Goal-Driven Dynamics Learning via Bayesian Optimization

Teddey Xiao

Acknowledgement

I would like to thank my advisor Professor Claire Tomlin of the Department of EECS at the University of California, Berkeley. Prof. Tomlin's unwavering kindness and support guided me throughout the entire process of academic research. More importantly, Prof. Tomlin provided guidance on whatever topic I needed advice on, be it research, coursework, or career advice. I would also like to thank my mentor Somil Bansal who supported me throughout the Master's program. Somil was a great guide who assisted me with any questions or concerns I had. Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them.

Goal-Driven Dynamics Learning via Bayesian Optimization

by

Ted Xiao

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Claire Tomlin, Chair
Professor Sergey Levine

Spring 2017

The thesis of Ted Xiao, titled Goal-Driven Dynamics Learning via Bayesian Optimization, is approved:

Chair	____________________	Date	__________
	____________________	Date	__________

University of California, Berkeley

Goal-Driven Dynamics Learning via Bayesian Optimization

Abstract

Goal-Driven Dynamics Learning via Bayesian Optimization

by

Ted Xiao

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Claire Tomlin, Chair

Robotic systems are becoming increasingly complex and commonly act in poorly understood environments where it is extremely challenging to model or learn their true dynamics. Therefore, it might be desirable to take a task-specific approach, wherein the focus is on explicitly learning the dynamics model which achieves the best control performance for the task at hand, rather than learning the true system dynamics. In this work, we use Bayesian optimization in an active learning framework where a locally linear dynamics model is learned with the intent of maximizing the control performance, and then used in conjunction with optimal control schemes to efficiently design a controller for a given task. This model is updated directly in an iterative manner based on the performance observed in experiments on the physical system until a desired performance is achieved. We demonstrate the efficacy of the proposed approach through simulations and real experiments on a quadrotor testbed.

Contents

List of Figures

List of Tables

Acknowledgments

I would first and foremost like to thank my advisor Professor Claire Tomlin of the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. Prof. Tomlin's unwavering kindness and support guided me throughout the entire process of academic research. The door to Prof. Tomlin's office was always open whenever I ran into a trouble spot or had questions about my research or writing. More importantly, Prof. Tomlin provided guidance on whatever topic I needed advice on, be it research, coursework, or career advice.

I like to thank my mentor Somil Bansal who supported me throughout the Master's program. Somil was not only a brilliant driver behind this work, but also a great guide who assisted me with any questions or concerns I had. I spent countless hours chatting with Somil and taking away so much insight about machine learning and control systems. I owe a large amount of my understanding of control theory and robotics to Somil.

In addition, I would like to thank Roberto Calandra for his contribution to the theory and background work for this project.

I would also like to acknowledge Assistant Professor Sergey Levine of the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley as the second reader of this thesis, and I am gratefully indebted to him for his very valuable comments on this thesis.

This work would not have been possible without Somil, Roberto, Sergey, or Claire.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them. Thank you.

Ted Xiao, May 9th, 2017

Chapter 1

Introduction

Complex robotics systems have been the subject of much research and study. Beyond theoretical insight into control and learning models, there seems to be much promise in bringing practical wide-scale robotics to society in the foreseeable future. However, there are still many areas that require more study before applications are deployed in a robust and safe manner. As learning and control approaches are gradually being brought from controlled lab conditions to uncontrolled real-world scenarios, there is a need for methods that work well in complex environments that may have unknown dynamics. For instance, real-world conditions involve environments without accurate *a priori* models, such as situations with human interaction.

Advances in machine learning, reinforcement learning, and optimal control have all contributed towards learning and controlling unknown systems. In general, there are two main regimes for approaching this problem: model-based and model-free learning approaches. In model-based methods, the system dynamics are modelled and then used in conjunction with optimal control methods to produce policies or controllers. The dynamics can be modelled through a variety of methods, such as deep learning [3] or traditional systems identification [4]. In model-free methods, the policy is directly learned. Both methods have their benefits, but but we believe that there is significant promise in the intersection between model-free and model-based methods, where existing theory in optimal control can be leveraged while still using the benefits of robust deep learning methods. In fact, or scenarios that instead have abundance of data, deep learning may be applied [5]. In this work, however, we focus on a model-based methods that utilizes machine learning in the context of a very data-efficient regime. The goal of this work is not only to propose a framework for dynamics and control in low-data contexts, but to also provide a review into related methods and approaches in this subsection of model-based learning models.

Given the system dynamics, optimal control schemes such as LQR, MPC, and feedback linearization can efficiently design a controller that maximizes a performance criterion. However, depending on the system complexity, it can be quite challenging to model its true dynamics. However, for a given task, a globally accurate dynamics model is not always necessary to design a controller. Often, partial knowledge of the dynamics is sufficient, e.g., for trajectory tracking purposes a local linearization of a non-linear system is often sufficient. In this paper we argue that, for complex systems, it might be preferable to adapt the controller design process for the specific task, using a learned system dynamics model sufficient to achieve the desired performance.

We propose Dynamics Optimization via Bayesian Optimization (aDOBO), a Bayesian Optimization (BO) based active learning framework to learn the dynamics model that achieves the best performance for a given task based on the performance observed in experiments on the physical system. This active learning framework takes into account all past experiments and suggests the next experiment in order to learn most about the relationship between the performance criterion and the model parameters. Particularly important for robotic systems is the use of data-efficient approaches, where only few physical experiments are needed to obtain improved performance.

To make sure our approach is data-efficient, we learn a *locally linear* system model which we use to design the controller, as opposed to learning a global model or learning a control policy directly. The underlying hypothesis of this approach is that a good local model, in conjunction with well - optimal control schemes, can be used to design a controller more efficiently, and can capture a richer controller space. Specifically, we use BO to optimize the dynamics model with respect to the desired task, where the dynamics model is updated after every experiment so as to maximize the performance on the physical system. BO may intuitively be a good fit for this task, because BO is an approach that optimizes a performance criterion while keeping the number of evaluations of the physical system small [6]. A flow diagram of our framework is shown in Figure 1.1. The current locally linear dynamics model, together with the cost function (also referred to as performance criterion), are used to design a controller with an appropriate optimal control scheme. The cost (or performance) of the controller is evaluated in closed-loop operation with the actual (unknown) physical plant. BO uses this performance information to iteratively update the dynamics model to improve the performance. This procedure corresponds to optimizing the (locally linear) system dynamics with the purpose of maximizing the performance of the final controller. Hence, unlike traditional system identification approaches, our approach does not necessarily correspond to finding the most accurate dynamics model, but rather the model yielding the best controller performance when provided to the optimal control method used. An interesting question to study in the context of this method is the degree to which the resulting model actually corresponds to the true dynamics of the system. We study this in Section 4.1.

Traditional system identification approaches are divided into two stages: 1) creating a dynamics model by minimizing some prediction error (e.g., using least squares) 2) using this dynamics model to generate an appropriate controller. In this approach, modeling the dynamics can be considered an offline process as there is no information flow between the two design stages. In online methods, the dynamics model is instead iteratively updated using new data collected by evaluating the controller [7]. Our approach is an online method. Both for the online and the offline cases, creating a dynamics model-based only on minimizing the prediction error can introduce sufficient inaccuracies to lead to suboptimal control perfor-

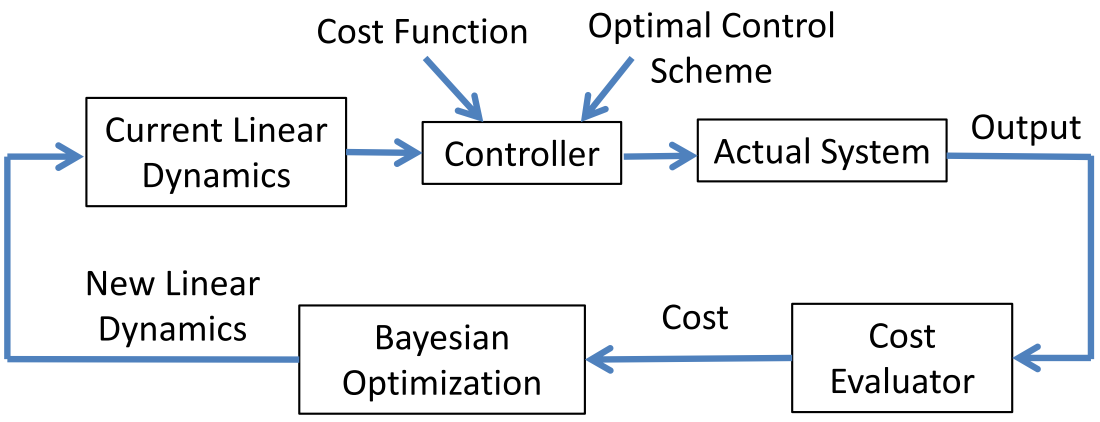


Figure 1.1: aDOBO: A Bayesian optimization-based active learning framework for optimizing the dynamics model for a given cost function.

mance [8]. Using machine learning techniques, such as Gaussian processes, does not alleviate this issue [9]. Instead, the authors in [8] proposed to optimize the dynamics model directly with respect to the controller performance, but since the dynamics model is optimized offline, the resultant model is not necessarily optimal for the actual system. We instead explicitly find the dynamics model that produces the best control performance for the *actual* system.

Previous studies addressed the problem of optimizing a controller using BO. In [1, 10, 11] authors tuned the penalty matrices in an LQR problem for performance optimization. Parameters of a linear feedback controller are learned in [2] using BO. Although interesting results emerge from these studies, it is not clear how these methods perform for non-quadratic cost functions. Moreover, when an accurate system model is not available, tuning penalty matrices may not achieve the desired performance. Our approach overcomes these challenges as it does not rely on an accurate system dynamics model or impose any linear structure on the controller. In fact, aDOBO can easily design non-linear controllers as well (see Sec. 3.1). For example, for non-quadratic convex cost functions, we can use MPC as the optimal control scheme to design non-linear controllers and can automatically design a controller even when the system dynamics are unknown and/or the performance criterion is not quadratic.

The problem of updating a system model to improve control performance is also related to adaptive control, where the model parameters are identified from sensor data, and subsequently the updated model is used to design a controller (see [12–16]). However, in adaptive control, the model parameters are generally updated to get a good prediction model and not necessarily to maximize the controller performance. In contrast, we explicitly take into account the observed performance and search for the model that achieves the highest performance.

To the best of our knowledge, this is the first method that optimize a dynamics model to maximize the control performance on the actual system. Our approach does not require the prior knowledge of an accurate dynamics model, nor of its parameterized form. Instead, the dynamics model is optimized, in an active learning setting, directly with respect to the desired cost function using data-efficient BO. The contribution of this paper is to present an automatic approach to controller design. This approach does not require the prior knowledge of an accurate dynamics model, nor of its parameterized form. Instead, the dynamics model is optimized, in an active learning setting, directly with respect to the desired cost function using data-efficient Bayesian optimization. The idea of using a model that is not the most likely model, but rather the one that achieves the best expected reward has been previously proposed in [8]. Although similar in spirit, their approach relies on a gradient-descent optimizer which requires numerically approximating the gradient by central differences, resulting in an increase in the number of experiments required. Our approach instead, makes use of a global zero-order optimizer (i.e., it does not need gradients) and can therefore be more sample efficient. We further compare some of these approaches with the proposed approach in Section 5. To summarize, our main contributions in this work are:

- to efficiently and automatically design a controller for general performance criterion, even when the system dynamics are unknown;
- to compare different automatic controller design approaches and highlight their relative advantages and limitations.

This report is structured into seven main sections. In this *Introduction* chapter, we introduce the domain and viewpoint from which we propose our method. In the *Preliminaries* chapter, we define the learning problem and give a background on Gaussian Processes and Bayesian Optimization, which are key components of our method. In the *Solution* chapter, we introduce our framework aDOBO. Then, in the *Simulations* chapter, we explore numerical simulations demonstrating the effectiveness of our method. In the *Comparison with Other Methods* chapter, we weigh the benefits and costs of our method in comparison with other state of the art approaches. Next, in the *Quadrotor Position Tracking Experiments* chapter, we demonstrate a real-world experiment of our method showing significant results. Finally, we finish with the *Conclusion and Future Work* chapter, wrapping up our contributions with this work.

This work is performed jointly with Somil Bansal, Roberto Calandra, Sergey Levine, and Claire Tomlin and was submitted in March 2017 to the IEEE Conference on Decision and Control. A preprint appears in [17].

Chapter 2

Preliminaries

2.1 Problem Formulation

Consider an unknown, stable, discrete-time, potentially non-linear, dynamical system

$$z_{k+1} = f(z_k, u_k), \quad k \in \{0, 1, \ldots, N-1\}\,, \tag{2.1}$$

where $z_k \in \mathbb{R}^{n_x}$ and $u_k \in \mathbb{R}^{n_u}$ denote the system state and the control input at time k respectively. Given an initial state z_0, the objective is to design a controller that minimizes the cost function J subject to the dynamics in (2.1)

$$\begin{aligned} J_0^* = \min_{\mathbf{u}_0^{N-1}} J_0(\mathbf{z}_0^N, \mathbf{u}_0^{N-1}) = \min_{\mathbf{u}_0^{N-1}} &\sum_{i=0}^{N-1} l(z_i, u_i) + g(z_N, u_N)\,, \\ \text{subject to} \quad z_{k+1} =& f(z_k, u_k)\,, \end{aligned} \tag{2.2}$$

where $\mathbf{z}_i^N := (z_i, z_{i+1}, \ldots, z_N)$. $\mathbf{u}_i^{N-1}$ is similarly defined. One of the key challenges in designing such a controller is the modeling of the unknown system dynamics in (2.1). In this work, we model (2.1) as a linear time-invariant (LTI) system with system matrices (A_θ, B_θ). The system matrices are parametrized by $\theta \in \mathcal{M} \subseteq \mathbb{R}^d$, which is to be varied during the learning procedure. For a given θ and the current system state z_k, let $\pi_k(z_k, \theta)$ denote the optimal control sequence for the *linear* system (A_θ, B_θ) for the horizon $\{k, k+1, \ldots, N\}$

$$\begin{aligned} \pi_k(z_k, \theta) := \bar{\mathbf{u}}_k^{N-1} =& \arg\min_{\mathbf{u}_k^{N-1}} J_k(\mathbf{z}_k^N, \mathbf{u}_k^{N-1})\,, \\ \text{subject to} \quad z_{j+1} =& A_\theta z_j + B_\theta u_j. \end{aligned} \tag{2.3}$$

The key difference between (2.2) and (2.3) is that the controller is designed for the parameterized linear system as opposed to the true system. As θ is varied, different matrix pairs (A_θ, B_θ) are obtained, which result in different controllers $\pi(\cdot, \theta)$. Our aim is to find, among all linear models, the linear model $(A_{\theta^*}, B_{\theta^*})$ whose controller $\pi(\cdot, \theta^*)$ minimizes J_0 (ideally achieves J_0^*) for the *actual system*, i.e.,

$$\begin{aligned} \theta^* =& \arg\min_{\theta \in \mathcal{M}} J_0(\mathbf{z}_0^N, \mathbf{u}_0^{N-1}) \,, \\ \text{subject to } & \; z_{k+1} = f(z_k, u_k) \,, \quad u_k = \pi_k^1(z_k, \theta), \end{aligned} \tag{2.4}$$

where $\pi_k^1(z_k, \theta)$ denote the 1st control in the sequence $\pi_k(z_k, \theta)$. To make the dependence on θ explicit, we refer to J_0 in (2.4) as $J(\theta)$ here on. Note that $(A_{\theta^*}, B_{\theta^*})$ in (2.4) may not correspond to an actual linearization of the system, but simply to the linear model that gives the best performance on the actual system when its optimal controller is applied in a *closed-loop* fashion on the actual physical plant.

We choose LTI modeling to reduce the number of parameters used to represent the system, and make the dynamics learning process data efficient. Linear modeling also allows to efficiently design the controller in (2.3) for general cost functions (e.g., using MPC for any convex cost J). In general, the effectiveness of linear modeling depends on both the system and the control objective. If f is linear, a linear model is trivially sufficient for any control objective. If f is non-linear, a linear model may not be sufficient for all control tasks; however, for regulation and trajectory tracking tasks, a linear model is often adequate (see Sec. 4.1). A linear parameterization is also used in adaptive control for similar reasons [16]. Nevertheless, the proposed framework can handle more general model classes as long as the optimal control problem in (2.3) can be solved for that class.

Since f is unknown, the shape of the cost function, $J(\theta)$, in (2.4) is unknown. The cost is thus evaluated empirically in each experiment, which is often expensive as it involves conducting an experiment. Thus, the goal is to solve the optimization problem in (2.4) with as few evaluations as possible. In this paper, we do so via BO.

2.2 Background

In order to optimize (A_θ, B_θ), we use BO. In this section, we briefly introduce Gaussian processes and BO.

Gaussian Process (GP)

Since the function $J(\theta)$ in (2.4) is unknown a priori, we use nonparametric GP models to approximate it over its domain $\mathcal{M}$. GPs are a popular choice for probabilistic non-parametric regression, where the goal is to find a nonlinear map, $J(\theta) : \mathcal{M} \to \mathbb{R}$, from an input vector $\theta \in \mathcal{M}$ to the function value $J(\theta)$. Hence, we assume that function values $J(\theta)$, associated with different values of θ, are random variables and that any finite number of these random variables have a joint Gaussian distribution dependent on the values of θ [18]. For GPs, we define a prior mean function and a covariance function, $k(\theta_i, \theta_j)$, which defines the covariance (or kernel) of any two function values, $J(\theta_i)$ and $J(\theta_j)$. In this work, the mean is assumed to be zero without loss of generality. The choice of kernel is problem-dependent and encodes general assumptions such as smoothness of the unknown function. In the experimental section, we employ the 5/2 Matèrn kernel where the hyperparameters are optimized by maximizing the marginal likelihood [18]. This kernel function implies that the underlying function J is differentiable and takes values within the $2\sigma_f$ confidence interval with high probability.

The GP framework can be used to predict the distribution of the performance function $J(\theta^*)$ at an arbitrary input θ^* based on the past observations, $\mathcal{D} = \{\theta_i, J(\theta_i)\}_{i=1}^n$. Conditioned on $\mathcal{D}$, the mean and variance of the prediction are

$$\mu(\theta^*) = \boldsymbol{k}\boldsymbol{K}^{-1}\boldsymbol{J}; \quad \sigma^2(\theta^*) = k(\theta^*, \theta^*) - \boldsymbol{k}\boldsymbol{K}^{-1}\boldsymbol{k}^T \,, \tag{2.5}$$

where $\boldsymbol{K}$ is the kernel matrix with $K_{ij} = k(\theta_i, \theta_j)$, $\boldsymbol{k} = [k(\theta_1, \theta^*), \ldots, k(\theta_n, \theta^*)]$ and $\boldsymbol{J} = [J(\theta_1), \ldots, J(\theta_n)]$. Thus, the GP provides both the expected value of the performance function at any arbitrary point θ^* as well as a notion of the uncertainty of this estimate.

Bayesian Optimization (BO)

Bayesian optimization aims to find the global minimum of an unknown function [6,19,20]. BO is particularly suitable for the scenarios where evaluating the unknown function is expensive, which fits our problem in Sec. 2.1. At each iteration, BO uses the past observations $\mathcal{D}$ to model the objective function, and uses this model to determine informative sample locations. A common model used in BO for the underlying objective, and the one that we consider, are Gaussian processes (see Sec. 2.2). Using the mean and variance predictions of the GP from (2.5), BO computes the next sample location by optimizing the so-called acquisition function, $\alpha(\cdot)$. Different acquisition functions are used in literature to trade off between exploration and exploitation during the optimization process [6]. For example, the next evaluation for expected improvement (EI) acquisition function [21] is given by $\theta^* = \arg\min_\theta \alpha(\theta)$ where

$$\alpha(\theta) = \sigma(\theta)[u\Phi(u) + \phi(u)]; \quad u = (\mu(\theta) - T)/\sigma(\theta). \tag{2.6}$$

$\Phi(\cdot)$ and $\phi(\cdot)$ in (2.6), respectively, are the standard normal cumulative distribution and probability density functions. The target value T is the minimum of all explored data. Intuitively, EI selects the next parameter point where the expected improvement over T is maximal. Repeatedly evaluating the system at points given by (2.6) thus improves the observed performance. Note that optimizing $\alpha(\theta)$ in (2.6) does not require physical interactions with the system, but only evaluation of the GP model. When a new set of optimal parameters θ^* is determined, they are finally evaluated on the real objective function J (i.e., the system).

Chapter 3

Solution

3.1 Dynamics Optimization via BO (aDOBO)

This section presents the technical details of aDOBO, a novel framework for optimizing dynamics model for maximizing the resultant controller performance. In this work, $\theta \in \mathbb{R}^{n_x(n_x+n_u)}$, i.e., each dimension in θ corresponds to an entry of the A_θ or B_θ matrices. This parameterization is chosen for simplicity, but other parameterizations can easily be used.

Given an initial state of the system z_0 and the current system dynamics model $(A_{\theta'}, B_{\theta'})$, we design an optimal control sequence $\pi_0(z_0, \theta')$ that minimizes the cost function $J_0(\mathbf{z}_0^N, \mathbf{u}_0^{N-1})$, i.e., we solve the optimal control problem in (2.3). The first control of this control sequence is applied on the *actual system* and the next state z_1 is measured. We then similarly compute $\pi_1(z_1, \theta')$ starting at z_1, apply the first control in the obtained control sequence, measure z_2, and so on until we get z_N. Once $\mathbf{z}_0^N$ and $\mathbf{u}_0^{N-1}$ are obtained, we compute the true performance of $\mathbf{u}_0^{N-1}$ on the actual system by analytically computing $J_0(\mathbf{z}_0^N, \mathbf{u}_0^{N-1})$ using (2.2). We denote this cost by $J(\theta')$ for simplicity. We next update the GP based on the collected data sample $\{\theta', J(\theta')\}$. Finally, we compute θ^* that minimizes the corresponding acquisition function $\alpha(\theta)$ and repeat the process for $(A_{\theta^*}, B_{\theta^*})$. Our approach is illustrated in Figure 1.1 and summarized in Algorithm 1. Intuitively, aDOBO directly learns the shape of the cost function $J(\theta)$ as a function of linearizations (A_θ, B_θ). Instead of learning the global shape of this function through random queries, it analyzes the performance of all the past evaluations and by optimizing the acquisition function, generates the next query that provides the maximum information about the minima of the cost function. This direct *minima-seeking* behavior based on the *actual observed performance* ensures that our approach is data-efficient. Thus, in the space of all linearizations, we efficiently and directly search for the linearization whose corresponding controller minimizes J_0 on the actual system.

Since the problem in (2.3) is an optimal control problem for the linear system $(A_{\theta'}, B_{\theta'})$, depending on the form of the cost function J, different optimal control schemes can be used. For example, if J is quadratic, the optimal controller is a linear feedback controller given by the solution of a Riccati equation. If J is a general convex function, the optimal

Algorithm 1: aDOBO algorithm

```
1  $\mathcal{D} \longleftarrow$ if available: $\{\theta, J(\theta)\}$
2  Prior $\longleftarrow$ if available: Prior of the GP hyperparameters
3  Initialize GP with $\mathcal{D}$
4  while optimize do
5      Find $\theta^* = \arg\min_\theta \alpha(\theta)$;   $\theta' \longleftarrow \theta^*$
6      for $i = 0 : N-1$ do
7          Given $z_i$ and $(A_{\theta'}, B_{\theta'})$, compute $\pi_i(z_i, \theta')$
8          Apply $\pi_i^1(z_i, \theta')$ on the real system and measure $z_{i+1}$
9          $\mathbf{z}_0^N \longleftarrow (\mathbf{z}_0^N, z_{i+1})$
10         $\mathbf{u}_0^{N-1} \longleftarrow (\mathbf{u}_0^{N-1}, \pi_i^1(z_i, \theta'))$
11     Evaluate $J(\theta') := J_0(\mathbf{z}_0^N, \mathbf{u}_0^{N-1})$ using (2.2)
12     Update GP and $\mathcal{D}$ with $\{\theta', J(\theta')\}$
```

control problem is solved through a general convex MPC solver, and the resultant controller could be non-linear. Thus, depending on the form of J, the controller designed by aDOBO can be linear or non-linear. This property causes aDOBO to perform well in the scenarios where a linear controller is not sufficient, as shown in Sec. 5.1. More generally, the proposed framework is modular and other control schemes can be used that are more suitable for the given cost function, which allows us to capture a richer controller space.

Note that the GP in our algorithm can be initialized with dynamics models whose controllers are known to perform well on the actual system. This generally leads to a faster convergence. For example, when a good linearization of the system is known, it can be used to initialize $\mathcal{D}$. When no information is known about the system a priori, the initial models are queried randomly.

Chapter 4

Simulations

4.1 Numerical Simulations

In this section, we present some simulation results on the performance of the proposed method for controller design.

Dubins Car System

For the first simulation, we consider a three dimensional non-linear Dubins car whose dynamics are given as

$$\dot{x} = v\cos\phi, \quad \dot{y} = v\sin\phi, \quad \dot{\phi} = \omega\,, \tag{4.1}$$

where $z := (x, y, \phi)$ is the state of system, $p = (x, y)$ is the position, ϕ is the heading, v is the speed, and ω is the turn rate. The input (control) to the system is $u := (v, \omega)$. For simulation purposes, we discretize the dynamics at a frequency of 10Hz. Our goal is to design a controller that steers the system to the equilibrium point $z^* = 0, u^* = 0$ starting from the state $z_0 := (1.5, 1, \pi/2)$. In particular, we want to minimize the cost function

$$J_0(\mathbf{z}_0^N, \mathbf{u}_0^{N-1}) = \sum_{k=0}^{N-1} \left(z_k^T Q z_k + u_k^T R u_k\right) + z_N^T Q_f z_N\,. \tag{4.2}$$

We choose $N = 30$. Q, Q_f and R are all chosen as identity matrices of appropriate sizes. We also assume that the dynamics are not known; hence, we cannot directly design a controller to steer the system to the desired equilibrium. Instead, we use aDOBO to find a linearization of dynamics in (4.1) that minimizes the cost function in (4.2), directly from the experimental data. In particular, we represent the system in (4.1) by a parameterized linear system $z_{k+1} = A_\theta z_k + B_\theta u_k$, design a controller for this system and apply it on the actual system. Based on the observed performance, BO suggests a new linearization and the process is repeated. Since the cost function is quadratic in this case, the optimal control problem for a particular θ is an LQR problem, and can be solved efficiently. For BO, we use the MATLAB

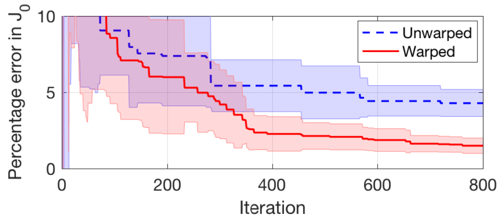


Figure 4.1: Dubins car: mean and standard deviation of η during the learning process. Using the log warping the learned controller reaches within 6% of the optimal cost in 200 iterations, outperforming the unwarped case.

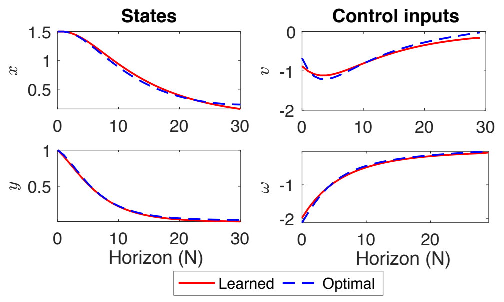


Figure 4.2: Dubins car: state and control trajectories for the learned and the true system. The two trajectories are very similar, indicating that the learned matrices represent system behavior accurately around the equilibrium point.

library *BayesOpt* [22]. Since there are 3 states and 2 inputs, we learn 15 parameters in total, one corresponding to each entry of the A_θ and B_θ matrices. The bounds on the parameters are chosen randomly as $\mathcal{M} = [-2, 2]^{15}$. As acquisition function, we use EI (see eq. (2.6)). Since no information is assumed to be known about the system, the GP was initialized with a random θ. We also warp the cost function J using the *log* function before passing it to BO. Warping makes the cost function smoother while maintaining its monotonic properties, which makes the sampling process in BO more efficient and leads to a faster convergence.

For comparison, we solve the true optimal controller that minimizes (4.2) subject to the dynamics in (4.1) using the non-linear solver *fmincon* in MATLAB to get the minimum achievable cost J_0^* across all controllers. We use the percentage error between the true optimal cost J_0^* and the cost achieved by aDOBO as our comparison metric in this work

$$\eta_n = 100 \times (J_0^* - J(\theta_n))/J_0^* \,, \tag{4.3}$$

where $J(\theta_n)$ is the best cost achieved by aDOBO by iteration n. In Fig. 4.1, we plot η_n for Dubins car. As learning progresses, aDOBO gathers more and more information about the minimum of J_0 and reaches within 6% of J_0^* in 200 iterations, demonstrating its effectiveness in designing a controller for an unknown system just from the experimental data. Fig. 4.1 also highlights the effect of warping in BO. A well warped function converges faster to the optimal performance. We also compared the control and state trajectories obtained from the learned controller with the optimal control and state trajectories. As shown in Fig. 4.2, the learned system matrices not only achieve the optimal cost, but also follow the optimal state and control trajectories very closely. Even though the trajectories are very close to each other for the true system and its learned linearization, this linearization may not correspond to any *actual* linearization of the system. The next simulation illustrates this property more clearly.

A Simple 1D Linear System

For this simulation, we consider a simple 1D linear system

$$z_{k+1} = z_k + u_k \,, \tag{4.4}$$

where z_k and u_k are the state and the input of the system at time k. Although the dynamics model is very simple, it illustrates some key insights about the proposed method. Our goal is to design a controller that minimizes (4.2) starting from the state $z_0 = 1$. We choose $N = 30$ and $R = Q = Q_f = 1$. Since the dynamics are unknown, we use aDOBO to learn the dynamics. Here $\theta := (\theta_1, \theta_2) \in \mathbb{R}^2$ are the parameters to be learned.

The learning process converges in 45 iterations to the true optimal performance ($J_0^* = 1.61$), which is computed using LQR on the real system. The converged parameters are $\theta_1 = 1.69$ and $\theta_2 = 2.45$, which are vastly different from the true parameters $\theta_1 = 1$ and $\theta_2 = 1$, even though the actual system is a linear system. To understand this, we plot the cost obtained on the true system J_0 as a function of linearization parameters (θ_1, θ_2) in Fig. 4.3. Since the performances of the two sets of parameters are very close to each other, a direct performance based learning process (e.g., aDOBO) cannot distinguish between them and both sets are equally optimal for it. More generally, a wide range of parameters lead to similar performance on the actual system. Hence, we expect the proposed approach to recover the optimal controller and the actual state trajectories, but not necessarily the true dynamics or its true linearization. This simulation also suggests that the true

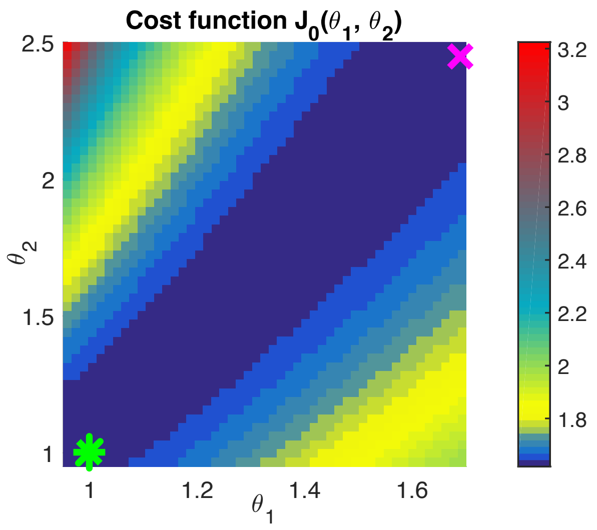


Figure 4.3: Cost of the actual system in (4.4) as a function of the linearization parameters (θ_1, θ_2). The parameters obtained by aDOBO (the pink X) yield to performance very close to the true system parameters (the green $*$). Note that aDOBO does not necessarily converge to the true parameters.

dynamics of the system may not even be required as far as the control performance is concerned.

Cart-pole System

We next apply aDOBO to a cart-pole system

$$\begin{aligned} (M+m)\ddot{x} - ml\ddot{\psi}\cos\psi + ml\dot{\psi}^2\sin\psi &= F\,, \\ l\ddot{\psi} - g\sin\psi &= \ddot{x}\cos\psi\,, \end{aligned} \tag{4.5}$$

where x denotes the position of the cart with mass M, ψ denotes the pendulum angle, and F is a force that serves as the control input. The massless pendulum is of length l with a mass m attached at its end. Define the system state as $z := (x, \dot{x}, \psi, \dot{\psi})$ and the input as $u := F$. Starting from the state $(0, 0, \frac{\pi}{6}, 0)$, the goal is to keep the pendulum straight up, while keeping the state within given lower and upper bounds. In particular, we want to minimize the cost

$$\begin{aligned} J_0(\mathbf{z}_0^N, \mathbf{u}_0^{N-1}) = &\sum_{k=0}^{N-1} \left(z_k^T Q z_k + u_k^T R u_k\right) + z_N^T Q_f z_N \\ &+ \lambda \sum_{i=0}^{N} \max(0, \underline{z} - z_i, z_i - \overline{z}), \end{aligned} \tag{4.6}$$

where λ penalizes the deviation of state z_i below $\underline{z}$ and above $\overline{z}$. We assume that the dynamics are unknown and use aDOBO to optimize the dynamics. For simulation, we discretize the dynamics at a frequency of 10Hz. We choose $N = 30$, $M = 1.5$Kg, $m = 0.175$Kg, $\lambda = 100$ and $l = 0.28$m. The $Q = Q_f = \text{diag}([0.1, 1, 100, 1])$ and $R = 0.1$ matrices are chosen to penalize the angular deviation significantly. We use $\underline{z} = [-2, -\infty, -0.1, -\infty]$ and $\overline{z} = [2, \infty, \infty, \infty]$, i.e., we are interested in controlling the pendulum while keeping the cart position within $[-2, 2]$, and limiting the pendulum overshoot to 0.1. The optimal control problem for a particular linearization is a convex MPC problem and solved using YALMIP [23]. The true J_0^* is computed using *fmincon*.

As shown in Fig. 4.4, aDOBO reaches within 20% of the optimal performance in 250 iterations and continue to make progress towards finding the optimal controller. This simulation demonstrates that the proposed method (a) is applicable to highly non-linear systems, (b) can handle general convex cost functions that are not necessarily quadratic, and (c) different optimal control schemes can be used within the proposed framework. Since an MPC controller can in general be non-linear, this implies that the proposed method can also design non-linear controllers.

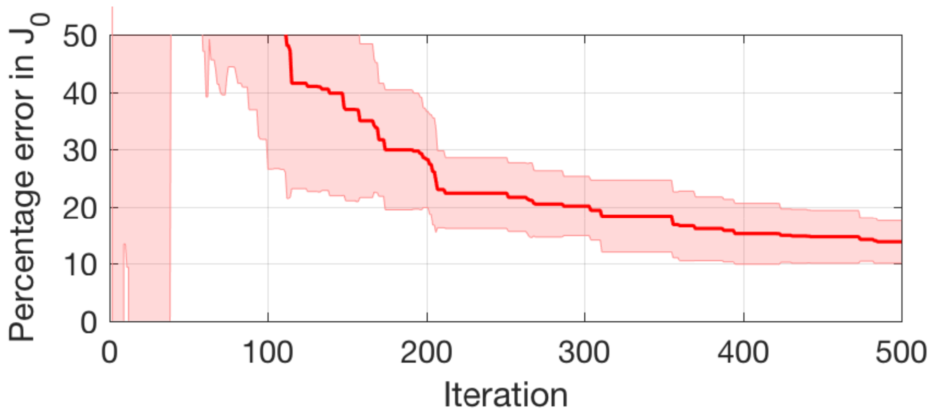


Figure 4.4: Cart-pole system: mean and standard deviation of η during the learning process. The learned controller reaches within 20% of the optimal cost in 250 iterations, demonstrating the applicability of aDOBO to highly non-linear systems.

Chapter 5

Comparison with Other Methods

In this section, we compare our approach with some other online learning schemes for controller design.

5.1 Tuning (Q, R) vs aDOBO

In this section, we consider the case in which the cost function J_0 is quadratic (see Eq. (4.2)). Suppose that the actual linearization of the system around $z^* = 0$ and $u^* = 0$ is known and given by (A^*, B^*). To design a controller for the actual system in such a case, it is a common practice to use an LQR controller for the linearized dynamics. However, the resultant controller may be sub-optimal for the actual non-linear system. To overcome this problem, authors in [1, 10] propose to optimize the controller by tuning penalty matrices Q and R in (4.2). In particular, we solve

$$\begin{aligned} \theta^* &= \arg\min_{\theta \in \mathcal{M}} J_0(\mathbf{z}_0^N, \mathbf{u}_0^{N-1}) \, , \\ \text{sub. to} \;\; z_{k+1} &= f(z_k, u_k), \quad u_k = K(\theta) z_k \, , \\ K(\theta) &= LQR(A^*, B^*, W_Q(\theta), W_R(\theta), Q_f) \, , \end{aligned} \tag{5.1}$$

where $K(\theta)$ denotes the LQR feedback matrix obtained for the system matrices (A^*, B^*) with W_Q and W_R as state and input penalty matrices, and can be computed analytically. For further details of LQR method, we refer interested readers to [24]. The difference between optimization problems (2.4) and (5.1) is that now we parameterize penalty matrices W_Q and W_R instead of system dynamics. The optimization problem in (5.1) is solved using BO in a similar fashion as we solve (2.4) [1]. The parameter θ, in this case, can be initialized by the actual penalty matrices Q and R, instead of a random query, which generally leads to a much faster convergence. An alternative approach is to use aDOBO, except that now we can use (A^*, B^*) as initializations for the system matrices A and B.

When (A^*, B^*) are known to a good accuracy, (Q, R) tuning method is expected to converge quickly to the optimal performance compared to aDOBO as it needs to learn fewer

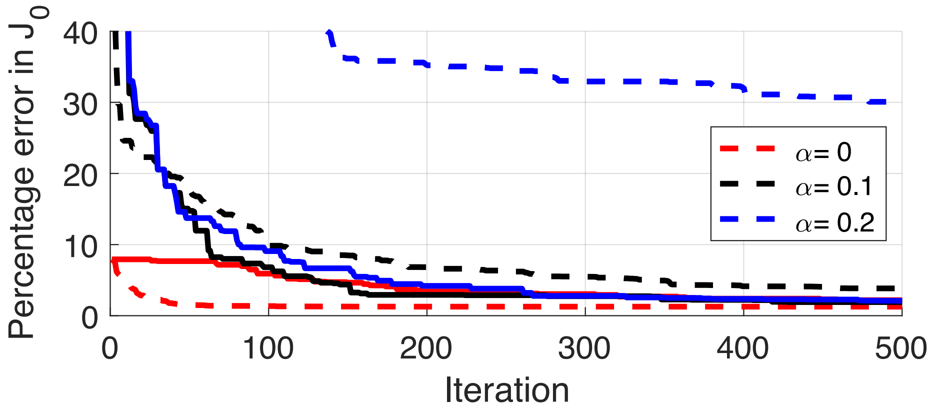


Figure 5.1: Dubins car: Comparison between (Q, R) tuning [1] (dashed curves), and aDOBO (solid curves) for different noise levels in (A^*, B^*). When the true linearized dynamics are known perfectly, the (Q, R) tuning method outperforms aDOBO because fewer parameters are to be learned. Its performance, however, drops significantly as noise increases, rendering the method impractical for the scenarios where system dynamics are not known to a good accuracy.

parameters, i.e., (n_x+n_u) (assuming diagonal penalty matrices) compared to $n_x(n_x+n_u)$ parameters for aDOBO. However, when there is error in (A^*, B^*) (or more generally if dynamics are unknown), the performance of the (Q, R) tuning method can degrade significantly as it relies on an accurate linearization of the system dynamics, rendering the method impractical for control design purposes. To compare the two methods we use the Dubins car model in Eq. (4.1). The rest of the simulation parameters are same as Section 4.1. We compute the linearization of Dubins car around $z^* = 0$ and $u^* = 0$ using (4.1) and add random matrices (A_r, B_r) to them to generate $A' = (1-\alpha)A^* + \alpha A_r$ and $B' = (1-\alpha)B^* + \alpha B_r$. We then initialize both methods with (A', B') for different αs. As shown in Fig. 5.1, the (Q, R) tuning method outperforms aDOBO, when there is no noise in (A^*, B^*). But as α increases, its performance deteriorates significantly. In contrast, aDOBO is fairly indifferent to the noise level, as it does not assume any prior knowledge of system dynamics. The only information assumed to be known is penalty matrices (Q, R), which are generally designed by the user and hence are known a priori. The another limitation of tuning (Q, R) is that, by design, it can only be used for a quadratic cost function J_0, whereas aDOBO can be used for more general cost functions as shown in Sec. 4.1.

Learning K vs aDOBO

When the cost function is quadratic, another potential approach is to directly parameterize and optimize the feedback matrix $K \in \mathbb{R}^{n_x n_u}$ in (5.1) [2] as

$$\begin{aligned} \theta^* &= \arg\min_{\theta \in \mathcal{M}} J_0(\mathbf{z}_0^N, \mathbf{u}_0^{N-1}), \\ \text{sub. to} \quad z_{k+1} &= f(z_k, u_k), \quad u_k = K_\theta z_k\,. \end{aligned} \tag{5.2}$$

The advantage of this approach is that only $n_x n_u$ parameters are learned compared to $n_x(n_x + n_u)$ parameters in aDOBO, which is also evident from Fig. 5.2a, wherein the learning process for K converges much faster than that for aDOBO. However, a linear controller might not be sufficient for general cost functions, and non-linear controllers are required to achieve a desired performance. As shown in Sec. 4.1, aDOBO is not limited to linear controllers; hence, it outperforms the K learning method in such scenarios. Consider, for example, the linear system

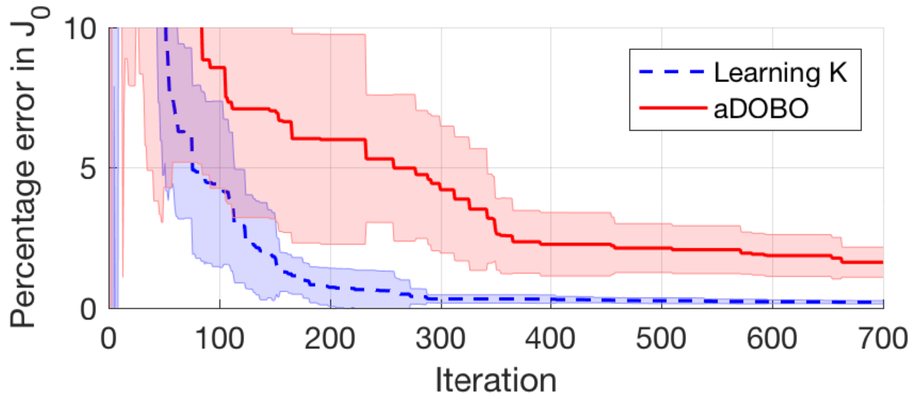
$$x_{k+1} = x_k + y_k, \quad y_{k+1} = y_k + u_k\,, \tag{5.3}$$

and the cost function in Eq. (4.6) with state $z_k = (x_k, y_k)$, $N = 30$, $\underline{z} = [0.5, -0.4]$ and $\overline{z} = [\infty, \infty]$. Q, Q_f and R are all identity matrices of appropriate sizes, and $\lambda = 100$.

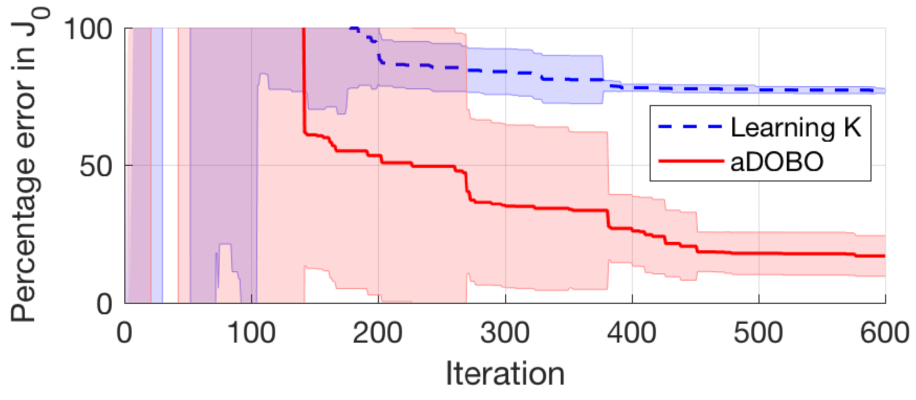
As evident from Fig. 5.2b, directly learning a feedback matrix performs poorly with an error as high as 80% from the optimal cost. Since the cost is not quadratic, the optimal controller is not necessarily linear; however, since the controller in (5.2) is restricted to a linear space, it performs rather poorly in this case. In contrast, aDOBO continues to improve performance and reaches within 20% of the optimal cost within few iterations, because we implicitly parameterize a much richer controller space via learning A and B. In this example, we capture non-linear controllers by using a linear dynamics model with a convex MPC solver. Since the underlying system is linear, the true optimal controller is also in our search space. Our algorithm makes sure that we make a steady progress towards finding that controller. However, we are not restricted to learning a linear controller K. One can also directly learn the actual control sequence to be applied to the system (which also captures the optimal controller). This approach may not be data-efficient compared to aDOBO as the control sequence space can be very large depending on the problem horizon, and will require a large number of experiments. As shown in Table 5.1, the performance error is more than 250% even after 600 iterations, rendering the method impractical for real systems.

5.2 Adaptive Control vs aDOBO

In this work, we aim to directly find the best linearization based on the observed performance. Another approach is to learn a true linearization of the system based on the observed state and input trajectory during the experiments. The underlying hypothesis is that as more and more data is collected, a better linearization is obtained, eventually leading to an improved control performance. This approach is in-line with the traditional model identification and



(a) Dubins car



(b) System of Eq. (5.3)

Figure 5.2: Mean and standard deviation of η obtained via directly learning K [2] and aDOBO for different cost functions. (a) Comparison for the quadratic cost function of Eq. (4.2). Directly learning K converges to the optimal performance faster because fewer parameters are to be learned. (b) Comparison for the non-quadratic cost function of Eq. (4.6). Since the optimal controller for the actual system is not necessarily linear in this case, directly learning K leads to a poor performance

the adaptive control frameworks. Let $({}_j\mathbf{z}_0^N, {}_j\mathbf{u}_0^{N-1})$ denotes the state and input trajectories for experiment j. We also let $\mathcal{D}_i = \cup_{j=1}^{i}({}_j\mathbf{z}_0^N, {}_j\mathbf{u}_0^{N-1})$. After experiment i, we fit an LTI model of the form $z_{k+1} = A_i z_k + B_i u_k$ using least squares on data in $\mathcal{D}_i$ and then use this model to obtain a new controller for experiment $i+1$. We apply the approach on the linear system in (5.3) and the non-linear system in (4.1) with the cost function in (4.2). For the linear system, the approach converges to the true system dynamics in 5 iterations. However, this approach performs rather poorly on the non-linear system, as shown in Table 5.2. When the underlying system is non-linear, all state and input trajectories may not contribute to the performance improvement. A good linearization should be obtained from the state and input trajectories in the region of interest, which depends on the task. For example, if we

Iteration	aDOBO	Learning Control Sequence
200	**53 ± 50%**	605 ± 420%
400	**27 ± 12%**	357 ± 159%
600	**17 ± 7%**	263 ± 150%

Table 5.1: System in (5.3): mean and standard deviation of η for aDOBO, and for directly learning the control sequence. Since the space of control sequence is huge, the error is substantial even after 600 iterations.

Iteration	aDOBO	Learning via LS
200	**6 ± 3.7%**	166.7 ± 411%
400	**2.2 ± 1.1%**	75.9 ± 189%
600	**1.8 ± 0.7%**	70.7 ± 166%

Table 5.2: Dubins car: mean and standard deviation of η obtained via learning (A, B) through least squares (LS), and through aDOBO.

want to regulate the system to the equilibrium $(0, 0)$, a linearization of the system around $(0, 0)$ should be obtained. Thus, it is desirable to use the system trajectories that are close to this equilibrium point. However, a naive prediction error based approach has no means to select these "good" trajectories from the pool of trajectories and hence can lead to a poor performance. In contrast, aDOBO does not suffer from these limitations, as it explicitly utilizes a performance based optimization. A summary of the advantages and limitations of the four methods is provided in Table 5.3.

Method	**Advantages**	**Limitations**
(Q, R) learning [1]	Only $(n_x + n_u)$ parameters are to be learned so learning will be faster.	Performance can degrade significantly if the dynamics are not known to a good accuracy; only applicable when the cost function is quadratic.
F learning [2]	Only $n_x n_u$ parameters are to be learned so learning will be faster.	Approach may not perform well for non-quadratic cost functions.
(A, B) learning via least squares	Can lead to a faster convergence when the underlying system is linear	Approach is not suitable for non-linear system.
aDOBO	Does not require any prior knowledge of system dynamics. Applicable to general cost functions.	Number of parameters to be learned is higher, i.e., $(n_x^2 + n_x n_u)$.

Table 5.3: Relative advantages and limitations of different methods for automatic controller design.

Chapter 6

Quadrotor Position Tracking Experiments

We now present the results of our experiments on Crazyflie 2.0, which is an open source nano quadrotor platform developed by Bitcraze. Its small size, low cost, and robustness make it an ideal platform for testing new control paradigms. Recently, it has been extensively used to demonstrate aggressive flights [25,26]. For small yaw, the quadrotor system is modeled as a rigid body with a ten dimensional state vector $s := \left[p, v, \zeta, \omega\right]$, which includes the position $p = (x, y, z)$ in an inertial frame I, linear velocities $v = (v_x, v_y, v_z)$ in I, attitude (orientation) represented by Euler angles ζ, and angular velocities ω. The system is controlled via three inputs $u := \left[u_1, u_2, u_3\right]$, where u_1 is the thrust along the z-axis, and u_2 and u_3 are rolling, pitching moments respectively. The full non-linear dynamics of a quadrotor are derived in [27], and its physical parameters are computed in [25].

$$\dot{s} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{\phi} \\ \dot{\psi} \\ \dot{\omega_x} \\ \dot{\omega_y} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ -\cos\phi \sin\psi \frac{u_1}{m} \\ \sin\phi \frac{u_1}{m} \\ g - \cos\phi \cos\psi \frac{u_1}{m} \\ \omega_x + \sin\phi \tan\psi \omega_y \\ \cos\phi \omega_y \\ \frac{L}{I_x} u_2 \\ \frac{L}{I_y} u_3 \end{bmatrix} \tag{6.1}$$

where L, m, I_x, I_y are physical parameters of the quadrotor and are obtained from [25]. Our goal in this experiment is to track a desired position p^* starting from the initial position $p_0 = [0, 0, 1]$. Formally, we minimize

$$J_0(\bar{\mathbf{s}}_0^N, \mathbf{u}_0^{N-1}) = \sum_{k=0}^{N-1} \left(\bar{s}_k^T Q \bar{s} + u_k^T R u_k\right) + \bar{s}_N^T Q_f \bar{s}\,, \tag{6.2}$$

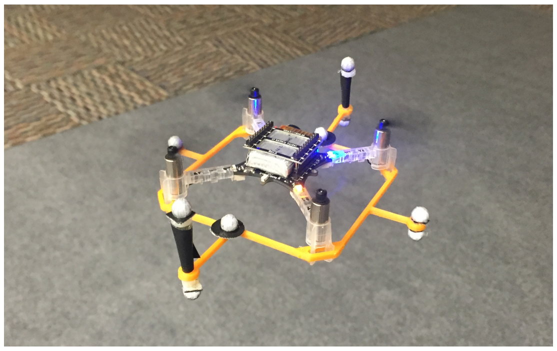


Figure 6.1: The Crazyflie 2.0

where $\bar{s} := \left[p - p^*, v, \zeta, \omega\right]$. Given the dynamics in [27], the desired optimal control problem can be solved using LQR; however, the resultant controller may still not be optimal for the actual system because (a) true underlying system is non-linear and (b) the actual system may not follow the dynamics in [27] exactly due to several unmodeled effects, as illustrated in our results. Hence, we assume that the dynamics of v_x and v_y are unknown, and model them as

$$\begin{bmatrix} f_{v_x} \\ f_{v_y} \end{bmatrix} = A_\theta \begin{bmatrix} \phi \\ \psi \end{bmatrix} + B_\theta u_1 \,, \tag{6.3}$$

where A and B are parameterized through θ. Our goal is to learn the parameter θ^* that minimizes the cost in (6.2) for the *actual* Crazyflie using aDOBO. We use $N = 400$; the penalty matrix Q is chosen to penalize the position deviation. In our experiments, Crazyflie was flown in presence of a VICON motion capture system, which along with on-board sensors provides the full state information at 100Hz. The optimal control problem for a particular linearization in (6.3) is solved using LQR. For comparison, we compute the *nominal* optimal controller using the full dynamics in [27]. Figure 6.2 shows the performance of the controller from aDOBO compared with the nominal controller during the learning process. The nominal controller outperforms the learned controller initially, but within a few iterations, aDOBO performs better than the controller derived from the known dynamics model of Crazyflie. This is because aDOBO optimizes controller based on the performance of the *actual* system and hence can account for unmodeled effects. In 45 iterations, the learned controller outperforms the nominal controller by 12%, demonstrating the performance potential of aDOBO on real systems.

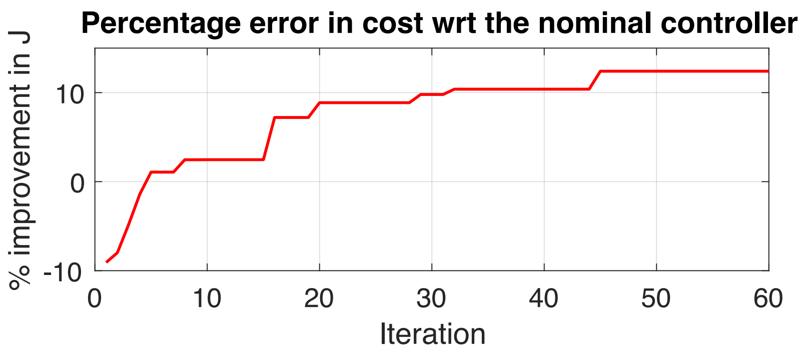


Figure 6.2: Crazyflie: Percentage error between the learned and nominal controllers. As learning progresses, aDOBO outperforms the nominal controller by 12% on the actual system.

Chapter 7

Conclusion and Future Work

In this work, we introduce aDOBO, an active learning framework to optimize the system dynamics with the intent of maximizing the controller performance. Through simulations and real-world experiments, we demonstrate that aDOBO achieves optimal control performance even when no prior information is known about the system dynamics. In addition, we compare aDOBO with similar Bayesian Optimization based methods in the space of learning systems dynamics with data efficiency constraints, and show improvements against existing benchmarks. For future work, it will be interesting to generalize aDOBO to optimize the dynamics for a class of cost functions. Leveraging the state and input trajectory data, along with the observed performance, to further increase the data-efficiency of the learning process is another promising direction. In addition, an accurate prediction model can be used to design a controller for a variety of cost functions, whereas aDOBO learns a model that is specific to a single cost function. It will be interesting to generalize aDOBO to optimize the dynamics for a class of cost functions. Finally, it will be interesting to see how aDOBO can scale to more complex non-linear dynamics models.

Bibliography

[1] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe, "Automatic LQR tuning based on Gaussian process global optimization," in *International Conference on Robotics and Automation*, 2016.

[2] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian optimization for learning gaits under uncertainty," *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1, pp. 5–23, 2015.

[3] K. Fragkiadak, S. Levine, P. Felsen, and J. Malik, "Recurrent network models for human dynamics." in *International Conference on Computer Vision*, 2015.

[4] C. Chamberlain, "System identification, state estimation, and control of unmanned aerial robots," Master's thesis, BYU, 2011.

[5] S. Bansal, A. Akametalu, F. Jiang, F. Laine, and C. Tomlin, "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors," *arXiv preprint arXiv:1610.05863*, 2016.

[6] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[7] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2015.

[8] J. Joseph, A. Geramifard, J. W. Roberts, J. P. How, and N. Roy, "Reinforcement learning with misspecified model classes," in *International Conference on Robotics and Automation*, 2013, pp. 939–946.

[9] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, 2011.

[10] S. Trimpe, A. Millane, S. Doessegger, and R. D'Andrea, "A self-tuning LQR approach demonstrated on an inverted pendulum," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11 281–11 287, 2014.

[11] J. W. Roberts, I. R. Manchester, and R. Tedrake, "Feedback controller parameterizations for reinforcement learning," in *Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, 2011, pp. 310–317.

[12] K. J. Åström and B. Wittenmark, *Adaptive control.* Courier Corporation, 2013.

[13] M. Grimble, "Implicit and explicit LQG self-tuning controllers," *Automatica*, vol. 20, no. 5, pp. 661–669, 1984.

[14] D. Clarke, P. Kanjilal, and C. Mohtadi, "A generalized LQG approach to self-tuning control part i. aspects of design," *International Journal of Control*, vol. 41, no. 6, pp. 1509–1523, 1985.

[15] R. Murray-Smith and D. Sbarbaro, "Nonlinear adaptive control using nonparametric Gaussian process prior models," *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 325–330, 2002.

[16] S. Sastry and M. Bodson, *Adaptive control: stability, convergence and robustness.* Courier Corporation, 2011.

[17] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomlin, "Goal-driven dynamics learning via bayesian optimization," *CoRR*, vol. abs/1703.09260, 2017. [Online]. Available: http://arxiv.org/abs/1703.09260

[18] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning.* The MIT Press, 2006.

[19] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," *Journal of Basic Engineering*, vol. 86, p. 97, 1964.

[20] M. A. Osborne, R. Garnett, and S. J. Roberts, "Gaussian processes for global optimization," in *Learning and Intelligent Optimization (LION3)*, 2009, pp. 1–15.

[21] J. Močkus, "On bayesian methods for seeking the extremum," in *Optimization Techniques IFIP Technical Conference*, 1975.

[22] R. Martinez-Cantin, "BayesOpt: a bayesian optimization library for nonlinear optimization, experimental design and bandits." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3735–3739, 2014.

[23] J. Lofberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *International Symposium on Computer Aided Control Systems Design*, 2005, pp. 284–289.

[24] D. J. Bender and A. J. Laub, "The linear-quadratic optimal regulator for descriptor systems: discrete-time case," *Automatica*, 1987.

[25] B. Landry, “Planning and control for quadrotor flight through cluttered environments,” Master’s thesis, MIT, 2015.

[26] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, “Learning quadrotor dynamics using neural network for flight control,” in *Conference on Decision and Control*, 2016, pp. 4653–4660.

[27] N. Abas, A. Legowo, and R. Akmeliawati, “Parameter identification of an autonomous quadrotor,” in *International Conference On Mechatronics*, 2011, pp. 1–8.