance in terms of distance traveled without crashing[5] and Table 6.2 shows the statistics of the experimental results. Overall, DNN policies trained with both online and batch imitation learning algorithms were able to achieve a similar speed as the MPC expert. However, with the same amount of training data, the policies trained with online imitation learning in general outperformed those trained with batch imitation learning. In particular, the policies trained using online imitation learning achieved better performance in terms of both completion ratio and imitation loss. It is worth noting that the traveled distance of the policy learned with a batch of 3,000 samples was longer than that of other batch learning policies. As shown in Table 6.2, this is mainly because this policy achieved better steering performance than throttle performance. As a result, although the

[5]We used the safe control module shown in Fig. 6.2 to manually terminate the rollout when the car crashed into the soft boundary.

vehicle was able to navigate without crashing, it actually traveled at a much slower speed. By contrast, the batch learning policies that used more data had better throttle performance and worse steering performance, resulting in faster speeds but higher chances of crashing.

6.6.2 Deep Neural Network Policy

One main feature of a DNN policy is that it can learn to extract both low-level and high-level features of an image and automatically detect the parts that have greater influence on steering and throttle. We validate this idea by showing in Fig. 6.7 the averaged feature map at each max-pooling layer (see Fig. 6.3), where each pixel represents the averaged unit activation across different filter outputs. We can observe that at a deeper level, the detected salient objects are boundaries of the track and parts of a building. Grass and dirt contribute little to the DNN's output.

We also analyze the importance of incorporating wheel speeds in our task. We compare the performance of the policy based on our DNN policy and a policy based on only the CNN subnetwork (without wheel-speed inputs) in batch imitation learning. The data was collected in accordance with Section 6.5.3. Fig. 6.8 shows the batch imitation learning loss in (6.9) of different network architectures. The full DNN policy in Fig. 6.3 achieved better performance consistently. While images contain position information, it is insufficient to infer velocities. Therefore, we conjecture state-of-the-art CNNs (e.g. [130]) cannot be directly used in this task. By contrast, while without a recurrent architecture, our DNN policy learned to combine wheel speeds in conjunction with CNN to infer hidden state and achieve better performance.

6.7 Summary

We introduce an end-to-end learning system to learn a deep neural network driving policy that maps raw on-board observations to steering and throttle commands by mimicking an expert's behavior. We investigate both online and batch learning frameworks theoretically and empirically and propose an imitation learning system. In real-world experiments, our system was able to perform fast off-road navigation autonomously at an average speed of $\sim$6 m/s and a top speed of $\sim$8 m/s, while only using low-cost monocular camera and wheel speeds sensors. Our current and future work include developing more complex policy representations, such as recurrent neural networks, and to improve robustness to visual distractions.

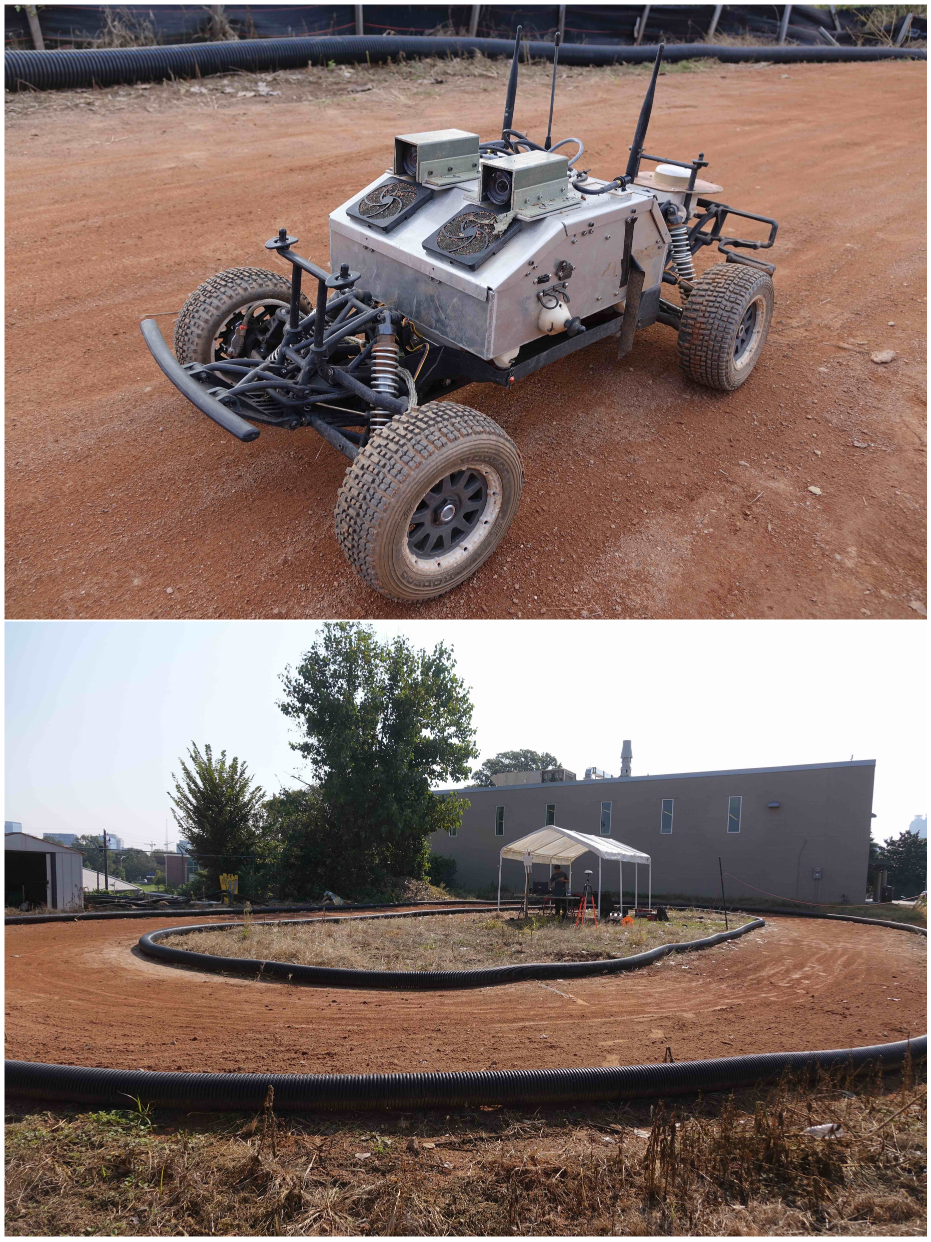
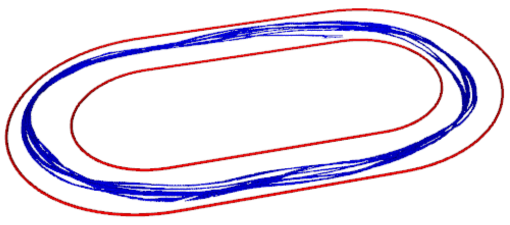
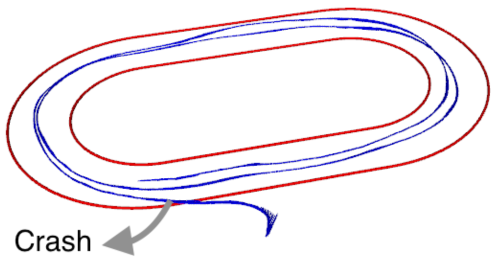


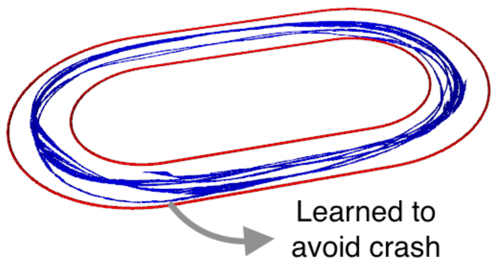
Figure 6.4: The AutoRally car and the test track.



(a) MPC expert



(b) Batch learning



(c) Online learning

Figure 6.5: Examples of vehicle trajectories. (a) Demonstration of the MPC expert. (b) Crashing case when using batch imitation learning. (c) The vehicle avoids crashing when using online imitation learning. Subfigures (b) and (c) depict test runs of the policies after training on 9,000 samples

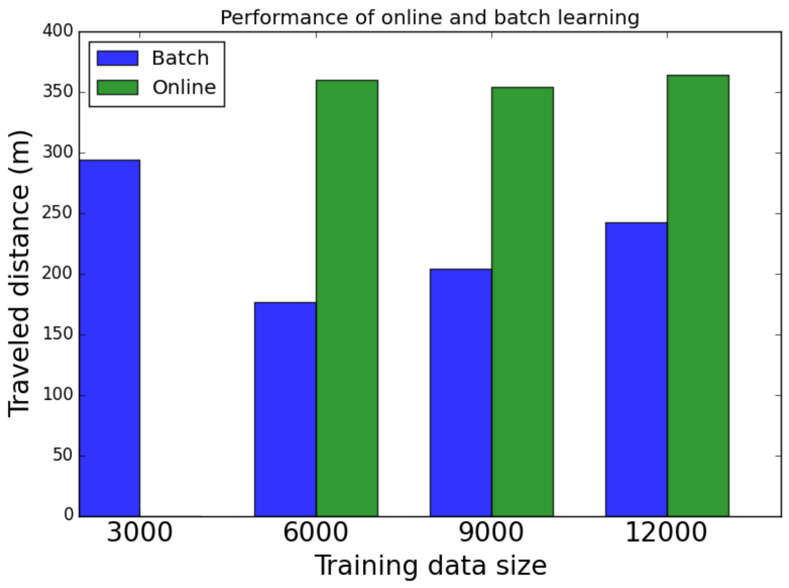
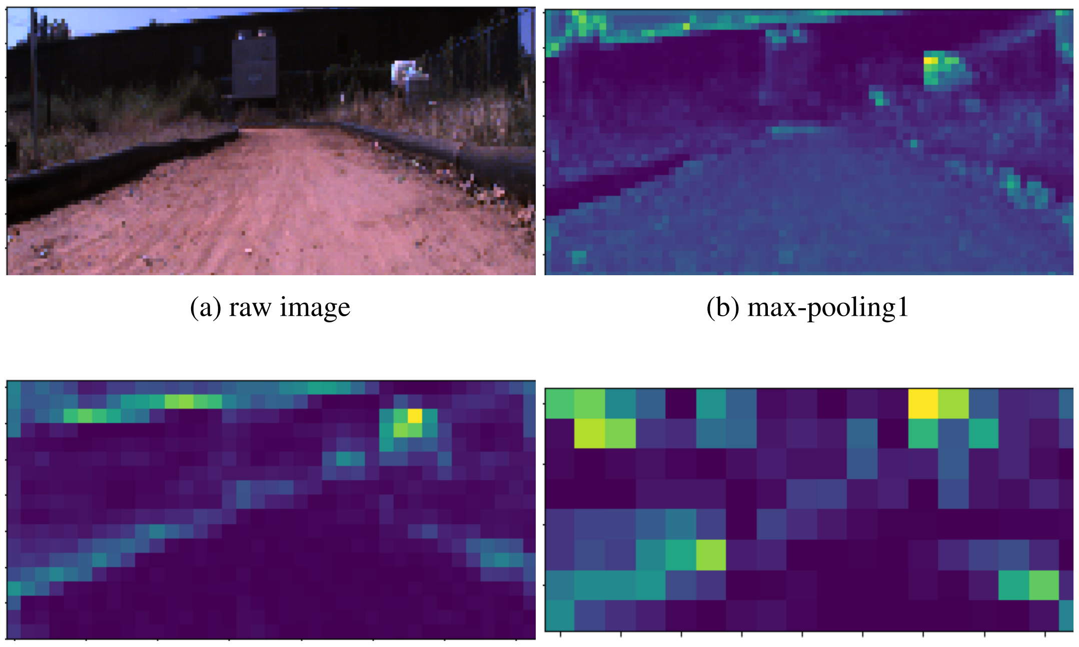


Figure 6.6: Performance of online and batch imitation learning in the distance (meters) traveled without crashing. The policy trained with a batch of 3,000 samples was used to initialize online imitation learning.



(a) raw image (b) max-pooling1

(c) max-pooling2 (d) max-pooling3

Figure 6.7: From left to right: input RGB image, averaged feature maps for each max-pooling layer.

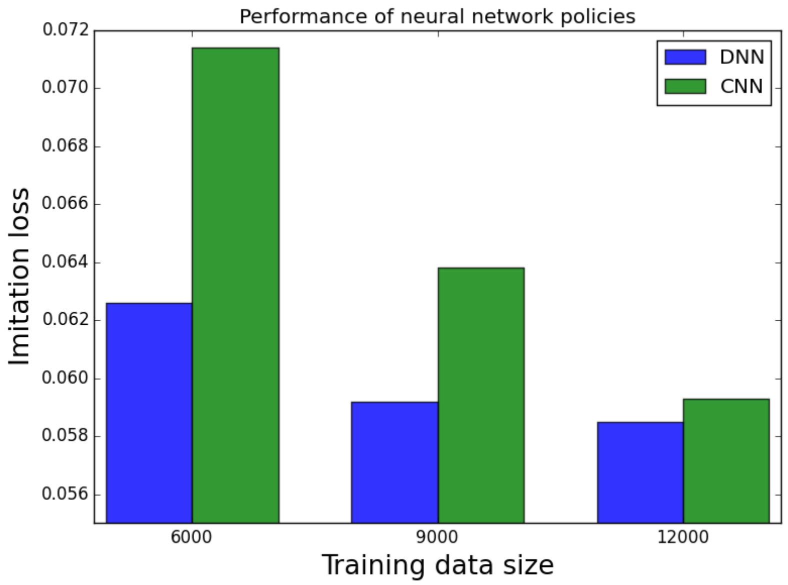


Figure 6.8: Performance comparison between our DNN policy and its CNN sub-network in terms of batch imitation learning loss, where the horizontal axis is the size of data used to train the neural network policies.

CHAPTER 7
CONCLUSIONS

The goal of this thesis is to develop novel methods for controlling robotic systems with unknown or partially known dynamics. There are several major challenges. First, many robotics systems have complex dynamics which are difficult to model. Learning a policy or dynamics model form data requires many interactions with the physical systems, which is impractical. Second, solving nonlinear optimal control problems in continuous domains suffers from the *curse of dimensionality*. Searching an optimal control policy for the whole state space is computationally intractable except for low dimensional problems.

Motivated by these challenges, we proposed *probabilistic trajectory optimization*, a framework for optimal control under unknown or partially known dynamics. Our framework combines the benefits of probabilistic inference and trajectory optimization, namely robustness to modeling errors, and scalability to high-dimensional state and action spaces. More specifically, we learn dynamics models from data using Gaussian processes (GPs), a Bayesian nonparametric approach that provides a superior model expressiveness compared with parametric methods. In terms of control, we developed approaches that perform *Dynamic Programming* approximately using two techniques 1) local approximation of the value function; 2) linearization of the Hamilton-Jacobi-Bellman (HJB) equation. In chapter 3 and 4, we presented algorithms under this framework that perform well in terms of data efficiency and computational efficiency. In chapter 5, we presented fast approximate inference methods that are based on sparse spectrum Gaussian processes (SSGPs). These methods are more efficient and general than their full GP counterparts, and can be directly applied to perform real-time probabilistic trajectory optimization in a receding horizon fashion.

When the system state is not easily accessible, we resort to *imitation learning* where the goal is to train a *learner* with data generated by an *expert*. In chapter 6, we developed an imitation learning system for off-road, high-speed autonomous driving. In our proposed system, the algorithmic expert is the receding horizon probabilistic trajectory optimizer developed in chapter 5. Only this expert has access to the system state provided by a state estimator and expensive sensors, i.e., Global Position System (GPS) and Inertial Measurement Unit (IMU), and a task cost function that depends on the state. The learner is a deep neural network that only has access to cheap sensors, i.e., camera and wheel speed sensor. The trained deep neural network driving policy is able to achieve state-of-the-art experimental results on real high-speed racing tasks at a much lower cost compared to receding horizon control systems.

There are several open problems that are worth further exploration. First, incorporating state constraints into our probabilistic trajectory optimization framework is an interesting direction that could improve its applicability to more complex planning tasks. Second, in approximate inference, a better balance between prediction accuracy and computational efficiency may be achieved by using variational inference techniques. In addition, in order to enforce safety in autonomous driving, a single reactive controller is not sufficient. Predictive modeling techniques may be integrated into an end-to-end driving system.

REFERENCES

[1] N. J. Nilsson, *The quest for artificial intelligence*. Cambridge University Press, 2009.

[2] G. Tesauro, “Temporal difference learning and td-gammon,” *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[3] M. Campbell, A. J. Hoane, and F.-h. Hsu, “Deep blue,” *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press Cambridge, 1998, vol. 135.

[7] G. J. Leishman, *Principles of helicopter aerodynamics with CD extra*. Cambridge university press, 2006.

[8] P. Abbeel, V. Ganapathi, and A. Y. Ng, “Learning vehicular dynamics, with application to modeling helicopters,” in *Advances in Neural Information Processing Systems*, 2006, pp. 1–8.

[9] P. Abbeel, A. Coates, T. Hunter, and A. Y. Ng, “Autonomous autorotation of an rc helicopter,” in *Experimental Robotics*, Springer, 2009, pp. 385–394.

[10] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*, 2. Athena scientific Belmont, MA, 1995, vol. 1.

[11] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, “Planning and acting in partially observable stochastic domains,” *Artificial intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[12] M. Hauskrecht, "Value-function approximations for partially observable markov decision processes," *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, 2000.

[13] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning.," in *HotNets*, 2016, pp. 50–56.

[14] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[15] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, 2006, vol. 1.

[16] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Learning, planning, and control for quadruped locomotion over challenging terrain," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 236–258, 2011.

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] J. Michels, A. Saxena, and A. Y. Ng, "High speed obstacle avoidance using monocular vision and reinforcement learning," in *International Conference on Machine learning*, 2005, pp. 593–600.

[20] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *IEEE International Conference on Robotics and Automation*, 2016, pp. 1433–1440.

[21] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. Rehg, B. Boots, and E. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *IEEE Conference on Robotics and Automation*, 2017.

[22] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[23] R Bellman, "Dynamic programming princeton university press princeton," *New Jersey Google Scholar*, 1957.

[24] L. S. Pontryagin, *Mathematical theory of optimal processes*. CRC Press, 1987.

[25] R. F. Stengel, *Optimal control and estimation*. Courier Corporation, 2012.

[26] D. Jacobson and D. Mayne, "Differential dynamic programming," 1970.

[27] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference, 2005*, IEEE, 2005, pp. 300–306.

[28] E. Theodorou, Y. Tassa, and E. Todorov, "Stochastic differential dynamic programming," in *American Control Conference (ACC), 2010*, IEEE, 2010, pp. 1125–1132.

[29] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming.," in *NIPS*, 2007.

[30] J. Morimoto and C. Atkeson, "Minimax differential dynamic programming: An application to robust biped walking," in *NIPS*, 2002, pp. 1539–1546.

[31] Y. Pan, K. Bakshi, and E. Theodorou, "Robust trajectory optimization: A cooperative stochastic game theoretic approach," in *Proceedings of Robotics: Science and Systems*, 2015.

[32] W. Fleming, "Exit probabilities and optimal stochastic control," *Applied Math. Optim*, vol. 9, pp. 329–346, 1971.

[33] W. H. Fleming and H. M. Soner, *Controlled Markov processes and viscosity solutions*, 1st, ser. Applications of mathematics. New York: Springer, 1993.

[34] H. J. Kappen, "Linear theory for control of nonlinear stochastic systems," *Phys Rev Lett*, vol. 95, pp. 200–201, 2005.

[35] ——, "Path integrals and symmetry breaking for optimal control theory," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 11, P11011, 2005.

[36] ——, "An introduction to stochastic control theory, path integrals and reinforcement learning," *AIP Conference Proceedings*, vol. 887, no. 1, 2007.

[37] S. Thijssen and H. J. Kappen, "Path integral control and state-dependent feedback," *Phys. Rev. E*, vol. 91, p. 032 104, 3 2015.

[38] E. Todorov, "Efficient computation of optimal actions," *Proceedings of the national academy of sciences*, vol. 106, no. 28, pp. 11 478–11 483, 2009.

[39] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[40] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, ACM, 2012, pp. 281–288.

[41] K. Rawlik, M. Toussaint, and S. Vijayakumar, "Path integral control by reproducing kernel hilbert space embedding," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, ser. IJCAI'13, 2013, pp. 1628–1634.

[42] Y. Pan and E. Theodorou, "Nonparametric infinite horizon kullback-leibler stochastic control," in *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, IEEE, 2014, pp. 1–8.

[43] V. Gómez, H. Kappen, J. Peters, and G. Neumann, "Policy search for path integral control," in *Machine Learning and Knowledge Discovery in Databases*, Springer, 2014, pp. 482–497.

[44] Y. Pan, E. Theodorou, and M. Kontitsis, "Sample efficient path integral control under uncertainty," in *Advances in Neural Information Processing Systems*, 2015, pp. 2314–2322.

[45] K. Dvijotham and E Todorov, "Linearly solvable optimal control," *Reinforcement learning and approximate dynamic programming for feedback control*, pp. 119–141, 2012.

[46] M. Kac, "On distributions of certain wiener functionals," *Transactions of the American Mathematical Society*, vol. 65, no. 1, pp. 1–13, 1949.

[47] C. Williams and C. Rasmussen, *Gaussian processes for machine learning*. MIT Press, 2006.

[48] J. Q. Candela, A. Girard, J. Larsen, and C. E. Rasmussen, "Propagation of uncertainty in bayesian kernel models-application to multiple-step ahead forecasting," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003.

[49] A. Girard, C. Rasmussen, J. Quinonero-Candela, and R. Murray-Smith, "Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting," in *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[50] M. Deisenroth, D. Fox, and C. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transsactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 75–90, 2015.

[51] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[52] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *AAAI*, 2010.

[53] G. Neumann, "Variational inference for policy search in changing situations," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 817–824.

[54] C. Atkeson and J. Santamaria, "A comparison of direct and model-based reinforcement learning," in *In International Conference on Robotics and Automation*, Citeseer, 1997.

[55] M. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics.," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[56] D. Bertsekas and J. Tsitsiklis, "Neuro-dynamic programming (optimization and neural computation series, 3)," *Athena Scientific*, vol. 7, pp. 15–23, 1996.

[57] A. Barto, W. Powell, J. Si, and D. Wunsch, "Handbook of learning and approximate dynamic programming," 2004.

[58] P. Abbeel, A. Coates, M. Quigley, and A. Y Ng, "An application of reinforcement learning to aerobatic helicopter flight," *NIPS*, vol. 19, p. 1, 2007.

[59] D. Mitrovic, S. Klanke, and S. Vijayakumar, "Adaptive optimal feedback control with learned internal dynamics models," in *From Motor Learning to Interaction Learning in Robots*, Springer, 2010, pp. 65–84.

[60] C. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning.," in *NIPS*, vol. 4, 2004, p. 1.

[61] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Local gaussian process regression for real time online model learning," in *NIPS*, 2008, pp. 1193–1200.

[62] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2677–2682.

[63] ——, "Online kernel-based learning for task-space tracking robot control," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 9, pp. 1417–1425, 2012.

[64] M. Deisenroth and C. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 465–472.

[65] P. Hemakumara and S. Sukkarieh, "Learning uav stability and control derivatives using gaussian processes," *IEEE Transactions on Robotics*, vol. 29, pp. 813–824, 2013.

[66] G. Chowdhary, H. Kingravi, J. P. How, P. Vela, *et al.*, "Bayesian nonparametric adaptive control using gaussian processes," *IEEE Transactions on Neural Networks and Learning Systems,*, vol. 26, no. 3, pp. 537–550, 2015.

[67] P. Dallaire, C. Besse, S. Ross, and B. Chaib-draa, "Bayesian reinforcement learning in continuous pomdps with gaussian processes," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2009, pp. 2604–2609.

[68] J. Boedecker, J. Springenberg, J. Wulfing, and M. Riedmiller, "Approximate real-time optimal control based on sparse gaussian process models," in *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, IEEE, 2014, pp. 1–8.

[69] M. Kuss, "Gaussian process models for robust regression, classification, and reinforcement learning," PhD thesis, Technische Universität, 2006.

[70] M. P. Deisenroth, M. F. Huber, and U. D. Hanebeck, "Analytic moment-based gaussian process filtering," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 225–232.

[71] J. Vinogradska, B. Bischoff, D. Nguyen-Tuong, H. Schmidt, A. Romer, and J. Peters, "Stability of controllers for gaussian process forward models," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 545–554.

[72] A. O'Hagan and J. Kingman, "Curve fitting and optimal design for prediction," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–42, 1978.

[73] T. Wu and J. Movellan, "Semi-parametric gaussian process for robot system identification," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2012, pp. 725–731.

[74] E. Schearer, Y.-W. Liao, E. Perreault, M. Tresch, W. Memberg, R. Kirsch, and K. Lynch, "Identifying inverse human arm dynamics using a robotic testbed," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, 2014, pp. 3585–3591.

[75] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, P. LEcuyer, *et al.*, "The cross-entropy method for optimization," *Machine Learning: Theory and Applications, V. Govindaraju and CR Rao, Eds, Chennai: Elsevier BV*, vol. 31, pp. 35–59, 2013.

[76] P. Whittle, *Risk-sensitive optimal control*. John Wiley & Son Ltd, 1990.

[77] Y. Tassa, N. Mansard, and E. Todorov, “Control-limited differential dynamic programming,” *ICRA 2014*,

[78] D. Bertsekas, “Projected newton methods for optimization problems with simple constraints,” *SIAM Journal on control and Optimization*, vol. 20, no. 2, pp. 221–246, 1982.

[79] L. Liao and C. Shoemaker, “Convergence in unconstrained discrete-time differential dynamic programming,” *IEEE Transactions on Automatic Control*, vol. 36, no. 6, pp. 692–706, 1991.

[80] L.-Z. Liao, *Global Convergence of Differential Dynamic Programming and Newton’s Method for Discrete-time Optimal Control*. Technical report, 1996.

[81] G. I. Boutselis, Y. Pan, G. De La Tore, and E. A. Theodorou, “Stochastic trajectory optimization for mechanical systems with parametric uncertainties,” *ArXiv preprint arXiv:1705.05506*, 2017.

[82] J. Van Den Berg, S. Patil, and R. Alterovitz, “Motion planning under uncertainty using iterative local optimization in belief space,” *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1263–1278, 2012.

[83] L. Csató and M. Opper, “Sparse on-line gaussian processes,” *Neural Computation*, vol. 14, no. 3, pp. 641–668, 2002.

[84] E. Snelson and Z. Ghahramani, “Sparse gaussian processes using pseudo-inputs,” *NIPS*, vol. 18, p. 1257, 2006.

[85] Y. Pan and E. Theodorou, “Probabilistic differential dynamic programming,” in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 1907–1915.

[86] T. N. Hoang, K. H. Low, P. Jaillet, and M. Kankanhalli, “Nonmyopic-bayes-optimal active learning of gaussian processes,” in *Proc. ICML*, 2014, pp. 739–747.

[87] C. K. Ling, K. H. Low, and P. Jaillet, “Gaussian process planning with lipschitz continuous reward functions: Towards unifying bayesian optimization, active learning, and beyond,” *Proc. AAAI*, 2016.

[88] J. Van Den Berg, S. Patil, and R. Alterovitz, “Motion planning under uncertainty using differential dynamic programming in belief space,” in *Intl Symposium on Robotics Research*, 2011.

[89] J. Morimoto, G. Zeglin, and C. G Atkeson, “Minimax differential dynamic programming: Application to a biped walking robot,” in *Proceedings of 2003 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS 2003)., IEEE, vol. 2, 2003, pp. 1927–1932.

[90] F. Meier, P. Hennig, and S. Schaal, "Incremental local gaussian regression," in *Advances in Neural Information Processing Systems*, 2014, pp. 972–980.

[91] M. Lázaro-Gredilla, J. Quiñonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal, "Sparse spectrum gaussian process regression," *The Journal of Machine Learning Research*, vol. 99, pp. 1865–1881, 2010.

[92] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in neural information processing systems (NIPS)*, 2005, pp. 1257–1264.

[93] Y. Pan, X. Yan, E. Theodorou, and B. Boots, "Adaptive probabilistic trajectory optimization via efficient approximate inference," *ArXiv preprint arXiv:1608.06235*, 2016.

[94] Y. Pan, X. Yan, E. A. Theodorou, and B. Boots, "Prediction under uncertainty in sparse spectrum Gaussian processes with applications to filtering and control," in *International Conference on Machine Learning*, 2017, pp. 2760–2768.

[95] A. Y. Ng and M. Jordan, "Pegasus: A policy search method for large mdps and pomdps," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 2000, pp. 406–415.

[96] Y. Pan, K. Saigol, and E. A. Theodorou, "Belief space stochastic control under unknown dynamics," in *American Control Conference (ACC), 2017*, IEEE, 2017, pp. 3764–3770.

[97] M. Gandhi, Y. Pan, and E. Theodorou, "Pseudospectral model predictive control under partially learned dynamics," *ArXiv preprint arXiv:1702.04800*, 2017.

[98] Y. Pan, X. Yan, E. Theodorou, and B. Boots, "Scalable reinforcement learning via trajectory optimization and approximate gaussian process regression," in *NIPS Workshop on Advances in Approximate Bayesian Inference*, 2015.

[99] E. Theodorou and E. Todorov, "Relative entropy and free energy dualities: Connections to path integral and kl control.," in *51st IEEE Conference on Decision and Control*, 2012, pp. 1466–1473.

[100] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 1071–1079.

[101] S. Levine and V. Koltun, “Learning complex neural network policies with trajectory optimization,” in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 829–837.

[102] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization,” *ArXiv preprint arXiv:1502.05477*, 2015.

[103] P. Hennig, “Optimal reinforcement learning for gaussian systems,” in *Advances in Neural Information Processing Systems (NIPS)*, 2011, pp. 325–333.

[104] E. Todorov, “Compositionality of optimal control laws,” in *Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 1856–1864.

[105] M. Deisenroth, P. Englert, J. Peters, and D. Fox, “Multi-task policy search for robotics,” in *Proceedings of 2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[106] A. Rahimi and B. Recht, “Random features for large-scale kernel machines,” in *Advances in neural information processing systems*, 2007, pp. 1177–1184.

[107] A. Gijsberts and G. Metta, “Real-time model learning using incremental sparse spectrum gaussian process regression,” *Neural Networks*, vol. 41, pp. 59–69, 2013.

[108] W. Rudin, *Fourier analysis on groups*. New York: Interscience Publishers, 1962.

[109] E. A Theodorou, “Nonlinear stochastic control and information theoretic dualities: Connections, interdependencies and thermodynamic interpretations,” *Entropy*, vol. 17, no. 5, pp. 3352–3375, 2015.

[110] N. Hansen and A. Ostermeier, “Completely derandomized self-adaptation in evolution strategies,” *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[111] F. Stulp, E. Theodorou, and S. Schaal, “Reinforcement learning with sequences of motion primitives for robust manipulation,” *IEEE Transactions on Robotics*, vol. 28, no. 6, pp. 1360–1370, 2012.

[112] J. Ko and D. Fox, “Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models,” *Autonomous Robots*, vol. 27, no. 1, pp. 75–90, 2009.

[113] M. P. Deisenroth, R. D. Turner, M. F. Huber, U. D. Hanebeck, and C. E. Rasmussen, “Robust filtering and smoothing with Gaussian processes,” *IEEE Transactions on Automatic Control*, vol. 57, no. 7, pp. 1865–1871, 2012.

[114] C. Archambeau, D. Cornford, M. Opper, and J. Shawe-Taylor, “Gaussian process approximations of stochastic differential equations.,” *Gaussian Processes in Practice*, vol. 1, pp. 1–16, 2007.

[115] B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M.-F. F. Balcan, and L. Song, “Scalable kernel methods via doubly stochastic gradients,” in *Advances in Neural Information Processing Systems*, 2014, pp. 3041–3049.

[116] X. Yan, B. Xie, L. Song, and B. Boots, “Large-scale Gaussian process regression via doubly stochastic gradient descent,” *The ICML Workshop on Large-Scale Kernel Learning*, 2015.

[117] M. K. Titsias, “Variational learning of inducing variables in sparse gaussian processes.,” in *AISTATS*, vol. 5, 2009, pp. 567–574.

[118] C.-A. Cheng and B. Boots, “Incremental variational sparse Gaussian process regression,” in *Proceedings of Advances in Neural Information Processing Systems 30 (NIPS)*, 2016.

[119] A. Kupcsik, M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann, “Model-based contextual policy search for data-efficient generalization of robot skills,” *Artificial Intelligence*, 2014.

[120] T. P. Minka, “A family of algorithms for approximate bayesian inference,” PhD thesis, Massachusetts Institute of Technology, 2001.

[121] L. Ljung, *System identification*. Springer, 1998, ch. 11, p. 300.

[122] A. Yamaguchi and C. G. Atkeson, “Neural networks and differential dynamic programming for reinforcement learning problems,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 5434–5441.

[123] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, “Isam2: Incremental smoothing and mapping using the bayes tree,” *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, 2012.

[124] E. Velenis, E. Frazzoli, and P. Tsiotras, “Steady-state cornering equilibria and stabilisation for a vehicle during extreme operating conditions,” *International Journal of Vehicle Autonomous Systems*, vol. 8, no. 2-4, pp. 217–241, 2010.

[125] S. Vijayakumar, A. D’souza, and S. Schaal, “Incremental online learning in high dimensions,” *Neural computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[126] C. Williams, S. Klanke, S. Vijayakumar, and K. M. Chai, "Multi-task gaussian process learning of robot inverse dynamics," in *Advances in Neural Information Processing Systems*, 2009, pp. 265–272.

[127] P. Boyle and M. Frean, "Dependent gaussian processes," in *Advances in neural information processing systems*, 2005, pp. 217–224.

[128] M. A. Alvarez, L. Rosasco, N. D. Lawrence, *et al.*, "Kernels for vector-valued functions: A review," *Foundations and Trends® in Machine Learning*, vol. 4, no. 3, pp. 195–266, 2012.

[129] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, Jan. 2016.

[130] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *ArXiv preprint arXiv:1704.07911*, 2017.

[131] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, 1989, pp. 305–313.

[132] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick, "Learning a deep neural net policy for end-to-end control of autonomous vehicles," in *IEEE American Control Conference*, 2017.

[133] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun, "Off-road obstacle avoidance through end-to-end learning," in *Advances in Neural Information Processing Systems*, 2006, pp. 739–746.

[134] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," *ArXiv preprint arXiv:1605.06450*, 2016.

[135] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, vol. 1, 2011, p. 6.

[136] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Model predictive control with a cnn cost model," *ArXiv preprint arXiv:1707.05303*, 2017.

[137] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *International Conference on Machine Learning*, vol. 2, 2002, pp. 267–274.

[138] S. Shalev-Shwartz *et al.*, "Online learning and online convex optimization," *Foundations and Trends® in Machine Learning*, vol. 4, no. 2, pp. 107–194, 2012.

[139] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 1765–1772.

[140] A. L. Gibbs and F. E. Su, "On choosing and bounding probability metrics," *International Statistical Review*, vol. 70, no. 3, pp. 419–435, 2002.

[141] M. Laskey, C. Chuck, J. Lee, J. Mahler, S. Krishnan, K. Jamieson, A. Dragan, and K. Goldberg, "Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations," *ArXiv preprint arXiv:1610.00850*, 2016.

[142] N. Koenig and A. H. Design, "Use paradigms for gazebo, an open-source multi-robot simulator ieee," in *IEEE International Conference on Intelligent Robots and Systems*, 2004.

[143] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming," in *Advances in Neural Information Processing Systems*, 2008, pp. 1465–1472.

[144] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ArXiv preprint arXiv:1412.6980*, 2014.