Optimal Controller Synthesis for Nonlinear Systems

Thesis by
Yoke Peng Leong

In Partial Fulfillment of the Requirements for the
degree of
Control and Dynamical Systems

Caltech

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2018
Defended November 27, 2017

ACKNOWLEDGEMENTS

Graduate school is not always smooth sailing. The beginning and the completion of my thesis would not have happened without these important people in my life.

First and foremost, I would like to express my gratitude to my adviser, Prof. Joel Burdick, for his generosity and patience. His guidance is what made this thesis possible. I also appreciate the time he took to provide advices on my professional career beyond research. I would also like to thank my co-adviser, Prof. John Doyle, for his stimulating research discussions and the wealth of interesting research ideas. He opens up my originally narrow view of control to applications beyond traditional engineering. Both of them have provided me the opportunity and freedom to explore, learn, and do many different things in research and beyond during my time at Caltech.

I would also like to thank my thesis committee, Prof. Richard Murray and Prof. Aaron Ames, for their time and advice on improving my thesis. In addition, I want to thank Prof. Richard Murray for being available when I need advices on navigating the academic life and making decisions related to my future career path.

Apart from that, I would like to thank Matanya Horowitz for his guidance in the beginning of my graduate career when I am lost in the ocean of research. This thesis would have been totally different without him. The summer of 2014 at Spain marked the beginning of a productive collaboration with Prof. Pavithra Prabhakar, who introduced me to a fascinating area of research that becomes a part of this thesis. I would like to thank her for hosting me back then, and for her continuous guidance since then.

Life at Caltech would have been so much more mundane without my colleagues and friends in the CMS department, in particular, my classmates, Yorie Nakahira, Ania Baetica, Ioannis Filippidis, and Yuh-Shyang Wang. Stimulating discussions we had in Annenberg have inspired me in many ways in my research. In addition, I would have so much difficulties navigating the world of partial differential equations without my colleague and friend, Albert Chern, who spent hours selflessly answering all my questions related to applied mathematics. Of course, besides research, I appreciate all the great memories we have created and shared on campus and beyond.

My career in Caltech would not have existed without my undergraduate adviser, Prof. Todd Murphey. He introduced me to the world of research when I was young and clueless, and convinced me to choose Caltech when I was torn between staying or leaving my alma mater, Northwestern University, for graduate school. His words of advice and his energy and enthusiasm are what keeps me going when times are difficult at Caltech.

Last but not least, I would like to thank my family for their love and support. Most importantly, I would like to thank my husband, Tony Bartolotta, for keeping me sane and healthy during my time at Caltech. With you, everything is possible.

ABSTRACT

Optimal controller synthesis is a challenging problem to solve. However, in many applications such as robotics, nonlinearity is unavoidable. Apart from optimality, correctness of the system behaviors with respect to system specifications such as stability and obstacle avoidance is vital for engineering applications. Many existing techniques consider either the optimality or the correctness of system behavior. Rarely, a tool exists that considers both. Furthermore, most existing optimal controller synthesis techniques are not scalable because they either require ad-hoc design or they suffer from the curse of dimensionality.

This thesis aims to close these gaps by proposing optimal controller synthesis techniques for two classes of nonlinear systems: linearly solvable nonlinear systems and hybrid nonlinear systems. Linearly solvable systems have associated Hamilton-Jacobi-Bellman (HJB) equations that can be transformed from the original nonlinear partial differential equation (PDE) into a linear PDE through a logarithmic transformation. The first part of this thesis presets two methods to synthesize optimal controller for linearly solvable nonlinear systems. The first technique uses a hierarchy of sums-of-square programs to compute a sequence of suboptimal controllers that have non-increasing suboptimality for first exit and finite horizon problems. This technique is the first systematic approach to provide stability and suboptimal performance guarantees for stochastic nonlinear systems in one framework. The second technique uses the low rank tensor decomposition framework to solve the linear HJB equation for first exit, finite horizon, and infinite horizon problems. This technique scale linearly with dimensions, alleviating the curse of dimensionality and enabling us to solve the linear HJB equation for a quadcopter model that is a twelve-dimensional system on a personal laptop. A new algorithm is proposed for a key step in the controller synthesis algorithm to solve the ill-conditioning issue that arises in the original algorithm. A MATLAB toolbox that implements the algorithms is developed, and the performance of these algorithms is illustrated by a few engineering examples.

Apart from stability, in many applications, more complex specifications such as obstacle avoidance, reachability, and surveillance are required. The second part of the thesis describes methods to synthesize optimal controllers for hybrid nonlinear systems with quantitative objectives (*i.e.,* minimizing cost) and qualitative objectives (*i.e.,* satisfying specifications). This thesis focuses on two types of qualitative objectives, regular objectives, and ω-regular objectives. Regular objectives capture bounded time behavior such as reachability, and ω-regular objectives capture long term behavior such as surveillance. For both types of objectives, an abstraction-refinement procedure that preserves the cost is developed. A two-player game is solved on the product of the abstract system and the given objectives to synthesize the suboptimal controller for the hybrid nonlinear system. By refining the abstract system, the algorithms are guaranteed to converge to the optimal cost and return the optimal controller if the original systems are robust with respect to the initial states and the optimal controller inputs. The proposed technique is the first abstraction-refinement based technique to combine both quantitative and qualitative objectives into one framework. A Python implementation of the algorithms are developed, and a few engineering examples are presented to illustrate the performance of these algorithms.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] Y. P. Leong and P. Prabhakar, "Abstraction-refinement based optimal control with regular objectives," arXiv:1504.02838, 2017,
Y.P.L. participated in the conception of the project, derived and analyzed the solution technique, developed the numerical tool, and participated in the writing of the manuscript.

[2] Y. P. Leong and P. Prabhakar, "Optimal control with regular objectives using an abstraction-refinement approach," in *American Controls Conf. (ACC)*, 2016, pp. 5161–5168. DOI: `10.1109/ACC.2016.7526478`,
Y.P.L. participated in the conception of the project, derived and analyzed the solution technique, developed the numerical tool, and participated in the writing of the manuscript.

[3] Y. P. Leong, M. B. Horowitz, and J. W. Burdick, "Linearly solvable stochastic control lyapunov functions," *SIAM Journal on Control and Optimization*, vol. 54, no. 6, pp. 3106–3125, 2016. DOI: `10.1137/16M105767X`,
Y.P.L. participated in the conception of the project, derived and analyzed the solution technique, and participated in the writing of the manuscript.

[4] E. Stefansson and Y. P. Leong, "Sequential alternating least squares for solving high dimensional linear Hamilton-Jacobi-Bellman equation," in *IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016, pp. 3757–3764. DOI: `10.1109/IROS.2016.7759553`,
Y.P.L. participated in the conception of the project, and participated in the writing of the numerical tool and the manuscript.

[5] Y. P. Leong, M. B. Horowitz, and J. W. Burdick, "Suboptimal stabilizing controllers for linearly solvable system," in *IEEE Int. Conf. on Decision and Control (CDC)*, 2015, pp. 7157–7164. DOI: `10.1109/CDC.2015.7403348`,
Y.P.L. participated in the conception of the project, derived and analyzed the solution technique, and participated in the writing of the manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABBREVIATIONS

ALS. Alternating least squares.

CLF/SCLF. Control Lyapunov function/stochastic control Lyapunov function.

CP. CANDECOMP/PARAFAC.

HJB. Hamilton-Jacobi-Bellman.

LQG. Linear quadratic Gaussian regular.

LQR. Linear quadratic regular.

LSOC. Linear stochastic optimal control.

PDE. Partial differential equation.

SeALS. Sequential alternating least squares.

SOS. Sum-of-Squares.

VTOL. Vertical takeoff and landing.

LIST OF NOTATIONS

Notations commonly used throughout the thesis are listed here. Notations specific to a particular chapter are defined individually in the corresponding chapter.

Sets

$\mathbb{Z}$	All integers.
$\mathbb{Z}_+$	All positive integers (excluding zero).
$\mathbb{N}$	All natural numbers (including zero).
$\mathbb{R}$	All real numbers.
$\mathbb{R}_+$	All nonnegative real numbers.
$\mathbb{R}^n$	All n-dimensional real vectors.
$\mathbb{R}^{n\times m}$	All $n \times m$ real matrices.

Function Classes

Given a function class $\mathcal{F}$, with abuse of notation, let $\mathcal{F}(x)$ denotes all real functions of $x \in \mathbb{R}^n$ and $\mathcal{F}(\Omega)$ denotes all real functions of $x \in \Omega \subseteq \mathbb{R}^n$.

$\mathcal{R}$	All real polynomial functions.
$\mathcal{R}^{n\times m}$	All matrices of size $n \times m$ such that $M_{i,j} \in \mathcal{R}\ \forall\ i, j$.
$\mathcal{K}$	All real continuous nondecreasing functions $\mu : \mathbb{R}_+ \to \mathbb{R}_+$ such that $\mu(0) = 0$, $\mu(r) > 0$ if $r > 0$, and $\mu(r) \geq \mu(r')$ if $r > r'$.
C^k	All real k-differentiable functions f.
$C^{k,k'}$	All real functions f such that f is k-differentiable with respect to the first argument and k'-differentiable with respect to the second argument.
$\mathcal{L}_p$	All real p-th integrable functions f, where $\left(\int \lvert f\rvert^p dx\right)^{\frac{1}{p}} < \infty$.
$\mathcal{SOS}$	All real sum-of-squares polynomials f.
$\mathcal{S}$	All real separable functions $f : \mathbb{R}^n \to \mathbb{R}$ such that $f(x) = \sum_{i=1}^{r} s_i \prod_{j=1}^{d} f_j^i(x_j)$, where $f_j^i \in C^\infty(x_j)$ and x_j is the j-th component of x.
$\mathcal{S}^{n\times m}$	All matrices of size $n \times m$ such that $M_{i,j} \in \mathcal{S}\ \forall\ i, j$.

Sequences

$[k]$	An integer sequence from $0, 1, \ldots, k$.
$[k]_+$	An integer sequence from $1, 2, \ldots, k$.

$\{x_i\}_{i=m}^{n}$	A sequence $x_m, x_{m+1}, \ldots, x_{n-1}, x_n$, where $n \geq m$.
$\{x_i\}_{i\in[n]}$	A sequence $x_0, x_1, \ldots, x_{n-1}, x_n$.

Norms

Let $x \in \mathbb{R}^n$ be a vector, $M \in \mathbb{R}^{n\times n}$ be a matrix, $f \in C^\infty(\Omega)$ be a function on Ω, and F be a CP tensor function.

$\|x\|_\infty$	$\max_{k\in[n]_+} \lvert x_k\rvert$.
$\|x\|_1$	$\sum_{k\in[n]} \lvert x_k\rvert$.
$\|f\|_\infty$	$\sup_{x\in\Omega} \lvert f(x)\rvert$ if exists.
$\|M\|_F$	Frobenius norm $\sqrt{\sum_{i=1}^n \sum_{j=1}^n \lvert M_{ij}\rvert^2}$.
$\|M\|_\infty$	$\max_{i\in[n]_+} \sum_{j=1}^n \lvert M_{ij}\rvert$.
$\|F\|$	$\sqrt{\langle F, F\rangle}$.

Others

Let $x \in \mathbb{R}^n$ be a vector, $M \in \mathbb{R}^{n\times n}$ be a matrix, $f \in C^\infty(\Omega)$ be a function on Ω, $\mathcal{A}$ be a set, and F be a CP tensor function.

$\lvert\mathcal{A}\rvert$	The total number of elements in the set $\mathcal{A}$.
$f(\cdot)$	Abbreviation of a multivariate function $f(x)$. A function $f(x)$ may also be abbreviated as f if the context is clear.
Tr	Matrix trace.
∇_x	Gradient with respect to x.
∇_{xx}	Hessian with respect to x.
∂_k	Partial derivative with respect to k.
$\mathbb{1}_n$	A vector of all ones in $\mathbb{R}^n$. The size n is abbreviated when the context is clear.
I_n	Identity matrix in $\mathbb{R}^{n\times n}$. The size n is abbreviated when the context is clear.

Chapter 1

INTRODUCTION

Feedback control is a very old concept that has profound impact on the development of today's technology [1]. An interesting history of feedback control can be found in [2] that traces the control of devices to the ancient past. Applications of feedback control ranged from traditional engineering fields like chemical processes [3], aircrafts [4], spacecrafts [5], and robotics [6] to new areas like algorithm analysis [7], biology [8], [9], and neuroscience [10]–[12]. Typically, the goal of a controller is to ensure that the systems controlled is able to perform a task efficiently and reliably. This requirement translates to requiring the controller to be both optimal with respect to a metric and correct with respect to a given set of specifications, and perhaps robust against random disturbances.

Optimal controller synthesis for linear systems is a well studied area with many useful tools and techniques. An important milestones for optimal controller synthesis is the linear quadratic regular (LQR) introduced by Kalman, who showed that the optimal controller is a linear feedback of the state variables using the calculus of variation [13]. Despite initial successes in the 1970's, the LQR controller and its sister, linear quadratic Gaussian regular (LQG), have poor robustness properties [14]. Thus, in the 1980's, tools and techniques for robust control emerged including optimal H_2/H_∞ controller synthesis that uses state-space formulation and the Ricatti equation [15], [16]. More recently, the system level synthesis framework was developed in [17], [18] that enables distributed, localized, and scalable synthesis using convex optimization.

Linear systems are useful for modeling systems that only operate in the linear regime and designing optimal stabilizing controllers for these systems. But nonlinearity is unavoidable in many engineering applications such as controlling a walking robot [6]. Furthermore, using the full nonlinear model for controller synthesis typically results in a more efficient controller than only using the linearized model for controller synthesis. Thus, optimal controller synthesis for nonlinear systems is an important area of study. Yet, despite major progresses in optimal controller synthesis for linear systems, optimal controller synthesis for nonlinear systems does not share the same level of generality, computability, and scalability. Due to the nonlinearity, many efficient techniques that are developed for linear systems are no longer applicable for the nonlinear systems.

A fundamental concept in optimal control synthesis for nonlinear system is the Hamilton-Jacobi-Bellman (HJB) equation, a novel view by Bellman and his colleagues that extends the earlier works on optimal control based on calculus of variation. Bellman introduced the idea of dynamics programming for computing the optimal controller [19] based on the following simple idea: if an optimal path is found between two points, then for any other two points on the optimal path, the original optimal path between these two points is also the optimal path between these two points. Almost in parallel, the Pontraygin's maximum principle [20], [21] was developed by Pontraygin and his colleagues for solving the optimal control problem. The maximum principle provides the necessary conditions for optimal control using the concept of a Hamiltonian. This formulation transforms the optimal control problem into a nonlinear programming problem. However, unlike the dynamic programming approach, nonlinear programming that solves the Pontraygin's maximum principle does not provide a feedback controller; instead it provides an open loop optimal control trajectory for the system.

These two main schools of thought have major influences on the development of nonlinear optimal control synthesis since the 1950's. This thesis adopts Bellman's view of optimal control because we seek to compute optimal feedback controllers for nonlinear systems such as robots and quadcopters. The major challenge with this view of optimal control is the curse of dimensionality [22]. As computers become more efficient with more memory, the curse of dimensionally is not as daunting as before. But solving nonlinear optimal control synthesis is still a challenging task for systems that have more than 2 or 3 dimensions. Yet many engineering systems have more dimensions than three. For example, a simple quadcopter model has 12 state dimensions [23], and a biped robot can easily have more than 30 state dimensions [6].

Apart from scalability, unlike the tools for linear systems, satisfying both optimality and correctness of system behaviour with respect to system specifications is extremely challenging for nonlinear systems. Most existing methods usually satisfy one or the other. Rarely do we find a tool that satisfies both. Methods derived from the HJB equations and the maximum principle typically only consider optimality without explicitly considering correctness of system behaviors such as stability and obstacle avoidance. On the other hand, many methods [24], [25] are available that satisfy system specifications without considering optimality. Techniques for ensuring stability, the most common form of specifications for control, includes feedback linearization and differential geometry based methods [24], [25]. In essence, many of these synthesis techniques rely on finding control Lyapunov functions for the dynamical systems of interest. As a result, the techniques are generally not scalable, and in many cases ad-hoc to the specific applications. More recently, the development of convex optimization, in particular sums-of-squares programming, helps automate the process of searching for control Lyapunov functions [26], [27]. Apart from stability, robotic systems can have many other specifications such as surveillance, obstacle avoidance, and reachability. For these more complex specifications, the formal method emerges as the principled automatic synthesis technique that also formally verifies the correctness of the controlled system behaviors [28]–[31]. These formal techniques generally search for any feedback controller that is correct with respect to the specifications. Most of synthesis techniques mentioned in this paragraph focus on achieving correct system behaviors, but lack any form of optimality guarantees.

This thesis aims to close some of these gaps by proposing optimal controller synthesis techniques for nonlinear systems that are scalable and correct with respect to specifications. This thesis focuses on two general classes of nonlinear systems: linearly solvable nonlinear systems and hybrid nonlinear systems.

Part I Optimal Control Synthesis for Linearly Solvable Nonlinear Systems

As mentioned before, in nonlinear control theory, synthesizing any stabilizing controller is a huge challenge on its own. However, in many practical applications, where resources are limited, optimality is also important. Optimal controller synthesis is challenging because it involves solving the Hamilton-Jacobi-Bellman (HJB) equation that is typically a nonlinear partial differential equation. This part of the thesis aims to solve the HJB equation for the class of nonlinear affine stochastic systems that can be transformed into a linear PDE. This type of system is called a *linearly solvable system* throughout this thesis.

Two approaches are proposed to solve the linear HJB equation: Sums-of-Squares based technique and low rank tensor decomposition based technique, and three typical types of cost functionals are considered that give three types of control problems: the first exit problem, finite horizon problem, and infinite horizon problem. The first method synthesizes a suboptimal controller using SOS programming, a convex optimization based method, to solve the linear HJB equation. The resulting controller is guaranteed to be stabilizing and the trajectory cost of the controlled system is bounded. This method is among the first to explicitly consider both optimality and stability for stochastic nonlinear systems. The second method synthesizes the controller by solving the linear HJB equation using a low rank tensor decomposition based approach. This approach scales linearly with the dimensions, avoiding the curse of dimensionality suffered by the first method. The implementation of this technique is available online at [32].

The first part of the thesis includes the following chapters:

Chapter 2 provides a general overview of Part I and describes the main contributions.

Chapter 3 introduces background materials necessary for understanding the rest of the chapters in this part, including stochastic control Lyapunov function, linear HJB equation, viscosity solutions, CANDECOMP/PARAFAC tensor, and spectral discretization scheme.

Chapter 4 presents optimal controller synthesis technique that uses Sum-of-Squares program. This technique is the first to combine the optimality condition with the stability criteria in one framework using convex optimization. This technique not only synthesizes a suboptimal controller, but it provides guarantees on trajectory cost and system stability. However, one major limitation of this technique is the curse of dimensionality, which is addressed by the next chapter.

Chapter 5 synthesizes controller for a high dimensional system using low rank tensor decomposition. This technique scales linearly with dimensions and thus avoid the curse of dimensionality. The existing Alternating Least Square algorithm is improved with sequential computation. A MATLAB toolbox that implements algorithms presented in this chapter is developed.

Appendix A presents implementation of algorithms described in Chapter 5 for multiple engineering examples. The algorithm is able to solve for a quadcopter controller that has 12 degrees of freedom on a laptop.

Part II Optimal Control Synthesis for Hybrid Systems with Qualitative and Quantitative Objectives

Apart from qualitative criteria (*i.e.,* optimality with respect to a cost function), in many applications, there is also high level quantitative specifications, for instance, a mobile robot has to perform surveillance while avoiding obstacles. The second part of the thesis describes methods to synthesize optimal controllers for hybrid nonlinear systems that have both quantitative and qualitative specifications.

Two classes of quantitative objectives are considered: regular and ω-regular. The former captures bounded time behavior of the systems, including reachability, and the latter captures long term behavior such as surveillance. The first method considers systems with regular objectives. An abstraction-refinement approach that preserves the cost is developed for synthesizing an optimal controller that is correct with respect to the regular objectives. The second method considers systems with ω-regular objectives. A similar cost preserving abstraction-refinement approach in conjunction with solving a two-player quantitative game (*i.e.,* mean payoff parity game) is used to synthesize the controller. Both methods use an iterative abstraction-refinement approach that converges to the optimal controller if the systems are robust with respect to the initial states and the optimal inputs. The implementation of both techniques is available online at [33].

The second part of the thesis includes the following chapters:

Chapter 6 provides a general overview of the state-of-art of formal controller synthesis, and describes the main contributions of Part II in the context of the previous works.

Chapter 7 introduces the mathematical notations, presents the semantic model for discrete time hybrid systems with cost (*i.e.,* weighted transition systems), and formalizes the optimal control problem.

Chapter 8 defines the preorder for optimal control that preserves the cost and presents the abstraction refinement procedure for constructing finite state systems, which simulate a given transition system, termed the abstract system. The abstract system satisfies the condition that the cost of the optimal control on the abstract system provides an upper bound on the cost of the optimal control for the original system. Furthermore, each suboptimal controller yields trajectories that have the cost upper bounded by the cost of the optimal control on the corresponding abstract system.

Chapter 9 presents the abstraction-refinement method to synthesize control inputs for a discrete-time hybrid system. The controlled system behavior satisfies a finite-word linear-time temporal objective while incurring minimal cost. An abstract finite state weighted transition system is constructed from finite partitions of the state and input spaces by solving optimization problems.

A sequence of suboptimal controllers is obtained by considering a sequence of uniformly refined partitions. In fact, the costs achieved by the sequence of suboptimal controllers converge to the optimal cost for a class of hybrid systems that has robust optimal input trajectories. Examples illustrate the feasibility of this approach to synthesize automatically suboptimal controllers with improving optimal costs.

Chapter 10 presents the abstraction-refinement based framework for optimal controller synthesis of discrete-time hybrid systems with respect to ω-regular objectives. Similar to Chapter 9, it consists of first abstracting the discrete-time "concrete" system into a finite weighted transition system using a finite partition of the state-space. Then, a two-player mean payoff parity game is solved on the product of the abstract system and the Büchi automaton corresponding to the ω-regular objective, to obtain an optimal "abstract" controller that satisfies the ω-regular objective. The abstract controller is guaranteed to be implementable in the concrete discrete-time system, with a sub-optimal cost. The abstraction is refined with finer partitions to reduce the sub-optimality. Under the assumption on the existence of certain robust controllers, the refinement procedure is guaranteed to find controllers whose costs are arbitrarily close to the optimal cost. An example is presented to illustrate the feasibility of the approach.

Appendices B–D contain the full proofs for results in Chapters 8–10, respectively.

Finally, this thesis ends with **Chapter 11**, which summarizes the contributions of this thesis, potential future work, and other final thoughts.

Part I

Optimal Control Synthesis for Linearly Solvable Nonlinear Systems

Chapter 2

INTRODUCTION TO PART I

This part of the thesis presents techniques to solve for optimal controller for a class of stochastic nonlinear affine dynamical systems for three types of cost functionals: first exit, finite horizon, and infinite horizon. In particular, the methods presented seek to solve the Hamilton-Jacobi-Bellman (HJB) equations associated with the optimal control problems. In general, the HJB equation is a nonlinear partial differential equation (PDE), but for a class of systems, termed linearly solvable systems, the HJB equation can be transformed into a linear PDE. The methods described in this part of the thesis solve this linear PDE.

Two techniques are proposed: convex optimization based technique and low rank tensoder decomposition based technique. The former provides performance guarantees with limited scalability, while the latter provides scalability with limited performance guarantees. Each of these methods and its related works are discussed in the individual chapters.

This part of the thesis includes the following chapters:

Chapter 3 introduces background materials necessary for understanding the rest of the chapter in this part, including stochastic control Lyapunov function, linear HJB equation, viscosity solutions, CANDECOMP/PARAFAC tensor, and spectral discretization scheme.

Chapter 4 presents optimal controller synthesis technique that uses Sum-of-Squares program. This technique is the first to combine the optimality condition with the stability criteria in one framework using convex optimization. This technique not only synthesizes a suboptimal controller, but it provides guarantees on trajectory cost and system stability. However, one major limitation of this technique is the curse of dimensionality, which is addressed by the next chapter.

Chapter 5 synthesizes controller for a high dimensional system using low rank tensor decomposition. This technique scales linearly with dimensions and thus avoid the curse of dimensionality. The existing Alternating Least Square algorithm is improved with sequential computation. A MATLAB toolbox is developed.

Appendix A presents implementation of algorithms described in Chapter 5 for multiple engineering examples. The algorithm is able to solve for a quadcopter controller that has 12 degrees of freedom on a laptop.

Part of the contents in this part appeared in these publications [34]–[36].

Chapter 3

PRELIMINARIES

This chapter presents the notations and definitions, and describes several topics that are useful for understanding the following chapters of this part of the thesis.

3.1 Notations and Definitions

A compact domain in $\mathbb{R}^n$ is denoted as Ω, where $\Omega \subset \mathbb{R}^n$, and its boundary is denoted as $\partial\Omega$. A domain Ω is a *basic closed semialgebraic* set if there exists $g_i(x) \in \mathbb{R}(x)$ for $i = 1, 2, \ldots, m$ such that $\Omega = \{x \mid g_i(x) \geq 0 \ \forall i = 1, 2, \ldots, m\}$.

A point on a trajectory, $x(t) \in \mathbb{R}^n$, at time t is denoted $x(t)$, while the segment of this trajectory over the interval $[t, T]$ is denoted by $x(t : T)$.

Given a function $p(x)$, $p(x)$ is positive on domain Ω if $p(x) > 0 \ \forall x \in \Omega$, $p(x)$ is nonnegative on domain Ω if $p(x) \geq 0 \ \forall x \in \Omega$, and $p(x)$ is positive definite on domain Ω, where $0 \in \Omega$, if $p(0) = 0$ and $p(x) > 0$ for all $x \in \Omega \backslash \{0\}$.

3.2 Stochastic Affine Nonlinear Dynamical Systems

This part of the thesis will focus on the following stochastic affine nonlinear dynamical system

$$dx(t) = (f(x(t)) + G(x(t))u(t))\,dt + B(x(t))\,d\omega(t), \tag{3.1}$$

where $x(t) \in \Omega$ is the state at time t in a domain $\Omega \subseteq \mathbb{R}^n$, $u_t \in \mathbb{R}^m$ is the control input, and $f_i(x) \in C^\infty(\Omega) \ \forall i \in [n]_+$, $G_{i,j}(x) \in C^\infty(\Omega) \ \forall i \in [n]_+, j \in [m]_+$, and $B_{i,j}(x) \in C^\infty(\Omega) \ \forall i \in [n]_+, j \in [l]_+$ are smooth functions of the state variables x. The symbol $\omega(t) \in \mathbb{R}^l$ is a vector consisting of Brownian motions with covariance Σ_ε, *i.e.*, ω_t has independent increments with $\omega_t - \omega_s \sim \mathcal{N}(0, \Sigma_\varepsilon(t-s))$, for $\mathcal{N}\left(\mu, \sigma^2\right)$, a normal distribution. The constants n, m, and l are the numbers of states, controller inputs, and noise inputs, respectively. Without loss of generality, let $0 \in \Omega$ and $x = 0$ be the equilibrium point, whereby $f(0) = 0$, $G(0) = 0$, and $B(0) = 0$.

The specific conditions on the functions f, G, and B and the domain Ω will be provided in following chapters, where the techniques to solve the problem are described.

This class of dynamical systems arises in many robotic systems, including quadcopter and other examples shown in later chapters.

3.3 Stochastic Control Lyapunov Functions (SCLF)

The study of system stability is a central theme of control engineering. A primary tool for such studies is Lyapunov theory, wherein an energy-like function is used to show that some measure of distance from a stability point decays over time. The practical machinery for construction of Lyapunov functions that certify system stability advanced considerably with the introduction of Sums of Squares (SOS) programming, which has allowed for Lyapunov functions to be synthesized for both polynomial systems [37] and more general vector fields [38].

To address the more challenging problem of stabilization, rather than the analysis of an existing closed loop system, it is possible to generalize Lyapunov functions to incorporate control inputs. The existence of a control Lyapunov function (CLF) (see [39]–[41]) is sufficient for the construction of a stabilizing controller. However, the synthesis of a CLF for general systems remains an open question. Unfortunately, the SOS-based methods cannot be naively extended to the generation of CLFs, due to the bilinearity between the Lyapunov function and control input.

Due to the lack of a general CLF synthesis technique, an alternative is the use of Receding Horizon Control (RHC), which allows for the incorporation of optimality criteria. Euler-Lagrange equations are used to construct a locally optimum trajectory [42], and stabilization is guaranteed by constraining the terminal cost in the RHC problem to be a CLF. Suboptimal CLFs have found extensive use with applications in legged locomotion [43] and distributed control [44]. Adding stochasticity to the governing dynamics compounds the difficulties of constructing Lyapunov functions [45], [46].

This section introduces the notion of stability for the stochastic affine nonlinear dynamical systems, and the stochastic control Lyapunov function (SCLF).

3.3.1 Stability

Two forms of stability are given, following the definitions in [47, Ch. 5].

Definition 3.1. Given (3.1), the equilibrium point at $x = 0$ is stable in probability for $t \geq 0$ if for any $s \geq 0$ and $\varepsilon > 0$,

$$\lim_{x \to 0} P \left\{ \sup_{t > s} |X^{x,s}(t)| > \varepsilon \right\} = 0,$$

where $X^{x,s}$ is the trajectory of (3.1) from x at time s.

Intuitively, Definition 3.1 is similar to the notion of stability for deterministic systems. The following is a stronger stability definition that is similar to the notion of asymptotic stability for deterministic systems.

Definition 3.2. Given (3.1), the equilibrium point at $x = 0$ is asymptotically stable in probability if it is stable in probability and

$$\lim_{x \to 0} P \left\{ \lim_{t \to \infty} |X^{x,s}(t)| = 0 \right\} = 1,$$

where $X^{x,s}$ is the trajectory of (3.1) from x at time s.

3.3.2 Stochastic Control Lyapunov Functions

The notions of stability introduced earlier can be realized through the construction of stochastic control Lyapunov functions (SCLFs).

Definition 3.3. A stochastic control Lyapunov function for system (3.1) is a positive definite function $\mathcal{V} \in C^{2,1}$ on a domain $O = \Omega \times \{t > 0\}$ such that

$$\mathcal{V}(0,t) = 0, \quad \mathcal{V}(x,t) \geq \mu(|x|) \quad \forall\, t > 0$$
$$\exists\, u(x,t) \text{ s.t. } L(\mathcal{V}(x,t)) \leq 0 \quad \forall\, (x,t) \in O \backslash \{(0,t)\},$$

where $\mu \in \mathcal{K}$, and

$$L(\mathcal{V}) = \partial_t \mathcal{V} + \nabla_x \mathcal{V}^T (f + Gu) + \frac{1}{2} Tr((\nabla_{xx} \mathcal{V}) B \Sigma_\varepsilon B^T). \tag{3.2}$$

Theorem 3.1 ([47] Thm. 5.3)**.** *For system* (3.1)*, assume that there exists a SCLF and a control $u(x,t)$ satisfying Definition 3.3. Then the equilibrium point $x = 0$ is stable in probability, and control $u(x,t)$ is a stabilizing controller.*

To achieve the stronger condition of asymptotic stability in probability, we have the following result.

Theorem 3.2 ([47] Thm. 5.5 and Cor. 5.1)**.** *For system* (3.1)*, suppose that in addition to the existence of a SCLF and a control $u(x,t)$ satisfying Definition 3.3 that control $u(x,t)$ is time-invariant, and*

$$\mathcal{V}(x,t) \leq \mu'(|x|) \quad \forall\, t > 0$$
$$L(\mathcal{V}(x,t)) < 0 \quad \forall\, (x,t) \in O \backslash \{(0,t)\},$$

where $\mu' \in \mathcal{K}$. Then, the equilibrium point $x = 0$ is asymptotically stable in probability, and control $u(x,t)$ is an asymptotically stabilizing controller.

3.4 Linearly Solvable Hamilton-Jacobi-Bellman (HJB) Equation

Given a stochastic dynamical system, an optimal control problem searches for a controller that minimizes a cost functional. The study of the Hamilton-Jacobi-Bellman (HJB) equation that governs the optimal control of a system is central to this problem [48]. Solving the HJB equation is nontrivial because the equation is a second order nonlinear PDE. Methods to calculate the solution to the HJB equation via semidefinite programming have been proposed previously by Lasserre *et al.* [49]. The method is quite general, applicable to any system with polynomial nonlinearities.

Since the late 1970s, Fleming [50], Holland [51] and other researchers thereafter [52], [53] have made connections between stochastic optimal control and reaction-diffusion equation through a logarithmic transformation. Recently, when studying stochastic control using the HJB equation, Kappen [54] and Todorov [55] discovered that particular assumptions on the structure of a dynamical system, given the name *linearly solvable* systems, allow a logarithmic transformation of the optimal control equation to a linear partial differential equation (PDE) form. The linearity of this class of problems has given rise to a growing body of research, with an overview available in [56]. Kappen's work focused on calculating solutions via *path integral* techniques. Todorov began with the analysis of particular Markov decision processes, and showed the connection between the two paradigms. This work was built upon by Theodorou *et al.* [57] into the Path Integral framework in use with Dynamic Motion Primitives. These results have been developed in many different directions [56], [58]–[60].

The rest of this section presents the cost functionals, the associated nonlinear Hamilton-Jacobi-Bellman (HJB) equations, and the linearly solvable HJB equations.

3.4.1 Classes of Cost Functionals

Given the dynamics (3.1), three classes of cost functions are considered: first exit, finite horizon, and infinite horizon.

First Exit

In the first exit problem, the cost functional is

$$J(x,u) = \mathbb{E}_\omega \left[\phi(x(T)) + \int_0^T q(x(t)) + \frac{1}{2} u(t)^T R u(t) \, dt \right], \tag{3.3}$$

where $\phi \in C^\infty(\Omega)$, $\phi : \Omega \to \mathbb{R}^+$ is the final state cost, $q \in C^\infty(\Omega)$, $q : \Omega \to \mathbb{R}_+$ is a state dependent cost, and $R \in \mathbb{R}^{m \times m}$ is a positive definite matrix. The expectation $\mathbb{E}_\omega$ is taken over all realizations of the noise ω in (3.1). The end time T, unknown a priori, is the time when the state reaches the boundary of Ω.

Finite Horizon

In finite horizon problem, the cost functional is

$$J(x, u) = \mathbb{E}_\omega \left[\phi(x(T), T) + \int_0^T q(x(t)) + \frac{1}{2} u(t)^T R u(t)\, dt \right], \tag{3.4}$$

where $\phi \in C^\infty(\Omega \times (0, T])$, $\phi : \Omega \times (0, T] \to \mathbb{R}^+$ is the final state cost, T is a given end time, and the other variables are defined similarly to (3.3).

Infinite Horizon

For infinite horizon, this work considers the average cost functional

$$J(x, u) = \lim_{T \to \infty} \frac{1}{T}\, \mathbb{E}_\omega \left[\int_0^T q(x(t)) + \frac{1}{2} u(t)^T R u(t)\, dt \right], \tag{3.5}$$

where the variables are defined similarly to (3.3). An infinite horizon cost functional is typically used in problems of stabilization. Without lost of generality, we consider stabilization to the origin. Thus, $q(\cdot)$ and $\phi(\cdot)$ are chosen to be positive definite functions.

3.4.2 Hamilton-Jacobi-Bellman Equation

The solution to the minimization problem given (3.1) and a cost functional is known as the *value function*, $V : \Omega \to \mathbb{R}_+$, where beginning from an initial point $x(t)$ at time t

$$V(x(t)) = \min_{u(t:T)} J(x(t : T), u(t : T)).$$

The value function is also dependent on time t for the finite horizon problem.

Based on the dynamic programming principle [61, Ch. III.7], the minimization problem for each cost functional given the system (3.1) has an associated HJB equation whose solution forms the optimal controller that minimizes the cost functional. The associated HJB equation for (3.3), (3.4), and (3.5) are nonlinear, second order partial differential equations (PDE):

$$\begin{aligned} \text{First exit:} \qquad 0 \quad &= \mathfrak{NL}(V) && (3.6a)\\ \text{Finite horizon:} \qquad -\partial_t V &= \mathfrak{NL}(V) && (3.6b)\\ \text{Infinite horizon:} \qquad c \quad &= \mathfrak{NL}(V), && (3.6c)\end{aligned}$$

where

$$\mathfrak{NL}(V) \triangleq q + (\nabla_x V)^T f - \frac{1}{2}(\nabla_x V)^T G R^{-1} G^T (\nabla_x V) + \frac{1}{2} Tr\left((\nabla_{xx} V)\, B \Sigma_\varepsilon B^T\right),$$

and the variable c is the optimal average cost that does not depend on the states x. For all three problems, the optimal control effort, u^*, is given by

$$u^* = -R^{-1} G^T \nabla_x V. \tag{3.7}$$

3.4.3 Linear Hamilton-Jacobi-Bellman Equation

Solving (3.6) is difficult due to its nonlinearity. But, when this equality holds

$$\lambda G(x_t) R^{-1} G(x_t)^T = B(x_t) \Sigma_\varepsilon B(x_t)^T \triangleq \Sigma(x_t) \triangleq \Sigma_t \tag{3.8}$$

for a $\lambda > 0$, the nonlinear PDE can be transformed into a linear PDE [55], [62], [63] using this logarithmic transformation

$$V = -\lambda \log \Psi. \tag{3.9}$$

Remark. Systems of the form $dx(t) = f(x(t))\, dt + G(x(t))\, (u(t)\, dt\ + d\omega(t))$ that are common in the adaptive control literature [64] will trivially satisfy (3.8). This constraint restricts the design of the control penalty R, such that control effort is highly penalized in subspaces with little noise, and lightly penalized in those with high noise. Additional discussion is given in [55, SI Sec. 2.2].

After substituting (3.8) and (3.9) into (3.6), and simplifying, the HJB equations become

$$\begin{aligned} \text{First exit:} \qquad 0 \quad &= \mathfrak{L}(\Psi) && (3.10a)\\ \text{Finite horizon:} \qquad -\partial_t \Psi &= \mathfrak{L}(\Psi) && (3.10b)\\ \text{Infinite horizon:} \qquad -c\Psi &= \mathfrak{L}(\Psi), && (3.10c)\end{aligned}$$

where

$$\mathfrak{L}(\Psi) \triangleq -\frac{1}{\lambda} q \Psi + f^T (\nabla_x \Psi) + \frac{1}{2} Tr\left((\nabla_{xx} \Psi)\, \Sigma_t\right).$$

The function Ψ is called the *desirability* function [55].

The equations in (3.10) are not well-posed PDE problems without specifying the boundary conditions [65]. The boundary conditions are defined as follows:

$$\text{First exit:} \quad \Psi(x) = \exp\left(-\frac{1}{\lambda}\phi(x)\right), \text{ for } x \in \partial\Omega \tag{3.11a}$$

$$\text{Finite horizon:} \quad \Psi(x,t) = \exp\left(-\frac{1}{\lambda}\phi(x,t)\right), \text{ for } (x,t) \in \partial\Omega \times (0,T] \tag{3.11b}$$

$$\Psi(x,T) = \exp\left(-\frac{1}{\lambda}\phi(x,T)\right), \text{ for } x \in \Omega$$

$$\text{Infinite horizon:} \quad \Psi(x) = 0, \text{ for } x \in \partial\Omega. \tag{3.11c}$$

Henceforth, we write $\Psi(\cdot) = \psi(\cdot)$ as a shorthand for the boundary conditions, and the specific definition of $\psi(\cdot)$ depends on the class of cost functions, which should be clear from the context. For infinite horizon problem, the domain is chosen to be large enough such that $V(x)$ is a large number at the boundary, and thus $\Psi(x)$ is close to zero at the boundary. The boundary conditions currently defined are called Dirichlet boundary conditions. However, other types of boundary conditions can also be imposed, including the periodic boundary condition for the dimension corresponding to the angle [65].

3.5 Viscosity Solutions of Partial Differential Equations (PDE)

If the linear HJB (3.10) is not uniformly elliptic/parabolic [66], a classical solution may not exist. The notion of viscosity solutions is developed to generalize the classical solution. Refer to [66] for a general discussion on viscosity solutions and [61] for a discussion on viscosity solutions related to Markov diffusion processes.

The first definition applies to elliptic PDE.

Definition 3.4 ([66] Def. 2.2)**.** Given $\Omega \subset \mathbb{R}^N$ and an elliptic partial differential equation

$$F(x, u, \nabla_x u, \nabla_{xx} u) = 0, \tag{3.12}$$

where $F : \mathbb{R}^N \times \mathbb{R} \times \mathbb{R}^N \times \mathcal{S}(N) \to \mathbb{R}$, $\mathcal{S}(N)$ is the set of real symmetric $N \times N$ matrices, and F satisfies

$$F(x, r, p, X) \leq F(x, s, p, Y) \text{ whenever } r \leq s \text{ and } Y \leq X,$$

then a **viscosity subsolution** of (3.12) on Ω is a function $u \in USC(\Omega)$ such that

$$F(x, u, p, X) \leq 0 \quad \forall\, x \in \Omega, (p, X) \in J_\Omega^{2,+} u(x).$$

Similarly, a **viscosity supersolution** of (3.12) on Ω is a function $u \in LSC(\Omega)$ such that

$$F(x, u, p, X) \geq 0 \quad \forall\, x \in \Omega, (p, X) \in J^{2,-}_{\Omega} u(x).$$

Finally, u is a **viscosity solution** of (3.12) on Ω if it is both a viscosity subsolution and a viscosity supersolution in Ω.

The notations $USC(\Omega)$ and $LSC(\Omega)$ represent the sets of upper and lower semicontinuous functions on domain Ω, respectively, and $J^{2,+}_{\Omega} u(x)$ and $J^{2,-}_{\Omega} u(x)$ represents the second order "superjets" and "subjets" of u at x, respectively. These "semi-jets" are approximations of the derivatives when solution is not differentiable. A more formal definition is available in [66].

The next definition applies to parabolic PDEs.

Definition 3.5 ([66] Sec. 8)**.** Let $\mathcal{O} = (0, T) \times \Omega$, where $\Omega \subset \mathbb{R}^N$. Given a parabolic partial differential equation

$$\partial_t u + F(t, x, u, \nabla_x u, \nabla_{xx} u) = 0, \tag{3.13}$$

where $F : [0, T] \times \mathbb{R}^N \times \mathbb{R} \times \mathbb{R}^N \times \mathcal{S}(N) \to \mathbb{R}$, $\mathcal{S}(N)$ is the set of real symmetric $N \times N$ matrices, and F satisfies

$$F(t, x, r, p, X) \leq F(t, x, s, p, Y)$$

whenever $r \leq s$ and $Y \leq X$ for each $t \in [0, T)$, then a **viscosity subsolution** of (3.13) on $\mathcal{O}$ is a function $u \in USC(\mathcal{O})$ such that

$$a + F(t, x, u, p, X) \leq 0\; \forall\, (t, x) \in \mathcal{O}, (a, p, X) \in P^{2,+}_{\mathcal{O}} u(t, x).$$

Similarly, a **viscosity supersolution** of (3.13) on $\mathcal{O}$ is a function $u \in LSC(\mathcal{O})$ such that

$$a + F(t, x, u, p, X) \geq 0\; \forall\, (t, x) \in \mathcal{O}, (a, p, X) \in P^{2,-}_{\mathcal{O}} u(t, x).$$

Finally, u is a **viscosity solution** of (3.13) on $\mathcal{O}$ if it is both a viscosity subsolution and a viscosity supersolution in $\mathcal{O}$.

The notations $P^{2,+}_{\mathcal{O}} u(t, x)$ and $P^{2,-}_{\mathcal{O}} u(t, x)$ represents the second order "superjets" and "subjets" of u at (t, x), respectively.

3.6 Sums-of-Squares (SOS) Programming

Sum-of-Squares (SOS) programming is a convex optimization technique that is widely used when the problem involves positivity of polynomials. One popular application of the SOS programming in control is for Lyapunov stability analysis [67]. A complete introduction to SOS programming is available in [37].

3.6.1 Brief Introduction

This section reviews the basic definition of SOS that is used throughout the thesis.

Definition 3.6. A multivariate polynomial $f(x)$ is a SOS polynomial if there exist polynomials $f_0(x), \ldots, f_m(x)$ such that

$$f(x) = \sum_{i=0}^{m} f_i^2(x).$$

The set of SOS polynomials in x is denoted as $\mathcal{SOS}(x)$.

Accordingly, a sufficient condition for nonnegativity of a polynomial $f(x)$ is that $f(x) \in \mathcal{SOS}(x)$. Membership in the set $\mathcal{SOS}(x)$ may be tested as a convex problem [37].

Theorem 3.3 ([37] Thm. 3.3)**.** *The existence of a SOS decomposition of a polynomial in n variables of degree* $2d$ *can be decided by solving a semidefinite programming (SDP) feasibility problem. If the polynomial is dense (no sparsity), the dimension of the matrix inequality in the SDP is equal to* $\binom{n+d}{d} \times \binom{n+d}{d}$.

Therefore, by restricting the set of all positive polynomials to be SOS, testing nonnegativity of a polynomial becomes a tractable SDP. The converse question "is a nonnegative polynomial necessarily a SOS" is unfortunately false, indicating that this test is conservative [37]. Theorem 3.3 guarantees a tractable procedure to determine whether a particular polynomial, possibly parameterized, is a SOS polynomial.

Multiple polynomial constraints can be combined into an optimization formulation. To do so, define the following polynomial sets.

Definition 3.7. The preordering of polynomials $g_i(x) \in \mathcal{R}(x)$ for $i = 1, 2, \ldots, m$ is the set

$$P(g_1, \ldots, g_m) = \left\{ \sum_{\nu \in \{0,1\}^m} s_\nu(x) g_1(x)^{\nu_1} \cdots g_m(x)^{\nu_m} \;\middle|\; s_\nu \in \mathcal{SOS}(x) \right\}. \qquad (3.14)$$

The following proposition is trivial, but it is useful to incorporate the domain Ω in optimization formulation.

Proposition 3.1. *Given $f(x) \in \mathcal{R}(x)$ and the domain*

$$\Omega = \{x \mid g_i(x) \in \mathbb{R}(x), g_i(x) \geq 0, i \in [m]_+\},$$

if $f(x) \in P(g_1, \ldots, g_m)$, then $f(x)$ is nonnegative on Ω. If there exists another polynomial $f'(x)$ such that $f'(x) \geq f(x)\ \forall x \in \Omega$, then $f'(x)$ is also nonnegative on Ω.

Proof. Because $g_i(x)$ and $s_i(x)$ are nonnegative, all functions in $P(\cdot)$ are nonnegative. The second statement is trivially true given the first statement. □

Example. To illustrate an application of Proposition 3.1, consider a polynomial $f(x)$ defined on the domain $x \in [-1, 1]$. The bounded domain can be equivalently defined by polynomials with $g_1(x) = 1+x$ and $g_2(x) = 1-x$. To certify that $f(x) \geq 0$ on the specified domain, construct a function $h(x) = s_1(x)(1 + x) + s_2(x)(1 - x) + s_3(x)(1 + x)(1 - x)$, where $s_i \in \mathcal{SOS}(x)$ and certify that $f(x) - h(x) \geq 0$. Notice that $h(x) \in P(1 + x, 1 - x)$, so $h(x) \geq 0$. If $f(x) - h(x) \geq 0$, then $f(x) \geq h(x) \geq 0$. Proposition 3.1 is applied here. Finding the correct $s_i(x)$ is not trivial in general. Nonetheless, as mentioned earlier, if we further impose that $f(x) - h(x) \in \mathcal{SOS}(x)$, then the process of checking if there exists $s_i(x)$ such that $f(x) - h(x) \in \mathcal{SOS}(x)$ becomes a SDP as given by Theorem 3.3.

To simplify notation in the remainder of this thesis, given a domain $\Omega = \{x \mid g_i(x) \in \mathcal{R}(x), g_i(x) \geq 0, i \in \{1, 2, \ldots, m\}\}$, we set the notation $P(\Omega) = P(g_1, \ldots, g_m)$.

Remark. Depending on the computational resources available, one may choose a subset of $P(\Omega)$ to reduce the size of the resulting SDP. However, the chances of finding a certificate are reduced as a consequent. This polynomial set is often used in the discussions of Schmüdgen's Positivstellensatz, which states that if $f(x)$ is positive on a compact domain Ω, then $f(x) \in P(\Omega)$ [37], [49].

3.6.2 Hierarchy of Sums-of-Squares (SOS) Programming

This thesis uses the SOS programming to solve for positive polynomials that minimize a given cost function while satisfying some constraints. The constrained SOS program typically has the following form:

$$\begin{aligned} \min_{\varepsilon, \{f_i(x)\}_{i\in[k]}} \quad & c^{\mathrm{T}}\varepsilon \\ s.t. \quad & \varepsilon_i - f_i(x) \in \mathcal{SOS}(x)\ \forall i \in [k] \\ & g_j = 0\ \forall j \in [l], \end{aligned} \tag{3.15}$$

where $c, \varepsilon \in \mathbb{R}^k$ is a vector, $f_i \in \mathcal{R}(x)$ are real polynomials in x, and g_j are a linear functions of the coefficients of f_i.

When the polynomial degrees for f_i are fixed, this optimization problem is convex and solvable using a SDP via Theorem 3.3. To systematically solve for the polynomials, a hierarchy of SOS programs with increasing polynomial degree is formed.

Let d be the maximum degree of f_i for all $i \in [k]$, and denote $(\varepsilon^d, \{f_i^d\}_{i\in[k]})$ as a solution to (3.15) when the maximum polynomial degree is fixed at d. The hierarchy of SOS programs with increasing polynomial degree produces a sequence of (possibly empty) solutions $(\varepsilon^d, \{f_i^d\}_{i\in[k]})_{d\in I}$, where $I \subset \mathbb{Z}_+$.

If solutions exist for d and d' such that $d > d'$, then $\varepsilon^d \leq \varepsilon^{d'}$ because the set of lower degree polynomials is a subset of the set of higher degree polynomials. Therefore, one could keep increasing the degree of polynomials in order to achieve tighter solutions. The use of such hierarchies is commonplace in polynomial optimization [37], [68]. If at certain degree, $\varepsilon^d = 0$, the optimal solutions $\{f_i\}_{i\in[k]}$ are found.

3.7 Spatial and Time Discretization

This section describes the state space and the time discretization scheme employed in later chapters beginning with the spatial discretization.

3.7.1 Spatial Discretization

The spatial discretization can be performed in many ways. This section will describes two of those: uniform finite difference scheme and spectral discretization scheme.

Uniform Finite Difference Discretization

Given a continuous state space, the simplest form of discretization scheme is the uniform finite difference scheme [69], [70].

For each dimension of the continuous space Ω, a function is approximated at uniform nodes as such,

$$X_i(k) = a_i + \frac{b_i - a_i}{N_i - 1}k$$

for $k = 0, 1, \ldots, M_i - 1$, where $X_i(k)$ is the k-th point in the i-th dimension of the domain, N_i is the total number of points in the i-th dimension, and a_i and b_i are the lower and upper bound of the domain in the i-th dimension, respectively.

As a result, a function $f(x)$ on a domain $x \in \Omega \subset \mathbb{R}^n$ is discretized in space to form a n-dimensional tensor $\mathcal{T}$ of size $N_1 \times N_2 \times \ldots \times N_n$, where $\mathcal{T}(k_1, \ldots, k_n) = f(X_1(k_1), X_2(k_2), \ldots, X_n(k_n))$. Note that the size of the tensor scales exponentially with the number of dimensions. Thus, a naive discretization suffers from the curse of dimensionally. However, for a separable function, the tensor is naturally decomposable into a CP tensor as defined in Definition 3.8 avoiding the curse of dimensionality. As the number of points per dimension increases, the approximation becomes more accurate, but the computation cost also increases.

Given this discretization scheme, a derivative of a function can be performed numerically via the finite difference differentiation matrix [71].

Spectral Discretization

Spectral discretization is typically used for its superior accuracy and convergence. The spectral methods converges exponentially instead of algebraic convergence rates for finite difference and finite element methods. Therefore, good accuracy can be obtained with coarse discretization. However, the spectral method has tighter stability restrictions, and the matrices are dense. For more details, refer to [72] and references therein.

For each dimension of the continuous space Ω, a function is approximated at the Chebyshev-Gauss-Lobatto nodes [73] as such,

$$X_i(k) = \frac{a_i + b_i}{2} + \frac{b_i - a_i}{2} \cos\left(\frac{k\pi}{N_i - 1}\right)$$

for $k = 0, 1, \ldots, N_i - 1$, where $X_i(k)$ is the k-th point on the i-th dimension of the domain, N_i is the total number of points in the i-th dimension, and a_i and b_i are the lower and upper bound of the domain in the i-th dimension, respectively.

In some cases, the domain for a specific dimension is periodic, for example, when the state represents angle. If the domain for the i-th dimension is periodic, instead of

Chebyshev-Gauss-Lobatto nodes, the function is approximated at the Fourier nodes [73] as such,

$$X_i(k) = a_i + (b_i - a_i)\frac{k}{N_i}$$

for $k = 0, 1, \ldots, N_i - 1$, where $X_i(k)$ is the k-th point for the i-th dimension, N_i is the total number of points in the i-th dimension, and a_i and b_i are the lower and upper bound of the domain in the i-th dimension, respectively.

Similarly, a function $f(x)$ on a domain $x \in \Omega \subset \mathbb{R}^n$ is discretized in space to form a n-dimensional tensor $\mathcal{T}$ of size $N_1 \times N_2 \times \ldots \times N_n$, where $\mathcal{T}(k_1, \ldots, k_n) = f(X_1(k_1), X_2(k_2), \ldots, X_n(k_n))$.

For both the Chebyshev-Gauss-Lobatto nodes and the Fourier nodes, the associated differentiation matrices are dense, unlike the finite difference differentiation matrix, which is sparse. The details of the differentiation matrices including their actual forms can be found in [73].

3.7.2 Time Discretization

Time stepping is necessary for solving the non-stationary HJB equation [74]. Forward and backward Euler methods are implemented in this thesis. Forward Euler is one the simplest form of explicit methods for time integration. Given a PDE $\partial_t u(x,t) = f(u(x,t))$, where f is a linear operator, an initial value $u_0(x)$, and boundary conditions, the forward Euler method solves for the solution at the next time step based on

$$\bar{u}_{k+1} = (I + Fh)\bar{u}_k, \tag{3.16}$$

where $\bar{u}_k$ is the discretization of $u(x,t)$ at time $t = kh$, F is the discretization of f, I is an identity operator with the appropriate size, and h is the time increment. Depending on the boundary conditions, (3.16) may have slightly different forms [74].

On the other hand, backward Euler is an implicit method with better numerical stability, but with higher computation cost than forward Euler and other explicit methods. More concretely, given a PDE $\partial_t u(x,t) = f(u(x,t))$, where f is a linear operator, an initial value $u_0(x)$, and boundary conditions, the backward Euler method solves for the solution at the next time step based on

$$(I - Fh)\bar{u}_{k+1} = \bar{u}_k, \tag{3.17}$$

where the variables are defined similar to (3.16). As before, depending on the boundary conditions, (3.17) may have slightly different forms [74].

Other choices of time discretization [74] such as leapfrog integration and Runge-Kutta can also be implemented in the framework discussed in this thesis. Refer to [74] for a more detailed discussion on spatial and time discretization.

3.8 Low Rank Tensor Decomposition

Low rank tensor decomposition is a technique to approximate a high-dimensional tensor that may not be low rank with a low rank tensor [75]. Multiple representation of low rank tensors are developed over the past years including CANDECOMP/PARAFAC (CP) tensor [76], [77], Tucker tensor [78], tensor train [79], and function train [80]. This thesis uses the CP tensor as a framework to approximate high dimensional functions in order to avoid the curse of dimensionality.

3.8.1 CANDECOMP/PARAFAC Tensor

The CANDECOMP/PARAFAC (CP) tensor is used to represent separable functions and operators that are discretized in space [75]–[77], [81].

The tensor product of two vectors u and v is written as $u \bigotimes v \triangleq w$, where $w_{ij} = u_i v_j$. The inner product of two vectors u and v is written as $\langle u, v\rangle$, where $\langle u, v\rangle = \sum_i u_i v_i$.

Definition 3.8. Given a separable continuous function $f(x) = \sum_{l=1}^{r} s_i \prod_{i=1}^{d} f_i^l(x_i)$, where $x \in \mathbb{R}^d$, the discretized function F is a CP tensor that is defined as

$$F = \sum_{l=1}^{r} s_l \bigotimes_{i=1}^{d} F_i^l,$$

where $F_i^l \in \mathbb{R}^{n_i}$ is a unit vector that represents the function $f_i^l(x)$ at n_i discretization points in the i-th dimension, s_l is a normalizing constant, r is the separation rank, and d is the dimension of the tensor. Each F_i^l is called a **basis function** in dimension i, and each summand $s_l^F \bigotimes_{i=1}^{d} F_i^l$ is called a **tensor term**.

By approximating the function f with a tensor function F, the number of points for storage increases linearly with dimension d for a given r, and linearly with r for a given d. Dimension d is usually determined by applications. As such, obtaining low rank approximations (small r) is vital for feasible computations. Nonetheless, a rank that is too low results in inaccurate approximations. Therefore, a balance between feasible computations and accurate approximations is a necessary consideration when determining suitable ranks.

A tensor operator is defined equivalently except that the F_i^l is a n by n matrix instead of a vector.

Definition 3.9. Given a separable linear operator $\mathcal{A}(f) = \sum_{l=1}^{r} s_i \prod_{i=1}^{d} \mathcal{A}_i^l(f)$, where $f \in \mathbb{R}(\Omega)$ is a real function that $\mathcal{A}$ acts on and $\Omega \subseteq \mathbb{R}^d$, the discretized operator $\mathbb{A}$ is a CP tensor that is defined as

$$\mathbb{A} = \sum_{l=1}^{r} s_l \bigotimes_{i=1}^{d} A_i^l,$$

where $A_i^l \in \mathbb{R}^{n_i \times n_i}$ is a normalized matrix (with respect to Frobenius norm) that represents the operator $\mathcal{A}_i^l(f)$ for n_i discretization points in the i-th dimension, s_l is a normalizing constant, r is the separation rank, and d is the dimension of the tensor.

We refer to the function in tensor form as tensor function, and the operator in tensor form as tensor operator. This representation needs $O(nrd)$ in space, and most algebraic computations scale linearly with dimensions [81].

The tensor operator and tensor function multiplication operation is

$$\mathbb{A}F = \sum_{m=1}^{r_A} \sum_{l=1}^{r_F} s_m^A s_l^F \bigotimes_{i=1}^{d} A_i^m F_i^l, \tag{3.18}$$

where the computation cost is $O(r_A r_F d n^2)$ assuming the number of points per dimension $n_i = n$ for all i. The inner product of two tensors F and G is given by

$$\langle G, F \rangle = \sum_{m=1}^{r_G} \sum_{l=1}^{r_F} s_m^G s_l^F \bigotimes_{i=1}^{d} \langle G_i^m, F_i^l \rangle, \tag{3.19}$$

where the computation cost is $O(r_G r_F d n)$. Given the inner product, the norm of a tensor function F is defined as $\|F\| = \sqrt{\langle F, F \rangle}$. A more detailed descriptions are available in [81].

For most linear algebra operations, the separation rank of the result often increases. For example, (3.18) increases the rank from r_F to $r_A r_F$. Therefore, after performing an operation, a low rank approximation of the resulting tensor is vital for feasible computations. Next, the algorithm used to produce low rank approximation, the ALS algorithm, is discussed.

Tensor decomposition is implemented numerically using the MATLAB Tensor Toolbox [82], [83].

3.8.2 Alternating Least Squares (ALS) Algorithm

This section describes the Alternating Least Squares (ALS) algorithm [81] that solves the linear equation in which the operator and the function are in tensor form. When the operator is an identity operator, this algorithm reduces the separation rank.

Formally, given a tensor function G and a tensor operator $\mathbb{A}$, ALS solves for F in

$$\mathbb{A}F = G \tag{3.20}$$

by minimizing $\|\mathbb{A}F - G\|$ for a fixed rank of F in which $\mathbb{A}$, F, and G are represented in tensor decomposition form

$$F = \sum_{l=1}^{r_F} \bigotimes_{i=1}^{d} F_i^l, \quad \mathbb{A} = \sum_{l=1}^{r_A} \bigotimes_{i=1}^{d} A_i^l, \quad G = \sum_{l=1}^{r_G} \bigotimes_{i=1}^{d} G_i^l,$$

where $F_i^l \in \mathbb{R}^{n_i}$, $G_i^l \in \mathbb{R}^{n_i}$, and $A_i^l \in \mathbb{R}^{n_i \times n_i}$. Note that here we do not require F_i^l, G_i^l, and A_i^l to have unit norm.

The minimum of the residual is achieved when the gradient of the residual with respect to F is zero, that is $\nabla_F \|\mathbb{A}F - G\| = 0$ for the minimum F, where ∇_F denotes the gradient with respect to all elements in F_i^l for $i = 1, \ldots, d$ and $l = 1, \ldots, r_F$. The gradient is not linear with respect to the terms in F. Thus, the algorithm first fixed a particular dimension k and solves for F_k^l assuming all other F_i^l are fixed for $i \neq k$, then it cycles through all the dimensions. As a result, for each dimension k, the following simple linear equation, called the normal equation, is solved.

$$\mathcal{M}\mathcal{F}_k = \mathcal{N}, \tag{3.21}$$

where

$$\mathcal{M} = \begin{pmatrix} M_{1,1} + \alpha I & M_{1,2} & \cdots & M_{1,r_F} \\ M_{2,1} & M_{2,2} + \alpha I & \cdots & M_{2,r_F} \\ \vdots & \vdots & \ddots & \vdots \\ M_{r_F,1} & M_{r_F,2} & \cdots & M_{r_F,r_F} + \alpha I \end{pmatrix},$$

$$\mathcal{F}_k = \begin{pmatrix} F_k^1 \\ F_k^2 \\ \vdots \\ F_k^{r_F} \end{pmatrix}, \quad \mathcal{N} = \begin{pmatrix} N_1 \\ N_2 \\ \vdots \\ N_{r_F} \end{pmatrix}$$

and $M_{i,j}$ and N_i are given by

$$\begin{aligned} M_{i,j} &= \sum_{i_A=1}^{r_A} \sum_{j_A=1}^{r_A} (A_k^{j_A})^T A_k^{i_A} \prod_{m \neq k} \langle A_m^{i_A} F_m^j, A_m^{j_A} F_m^i \rangle \\ N_i &= \sum_{i_A=1}^{r_A} \sum_{i_G=1}^{r_G} (A_k^{i_A})^T G_k^{i_G} \prod_{m \neq k} \langle A_m^{i_A} F_m^i, G_m^{i_G} \rangle. \end{aligned} \tag{3.22}$$

The ALS algorithm can be ill-conditioned in general. Thus, the term α in the normal equation acts as a regularizer [81]. Furthermore, references [35], [84] also proposed techniques to prevent ill-conditioned computation.

The vanilla ALS algorithm is summarized by Algorithm 1. First, the function *RandomTensor* creates a normalized random tensor of rank r_0. In other words, *RandomTensor* generates unit norm random vectors $F_j^l \in \mathbb{R}^{N_j}$ for $j \in [d]_+$ and $l \in [r_0]_+$, then sets $F = \sum_{l=1}^{r_0} \bigotimes_{j=1}^{d} F_j^l$. The function *ComputeResidual* computes the residual of the current solution by

$$res = \frac{\|\mathbb{A}F - G\|}{\sqrt{\prod_{i=1}^{d} n_i}}.$$

If the residual is smaller than a pre-specified tolerance ε_0, the algorithm terminates and return the solution F. Otherwise, the F will be updated. For each dimension k, *SolveNormal* solves (3.21) for the vector $\mathcal{F}_k$ and update F. It also returns l_k that indicates if (3.21) is ill-conditioned. If l_k is true for any k, the algorithm terminates without finding a F that satisfies the accuracy tolerance. Otherwise, the algorithm continues. If the difference between the residual from the previous iteration and the residual from the current iteration is smaller than the accuracy tolerance ε, a new rank-one random tensor function F^r is created. The random tensor function F^r is pre-conditioned by iterating *SolveNormal* for $\mathbb{A}(F + F^r) = G$ with fixed F (*i.e.,* by computing *SolveNormal*$(\mathbb{A}, F^r, G - \mathbb{A}F)$ for each dimension) to prevent F^r from dominating the approximate solution F. For more details on the ALS algorithm, refer to [81]. For the rest of the thesis, an iteration of Algorithm 1 refers to one iteration of the for-loop (line 5-7).

The special case $\mathbb{A} = \mathbb{I}$, an identity operator, is used to find low rank approximations for both tensor functions and tensor operators. The latter can be achieved by storing the operator matrices A_i^l as vectors, and performing the ALS algorithm as if it was a tensor function.

Algorithm 1 *ALS* Algorithm

```
Input: Tensor operator $\mathbb{A}$, tensor function $G$, accuracy tolerance $\varepsilon$, initial rank $r_0$
Output: Tensor function $F$
1:  $F := RandomTensor(r_0)$
2:  $res := ComputeResidual(\mathbb{A}, F, G)$
3:  while $res > \varepsilon$ do
4:      $res' := res$
5:      for $k = 1, 2, \dots, d$ do
6:          $F, l_k := SolveNormal(\mathbb{A}, F, G)$
7:      end for
8:      $res := ComputeResidual(\mathbb{A}, F, G)$
9:      if $l_k$ is True for any $k \in [d]_+$ then
10:         Terminate the algorithm and indicate that $F$ is not solved successfully
11:     else if $|res - res'| < \varepsilon$ then
12:         $F^r := PreRandomTensor(1)$
13:         $F := F + F^r$
14:     end if
15: end while
```

Chapter 4

OPTIMAL CONTROLLER SYNTHESIS USING SUM-OF-SQUARES

This chapter proposes an optimal controller synthesis technique based on convex optimization for the approximate solution to the linear HJB equation. This technique combines previously disparate fields of linearly solvable optimal control and Lyapunov theory, and provides a systematic way to construct stabilizing controllers with guaranteed performance. The result is a hierarchy of SOS programs that generate SCLFs for arbitrary linearly solvable systems. Such an approach has many benefits. First and foremost, this approach generates stabilizing controllers for an important class of nonlinear, stochastic systems even when the optimal controller is not found. We prove that the approximate solutions generated by the SOS programs are pointwise upper and lower bounds to the true solutions. In fact, the upper bound solutions are SCLFs, which can be used to construct stabilizing controllers, and they bound the performance of the system when they are used to construct suboptimal controllers. Existing methods for the generation of SCLFs do not have such performance guarantees. Additionally, we demonstrate that, although the technique is based on linear solvability, it may be readily extended to more general systems, including deterministic systems, while inheriting the same performance guarantees.

A preliminary version of this work appeared in [85] and [86], where the use of SOS programming for solving the HJB were first considered. This paper builds on this recent body of research, studying the stabilization and optimality properties of the resulting solutions. These previous works focused on path planning, rather than stabilization, and did not include the stability analysis or suboptimality guarantees presented in this chapter. Furthermore, the analysis and results are extended to the finite horizon problem that involves time evolving linear HJB equation. Some content of this work appeared in [34], [36].

The rest of this chapter is organized as follows. Section 4.1 introduces the problem formulation and assumptions. Section 4.2 introduces a relaxed formulation of the HJB solutions, which is efficiently computable using the SOS methodology for the first exit problem. Section 4.3 analyzes the properties of the relaxed solutions, such as approximation errors relative to the exact solutions. This section shows that the relaxed solutions are SCLFs, and that the resulting controller is stabilizing. The upper bound solution is also shown to bound the performance when using the sub-optimal controller. Section 4.4 presents the controller synthesis procedure for finite horizon problem, and Section 4.5 discusses the properties of the controller that is synthesized. Section 4.6 summarizes the extension of the method to deterministic systems and Section 4.7 considers the extension to the robust optimal control problems. Two examples are presented in Section 4.8 to illustrate the optimization technique and its performance. Section 4.9 summarizes the findings of this work and discusses future research directions.

4.1 Problem Formulation

This chapter considers synthesizing an optimal controller for system (3.1) with respect to first exit cost functional and finite horizon cost functional described in Section 3.4.1. The systems dynamics and the cost functionals are assumed to be governed by polynomial functions.

Assumption 4.1. System (3.1) and the cost functionals are described by polynomials. In other words, f, G, B, ϕ, and q consist of polynomials.

Although the system dynamics are limited to polynomials, the non-polynomial nonlinearities can be incorporated by projecting the non-polynomial functions to a polynomial basis. As polynomials are universal approximators in $\mathfrak{L}_2$ by the Stone-Weierstrass Theorem [87], this approximation can be made to arbitrary accuracy if the functions are continuous and the domain is bounded. A limited basis may introduce modeling error, but this may be dealt with via the robust optimization techniques outlined in Section 4.7.

For stabilization to the origin, the following assumption is imposed.

Assumption 4.2. The functions q and ϕ in the cost functionals are positive definite functions.

Lastly, the following assumption on the domain of (3.1) is necessary in moment and SOS-based methods [37], [49].

Assumption 4.3. System (3.1) evolves on a compact domain $\Omega \subset \mathbb{R}^n$, and Ω is a basic closed semialgebraic set such that $\Omega = \{x \mid g_i(x) \in \mathcal{R}(x), g_i(x) \geq 0, i \in [k]_+\}$ for some $k \geq 1$. Then, the boundary $\partial\Omega$ is polynomial representable: $\partial\Omega = \{x \mid h_i(x) \in \mathcal{R}(x), \prod_{i=1}^{k'} h_i(x) = 0\}$ for some $k' \geq 1$.

The following definitions formalize several operators that are useful in the sequel, in particular, when constructing the relevant sets for using Definition 3.7 and Proposition 3.1.

Definition 4.1. Given a basic closed semialgebraic set $\Omega = \{x \mid g_i(x) \in \mathcal{R}(x), g_i(x) \geq 0, i \in [k]_+\}$ and a set of SOS polynomials,

$$S = \{s_\nu(x) \mid s_\nu(x) \in \mathcal{SOS}(x), \nu \in \{0,1\}^k\},$$

define the operator $\mathcal{D}$ as

$$\mathcal{D}(\Omega, S) = \sum_{\nu \in \{0,1\}^k} s_\nu(x) g_1(x)^{\nu_1} \cdots g_k(x)^{\nu_k},$$

where $\mathcal{D}(\Omega, S) \in P(\Omega)$.

Definition 4.2. Given a polynomial inequality, $p(x) \geq 0$ defined on Ω, the boundary of a compact set $\partial\Omega = \{x \mid h_i(x) \in \mathcal{R}(x), \prod_{i=1}^{k'} h_i(x) = 0\}$ and a set of polynomials,

$$T = \{t_i(x) \mid t_i(x) \in \mathcal{R}(x), i \in [k']_+\},$$

define the operator $\mathcal{B}$ as

$$\mathcal{B}(p(x), \partial\Omega, T) = \{p(x) - t_i(x) h_i(x) \mid i \in [k']_+\},$$

where $\mathcal{B}$ returns a set of polynomials that is nonnegative on $\partial\Omega$.

For the remainder of this chapter, we assume a unique nontrivial viscosity solution to (3.6) and (3.10) exists (see [61], Chapter V) and denote them as V^* and Ψ^* respectively.

4.2 Controller Synthesis for First Exit Problem

Given the problem formulation described earlier, we can obtain the linear HJB equations (3.10), where the components in $\mathfrak{L}(\cdot)$ are real polynomial functions. The definition of the function $\mathfrak{L}(\cdot)$ is reproduced here for convenience:

$$\mathfrak{L}(\Psi) \triangleq -\frac{1}{\lambda} q \Psi + f^T (\nabla_x \Psi) + \frac{1}{2} Tr\left((\nabla_{xx} \Psi) \Sigma_t\right).$$

This section proceeds with the technique to solve (3.10) using SOS programming. SOS programming has found many uses in combinatorial optimization, control theory, and other applications. This section now adds solving the linear HJB to this list. This section focuses on the first exit problem.

4.2.1 Relaxation of the First Exit HJB Equation

The equality constraints of (3.10a) and its boundary condition (3.11a) may be relaxed as follows:

$$\begin{aligned} -\mathfrak{L}(\Psi) &\leq 0 \qquad x \in \Omega \\ \Psi(x) &\leq \psi(x) \qquad x \in \partial\Omega \end{aligned} \tag{4.1a}$$

and

$$\begin{aligned} -\mathfrak{L}(\Psi) &\geq 0 \qquad x \in \Omega \\ \Psi(x) &\geq \psi(x) \qquad x \in \partial\Omega. \end{aligned} \tag{4.1b}$$

Such a relaxation provides a point-wise bound to the solution Ψ^*, and this relaxation may be enforced via SOS programming. In particular, a solution to (4.1a), denoted as Ψ_l, is a lower bound on the solution Ψ^* over the entire problem domain, and a solution to (4.1b), denoted as Ψ_u, is an upper bound on the solution Ψ^* over the entire problem domain.

Theorem 4.1. *The following statements are true:*

1. *Given a smooth function Ψ_l that satisfies* (4.1a)*, then Ψ_l is a viscosity subsolution and $\Psi_l \leq \Psi^*$ for all $x \in \Omega$.*
2. *Given a smooth function Ψ_u that satisfies* (4.1b)*, then Ψ_u is a viscosity supersolution and $\Psi_u \geq \Psi^*$ for all $x \in \Omega$.*

Proof. By Definition 3.4, the solution Ψ_l is a viscosity subsolution, where F in (3.12) is given by (4.1a). Note that Ψ^* is both a viscosity subsolution and a viscosity supersolution, and $\Psi_l \leq \Psi^*$ on the boundary $\partial\Omega$. Thus, by the maximum principle [66, Thm. 3.3], $\Psi_l \leq \Psi^*$ for all $x \in \Omega$. The proof is identical for Ψ_u. □

Because the logarithmic transform (3.9) is monotonic, one can relate these bounds on the desirability function to bounds on the value function as follows:

Corollary 4.1. *If the solution to* (3.6) *is V^*, given solutions $V_u = -\lambda \log \Psi_l$ and $V_l = -\lambda \log \Psi_u$ from* (4.1)*, then $V_u \geq V^*$ and $V_l \leq V^*$.*

Proof. Recall that $V^* = -\lambda \log \Psi^*$. By monotonicity of the logarithmic function and Theorem 4.1, $V_u \geq V^*$ and $V_l \leq V^*$. □

The solutions to (4.1) do not satisfy (3.10a) exactly, but they provide point-wise bounds to the solution Ψ^*.

4.2.2 SOS Program

Given that relaxation (4.1) results in a point-wise upper and lower bound to the exact solution of (3.10a), we construct the following optimization problem that provides a suboptimal controller with bounded residual error:

$$\begin{aligned}
\min_{\Psi_l,\Psi_u} \quad & \varepsilon & & \\
s.t. \quad & -\mathfrak{L}(\Psi_l) \leq 0 & & x \in \Omega \\
& 0 \leq -\mathfrak{L}(\Psi_u) & & x \in \Omega \\
& \Psi_u - \Psi_l \leq \varepsilon & & x \in \Omega \\
& 0 \leq \Psi_l \leq \psi \leq \Psi_u & & x \in \partial\Omega \\
& \partial_{x^i}\Psi_l \leq 0 & & x^i \geq 0 \\
& \partial_{x^i}\Psi_l \geq 0 & & x^i \leq 0 \\
& \Psi_l(0) = 1, & &
\end{aligned} \tag{4.2}$$

where x^i is the i-th component of $x \in \Omega$. As mentioned in Section 4.2.1, the first two constraints result from the relaxations of the HJB equation, and the fourth constraint arises from the relaxation of the boundary conditions. The third constraint ensures that the difference between the upper bound and lower bound solution is bounded, and the last three constraints ensure that the solution yields a stabilizing controller, as will be made clear in Section 4.3. Note that in the optimization problem, Ψ_u and Ψ_l are polynomials, whereby the coefficients and the degree for both are optimization variables. The term ε is related to the error of the approximation.

As discussed in Section 3.6, a general optimization problem involving parameterized nonnegative polynomials is not necessarily tractable. In order to solve (4.2) using a polynomial-time algorithm, we restrict the polynomial inequalities such that they are SOS polynomials instead of nonnegative polynomials. We therefore apply

Proposition 3.1 to relax optimization problem (4.2) into

$$\begin{aligned}
\min_{\Psi_l,\Psi_u,S,T} \quad & \varepsilon && (4.3)\\
s.t. \quad & \mathfrak{L}(\Psi_l) - \mathcal{D}(\Omega, S_1) \in \mathcal{SOS}(x)\\
& -\mathfrak{L}(\Psi_u) - \mathcal{D}(\Omega, S_2) \in \mathcal{SOS}(x)\\
& \varepsilon - (\Psi_u - \Psi_l) - \mathcal{D}(\Omega, S_3) \in \mathcal{SOS}(x)\\
& \mathcal{B}(\Psi_l, \partial\Omega, T_1) \in \mathcal{SOS}(x)\\
& \mathcal{B}(\psi - \Psi_l, \partial\Omega, T_2) \in \mathcal{SOS}(x)\\
& \mathcal{B}(\Psi_u - \psi, \partial\Omega, T_3) \in \mathcal{SOS}(x)\\
& -\partial_{x^i}\Psi_l - \mathcal{D}(\Omega \cap \{x^i \geq 0\}, S_4) \in \mathcal{SOS}(x)\\
& \partial_{x^i}\Psi_l - \mathcal{D}(\Omega \cap \{-x^i \geq 0\}, S_5) \in \mathcal{SOS}(x)\\
& \Psi_l(0) = 1,
\end{aligned}$$

where $S = \{S_i\}_{i\in[5]_+}$, $S_i \subseteq \mathcal{SOS}(x)$ is defined as in Definition 4.1, $T = \{T_i\}_{i\in[3]_+}$, and $T_j \subseteq \mathbb{R}[x]$ is defined as in Definition 4.2. With a slight abuse of notation, $\mathcal{B}(\cdot) \in \mathcal{SOS}(x)$ implies that each polynomial in $\mathcal{B}(\cdot)$ is a SOS polynomial.

If the polynomial degrees are fixed, optimization problem (4.3) is convex and solvable using a semidefinite program via Theorem 3.3. The next section will discuss the systematic approach we used to solve the optimization problem. Henceforth, denote the solution to (4.3) as $(\Psi_u, \Psi_l, S, T, \varepsilon)$ (*i.e.,* the upper bound, the lower bound, and the sets of $\mathcal{SOS}$ polynomial certificates (S, T) yielding approximation error ε).

Remark. By Definition 3.4, the viscosity solution is a continuous function. Consequently, the solution Ψ^* is a continuous function defined on a bounded domain. Therefore, Ψ_u and Ψ_l can be made arbitrary close to Ψ^* by the Stone-Weierstrass Theorem [87] in (4.2). However, this guarantee is lost when Ψ_u and Ψ_l are restricted to be a SOS polynomials. The feasible set of the optimization problem (4.3) is therefore not necessarily non-empty for a given polynomial degree. One would not expect feasibility for all instances of (4.3) as this would imply there exists is a linear stabilizing controller for any given system.

4.2.3 Controller Synthesis

Let d be the maximum degree of Ψ_l, Ψ_u and polynomials in S and T, and denote $(\Psi_u^d, \Psi_l^d, S^d, T^d, \varepsilon^d)$ as a solution to (4.3) when the maximum polynomial degree is fixed at d. A hierarchy of SOS programs described in Section 3.6.2 is constructed with increasing polynomial degree. The hierarchy produces a sequence of (possibly empty) solutions $(\Psi_u^d, \Psi_l^d, S^d, T^d, \varepsilon^d)_{d\in I}$, where $I \subset \mathbb{Z}_+$. This sequence will be shown in the next section to improve, under the metric of the objective in (4.3).

In other words, if solutions exist for d and d' such that $d > d'$, then $\varepsilon^d \leq \varepsilon^{d'}$. Therefore, one could keep increasing the degree of polynomials in order to achieve tighter bounds on Ψ^*, and invariably, V^*. The use of such hierarchies has become commonplace in polynomial optimization [37], [68]. If at certain degree, $\varepsilon^d = 0$, the solution Ψ^* is found.

Once a satisfactory error is achieved or computational resources run out, the lower bound Ψ_l^d can be used to compute a suboptimal controller, where d is the maximum degree computed. Recall that $u^* = -R^{-1}G^T\nabla_x V^*$ and $V^* = -\lambda \log \Psi^*$. The suboptimal controller u^ε for a given degree d and error ε^d is computed as $u^{\varepsilon^d} = -R^{-1}G^T\nabla_x V_u^d$, where $V_u^d = -\lambda \log \Psi_l^d$. Even when ε^d is larger than a desired value, the solution Ψ_l^d still satisfies conditions in Definition 3.3 to yield a stabilizing suboptimal controller. Next section will analyze some properties of the solutions and the suboptimal controller.

4.3 Analysis for First Exit Problem

This section establishes several properties of the solutions to the optimization problem (4.3) that are useful for feedback control in the first exit problem. First we show that the solutions in the SOS program hierarchy are uniformly bounded relative to the exact solutions. We next prove that the relaxed solutions to the stochastic HJB equation are SCLFs, and the approximated solution leads to a stabilizing controller. Finally, we show that the costs of using the approximate solutions as controllers are bounded above by the approximated value functions.

4.3.1 Properties of Approximated Desirability Functions

First, the approximation error of Ψ_l or Ψ_u obtained from (4.3) is computed relative to the true desirability function Ψ^*.

Proposition 4.1. *Given a solution* $(\Psi_u^d, \Psi_l^d, S^d, T^d, \varepsilon^d)$ *to* (4.3) *for a given degree d, the approximation error of the desirability function is bounded as* $||\Psi^d - \Psi^*||_\infty \leq \varepsilon^d$, *where* Ψ^d *is either* Ψ_u^d *or* Ψ_l^d.

Proof. By Theorem 4.1, Ψ_l^d is the lower bound of Ψ^*, and Ψ_u^d is the upper bound of Ψ^*. So, $\varepsilon^d \geq \Psi_u^d - \Psi_l^d \geq 0$ and $\Psi_u^d \geq \Psi^* \geq \Psi_l^d$. Combining both inequalities, one has $\Psi_u^d - \Psi^* \leq \varepsilon^d$ and $\Psi^* - \Psi_l^d \leq \varepsilon^d$. Therefore, $||\Psi^d - \Psi^*||_\infty \leq \varepsilon^d$, where Ψ^d is either Ψ_u^d or Ψ_l^d. □

Proposition 4.2. *The hierarchy of SOS programs consisting of solutions to* (4.3) *with increasing polynomial degree produces a sequence of solutions* $(\Psi_u^d, \Psi_l^d, S^d, T^d, \varepsilon^d)$ *such that* $\varepsilon^{d+1} \leq \varepsilon^d$ *for all* d.

Proof. Polynomials of degree d form a subset of polynomials of degree $d+1$. Thus, at a higher polynomial degree $d+1$, a previous solution at a lower polynomial degree d is still a feasible solution when the coefficients for monomials with total degree $d+1$ is set to 0. Consequently, the optimal value ε^{d+1} cannot be larger than ε^d for all d. □

Thus, as the polynomial degree of the optimization problem is increased, the pointwise error ε is non-increasing. Therefore, one could keep increasing the degree of polynomials in order to achieve tighter bounds on Ψ^*, and invariably, V^*. However, ε is only non-increasing as the polynomial degree is increased, and a convergence of the bound ε to zero is not guaranteed because we restrict the approximating space to SOS. The possible lack of convergence to zero is the trade off for an efficient algorithm.

Although the *bound* on the pointwise error is non-increasing, the actual difference between Ψ and Ψ^* may increase between iterations. Figure 4.1 illustrates an example of this case. Although $\varepsilon^{d+1} \leq \varepsilon^d$, the actual error between Ψ_u^{d+1} and Ψ^* is larger than the actual error between Ψ_u^d and Ψ^*. The next corollary ensures that the increase in the actual error for Ψ_u^{d+1} is still bounded.

Corollary 4.2. *Suppose* $||\Psi^d - \Psi^*||_\infty \leq \varepsilon^d$ *and* $||\Psi^{d+1} - \Psi^*||_\infty = \gamma^{d+1}$. *Then,* $\gamma^{d+1} \leq \varepsilon^d$.

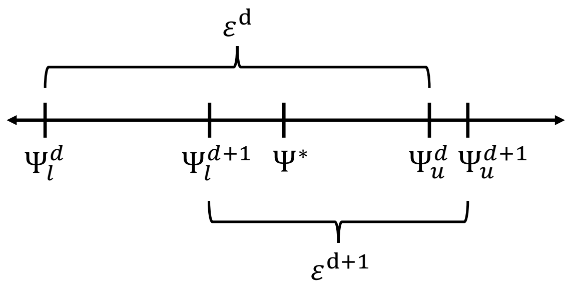


Figure 4.1: An example of increased error with non-increasing bound.

Proof. By Proposition 4.2, $\varepsilon^{d+1} \leq \varepsilon^d$. Because $\gamma^{d+1} \leq \varepsilon^{d+1}$, $\gamma^{d+1} \leq \varepsilon^d$ □

In other words, the approximation error of the desirability function for a SOS program using $d + 1$ polynomial degree cannot increase such that it is larger than ε^d in each step of the hierarchy of SOS programs, which is non-increasing.

4.3.2 Properties of Approximated Value Functions

Up to this point, the analysis has focused on properties of the desirability solution. We now investigate the implications of these results upon the value function, which is related to the desirability via the logarithmic transform (3.9). Henceforth, denote the solution to (3.6a) as $V^*(x_t) = \min_{u[t:T]} \mathbb{E}_{\omega_t}[J(x_t)] = -\lambda \log \Psi^*(x_t)$, the solution to (4.3) for a fixed degree d as $(\Psi_u, \Psi_l, S, T, \varepsilon)$, and the suboptimal value function computed from the solution of (4.3) as $V_u = -\lambda \log \Psi_l$. Only Ψ_l and V_u are considered henceforth, because Ψ_l, but not Ψ_u, gives an approximate value function that satisfies the properties of SCLF in Definition 3.3, a fact shown in the next section.

Theorem 4.2. *For all $x \in \Omega$, V_u is an upper bound of V^* such that*

$$0 \leq V_u - V^* \leq -\lambda \log \left(1 - \min\left\{1, \frac{\varepsilon}{\eta}\right\}\right),$$

where $\eta = e^{-\frac{\|V^*\|_\infty}{\lambda}}$.

Proof. By Corollary 4.1, $V_u \geq V^*$ and hence, $V_u - V^* \geq 0$. To prove the other inequality, by Proposition 4.1,

$$V_u - V^* = -\lambda \log \frac{\Psi_l}{\Psi^*} \leq -\lambda \log \frac{\Psi^* - \varepsilon}{\Psi^*} \leq -\lambda \log \left(1 - \frac{\varepsilon}{\eta}\right).$$

The last inequality holds because $\Psi^* \geq e^{-\frac{\|V^*\|_\infty}{\lambda}}$ by definition in (3.9). Since Ψ_l is the lower bound of Ψ^*, the right hand side of the first equality is always a positive number. Therefore, V_u is a point-wise upper bound of V^*. □

Corollary 4.3. *Let $V_u^d = -\lambda \log \Psi_l^d$ and $V_u^{d+1} = -\lambda \log \Psi_l^{d+1}$. If $\Psi_u^d - \Psi^* \leq \varepsilon^d$ and $V_u^{d+1} - V^* = \gamma^{d+1}$, then $\gamma^{d+1} \leq -\lambda \log \left(1 - \min\left\{1, \frac{\varepsilon^d}{\eta}\right\}\right)$.*

Proof. This result is given by Corollary 4.2 and Theorem 4.2. □

At this point, we have shown that the lower bound of the desirability function yields an upper bound of the suboptimal cost. More importantly, the upper bound of the suboptimal cost is not increasing as the degree of polynomial increases.

4.3.3 Approximate HJB Solutions are SCLFs

This section shows that the approximate value function derived from the approximation, Ψ_l, is a SCLF.

Theorem 4.3. *V_u is a stochastic control Lyapunov function according to Definition 3.3.*

Proof. The constraint $\Psi_l(0) = 1$ in (4.3) ensures that $V_u(0) = -\lambda \log \Psi_l(0) = 0$. Notice that all terms in $J(x, u)$ from (3.3) are positive definite, resulting in V^* being a positive definite function. In addition, by Corollary 4.1, $V^u \geq V^*$. Thus, V^u is also a positive definite function. The second and third to last constraints in (4.3) ensures that Ψ_l is nonincreasing away from the origin. Therefore, V_u is nondecreasing away form the origin satisfying $\mu(|x|) \leq V_u(x) \leq \mu'(|x|)$ for some $\mu, \mu' \in \mathcal{K}$.

Next, we show that there exists a u such that $L(V_u) \leq 0$. Following (3.7), let

$$u^\varepsilon = -R^{-1}G^T\nabla_x V_u, \tag{4.4}$$

the control law corresponding to V_u. Notice that from the definition of V_u, $\nabla_x V_u = -\frac{\lambda}{\Psi_l}\nabla_x\Psi_l$ and $\nabla_{xx}V_u = \frac{\lambda}{\Psi_l^2}(\nabla_x\Psi_l)(\nabla_x\Psi_l)^T - \frac{\lambda}{\Psi_l}\nabla_{xx}\Psi_l$. So, $u^\varepsilon = \frac{\lambda}{\Psi_l}R^{-1}G^T\nabla_x\Psi_l$. Then, from (3.2),

$$\begin{aligned} L(V_u) = &-\frac{\lambda}{\Psi_l}(\nabla_x\Psi_l)^T(f + \frac{\lambda}{\Psi_l}GR^{-1}G^T\nabla_x\Psi_l) \\ &\qquad + \frac{1}{2}Tr\left(\left(\frac{\lambda}{\Psi_l^2}(\nabla_x\Psi_l)(\nabla_x\Psi_l)^T - \frac{\lambda}{\Psi_l}\nabla_{xx}\Psi_l\right)B\Sigma_\varepsilon B\right), \end{aligned}$$

where $\partial_t V_u = 0$ because V_u is not a function of time. Applying the assumption in (3.8) and simplifying yields

$$L(V_u) = -\frac{\lambda}{\Psi_l}(\nabla_x\Psi_l)^T f - \frac{\lambda}{2\Psi_l^2}(\nabla_x\Psi_l)^T\Sigma_t\nabla_x\Psi_l - \frac{\lambda}{2\Psi_l}Tr\left((\nabla_{xx}\Psi_l)\,\Sigma_t\right).$$

From the first constraint in (4.3),

$$\begin{aligned} \frac{1}{\lambda}q\Psi_l - f^T(\nabla_x\Psi_l) - \frac{1}{2}Tr\left((\nabla_{xx}\Psi_l)\,\Sigma_t\right) \leq 0 \implies \\ -\frac{\lambda}{\Psi_l}(\nabla_x\Psi_l)^T f \leq -q + \frac{\lambda}{2\Psi_l}Tr\left((\nabla_{xx}\Psi_l)\,\Sigma_t\right). \end{aligned}$$

Substituting this inequality into $L(V_u)$ and simplifying yields

$$L(V_u) \leq -q - \frac{\lambda}{2\Psi_l^2}(\nabla_x\Psi_l)^T\Sigma_t\nabla_x\Psi_l \leq 0 \tag{4.5}$$

because $q \geq 0$, $\lambda > 0$ and Σ_t is positive semidefinite by definition. Since V_u satisfies Definition 3.3, V_u is a SCLF. □

Corollary 4.4. *The suboptimal controller* $u^{\varepsilon} = -R^{-1}G^T\nabla_x V_u$ *is stabilizing in probability within the domain* Ω.

Proof. This corollary is a direct consequence of the constructive proof of Theorem 4.3 and Theorem 3.1. □

Corollary 4.5. *If* Σ_t *is a positive definite matrix, the suboptimal controller* $u^{\varepsilon} = -R^{-1}G^T\nabla_x V_u$ *is asymptotically stabilizing in probability within the domain* Ω.

Proof. This corollary is a direct consequence of the constructive proof of Theorem 4.3 and Theorem 3.2. In (4.5), $L(V_u) < 0$ for $x \in \Omega\backslash\{0\}$ if Σ_t is positive definite. Recall that q is positive definite in the problem formulation. □

4.3.4 Bound on the Total Trajectory Cost

We conclude this section by showing that the expected total trajectory cost incurred by the system while operating under the suboptimal controller of (4.4) can be bounded as follows.

Theorem 4.4. *Given the control law* $u^{\varepsilon} = -R^{-1}G^T\nabla_x V_u$,

$$J_u \le V_u \le V^* - \lambda \log\left(1 - \min\left\{1, \frac{\varepsilon}{\eta}\right\}\right), \tag{4.6}$$

where $J_u = \mathbb{E}_{\omega_t}[\phi_T(x_T) + \int_0^T q(x_t) + \frac{1}{2}u_t^T R u_t \; dt]$, *the expected cost of the system when using the control law,* u^{ε}.

Proof. By Itô's formula,

$$dV_u(x_t) = L(V_u)(x_t)dt + \nabla_x V_u(x_t)B(x_t)d\omega_t,$$

where $L(V)$ is defined in (3.2). Then,

$$V_u(x_t) = V_u(x_0, 0) + \int_0^t L(V_u)(x_s)ds + \int_0^t \nabla_x V_u(x_s)B(x_s)d\omega_s. \tag{4.7}$$

Given that V_u is derived from polynomial function Ψ_l, the integrals are well defined, and we can take the expectation of (4.7) to get

$$\mathbb{E}[V_u(x_t)] = V_u(x_0, 0) + \mathbb{E}\left[\int_0^t L(V_u)(x_s)ds\right],$$

whereby the last term of (4.7) drops out because the noise is assumed to have zero mean. The expectations of the other terms return the same terms because they are deterministic. From (4.5),

$$\begin{aligned} L(V_u) &\leq -q - \frac{\lambda}{2\Psi_l^2}(\nabla_x \Psi_l)^T \Sigma_t \nabla_x \Psi_l \\ &= -q - \frac{1}{2}\left(\nabla_x V_u\right)^T GR^{-1}G^T \left(\nabla_x V_u\right) \\ &= -q - \frac{1}{2}(u^\varepsilon)^T R u^\varepsilon, \end{aligned}$$

where the first equality is given by the logarithmic transformation and the second equality is given by the control law $u^\varepsilon = -R^{-1}G^T \nabla_x V_u$. Therefore,

$$\begin{aligned} \mathbb{E}_{\omega_t}[V_u(x_T)] &= V_u(x_0) + \mathbb{E}_{\omega_t}\left[\int_0^T L(V_u)(x_s) ds\right] \\ &\leq V_u(x_0) - \mathbb{E}_{\omega_t}\left[\int_0^T q(x_s) + \frac{1}{2}(u_s^\varepsilon)^T R u_s^\varepsilon ds\right] \\ &= V_u(x_0) - J(x_0, u^\varepsilon) + \mathbb{E}_{\omega_t}[\phi(x_T)], \end{aligned}$$

where the last equality is given by (3.3). Consequently,

$$V_u(x_0) - J(x_0, u^\varepsilon) \geq \mathbb{E}_{\omega_t}[V_u(x_T) - \phi(x_T)].$$

By definition, $V_u(x_T) \geq \phi(x_T)$ for all $x_T \in \partial\Omega$. Thus, $\mathbb{E}_{\omega_t}[V_u(x_T) - \phi(x_T)] \geq 0$. Consequently, $V_u(x_0) - J(x_0, u^\varepsilon) \geq 0$, and $V_u(x_0) \geq J(x_0, u^\varepsilon)$. Theorem 4.2 gives the second inequality in the theorem. □

4.4 Controller Synthesis for Finite Horizon Problem

This section discusses the controller synthesis procedure for the finite horizon problem. The general procedure is the same as Section 4.2. Therefore, this section will outline the prodecure and results with minimal descriptions except when the approach is specific to the finite horizon problem. For more high level descriptions and intuitions, refer to Section 4.2.

The main difference between Section 4.2 and this section is that the controller is not necessarily stabilizing anymore. The controller obtained will minimize the cost functionals, but there is no guarantee on the system performance beyond the pre-specified time horizon.

4.4.1 Relaxation of the Finite Horizon HJB Equation

The equality constraints of (3.10b) and its boundary condition (3.11b) may be relaxed as follows:

$$\begin{aligned} -\partial_t \Psi - \mathfrak{L}(\Psi) &\leq 0, \ (x,t) \in \mathcal{O} \\ \Psi(x,t) &\leq \psi(x,t), \ (x,t) \in \partial\mathcal{O} \end{aligned} \tag{4.8a}$$

and

$$\begin{aligned} -\partial_t \Psi - \mathfrak{L}(\Psi) &\geq 0, \ (x,t) \in \mathcal{O} \\ \Psi(x,t) &\geq \psi(x,t), \ (x,t) \in \partial\mathcal{O}. \end{aligned} \tag{4.8b}$$

This relaxation provides a point-wise bound to the solution Ψ^*. In particular, a solution to (4.8a), denoted as Ψ_l, is a lower bound on the solution Ψ^* over the entire problem domain, and a solution to (4.8b), denoted as Ψ_u, is a upper bound on the solution Ψ^* over the entire problem domain.

Theorem 4.5. *The following statements are true:*

1. *Given a smooth function Ψ_l that satisfies* (4.8a)*, then Ψ_l is a viscosity subsolution and $\Psi_l \leq \Psi^*$ for all $(x,t) \in \mathcal{O}$.*
2. *Given a smooth function Ψ_u that satisfies* (4.8b)*, then Ψ_u is a viscosity supersolution and $\Psi_u \geq \Psi^*$ for all $(x,t) \in \mathcal{O}$.*

Proof. By Definition 3.5, the solution Ψ_l is a viscosity subsolution, where F in (3.13) is given by (4.8a). The sign of $\partial_t \Psi$ in (4.8a) is different from $\partial_t u$ in Definition 3.5. However, (3.10b) is solved backward in time given a terminal time "initial" condition, and thus the direction of time must be reversed relative to the time of the system's evolution. Furthermore, note that Ψ^* is both a viscosity subsolution and a viscosity supersolution, and $\Psi_l \leq \Psi^*$ on the boundary $\partial\mathcal{O}$. Therefore, by the maximum principle [66, Thm. 8.2], $\Psi_l \leq \Psi^*$ for all $(x,t) \in \mathcal{O}$. The proof is identical for Ψ_u. □

Because the logarithmic transform (3.9) is monotonic, one can relate these bounds on the desirability function to bounds on the value function as follows

Corollary 4.6. *If the solution to* (3.6b) *is V^*, given solutions $V_u = -\lambda \log \Psi_l$ and $V_l = -\lambda \log \Psi_u$ from* (4.8)*, then $V_u \geq V^*$ and $V_l \leq V^*$.*

Proof. Recall that $V^* = -\lambda \log \Psi^*$. By monotonicity of the logarithmic function and Theorem 4.5, $V_u \geq V^*$ and $V_l \leq V^*$. $\square$

The solutions to (4.8) do not satisfy (3.10b) exactly, but they provide point-wise bounds to the solution Ψ^*.

4.4.2 SOS Program

Given that relaxation (4.8) results in a point-wise upper and lower bound to the exact solution of (3.10b), the following optimization problem is constructed that provides a suboptimal controller with bounded residual error:

$$\begin{aligned} \min_{\Psi_l, \Psi_u} \quad & \varepsilon \\ s.t. \quad & -\partial_t \Psi_l - \mathfrak{L}(\Psi_l) \leq 0 && (x,t) \in \mathcal{O} \\ & 0 \leq -\partial_t \Psi_u - \mathfrak{L}(\Psi_u) && (x,t) \in \mathcal{O} \\ & \Psi_u - \Psi_l \leq \varepsilon && (x,t) \in \mathcal{O} \\ & 0 \leq \Psi_l \leq \psi \leq \Psi_u && (x,t) \in \partial\mathcal{O}, \end{aligned} \tag{4.9}$$

where x^i is the i-th component of $x \in \Omega$. The first two constraints result from the relaxations of the HJB equation, and the fourth constraint arises from the relaxation of the boundary conditions. The third constraint ensures that the difference between the upper bound and lower bound solution is bounded. Note that in the optimization problem, Ψ_u and Ψ_l are polynomials whose coefficients and degree are optimization variables. The term ε is related to the error of the approximation.

To solve (4.9) using a polynomial-time algorithm, restrict the polynomial inequalities such that they are SOS polynomials instead of nonnegative polynomials. The optimization problem (4.9) is then relaxed into

$$\begin{aligned} \min_{\Psi_l, \Psi_u, S, T} \quad & \varepsilon \\ s.t. \quad & -\partial_t \Psi_l - \mathfrak{L}(\Psi_l) - \mathcal{D}(\mathcal{O}, S_1) \in \mathcal{SOS}(x,t) \\ & \partial_t \Psi_l + \mathfrak{L}(\Psi_l) - \mathcal{D}(\mathcal{O}, S_2) \in \mathcal{SOS}(x,t) \\ & \varepsilon - (\Psi_u - \Psi_l) - \mathcal{D}(\mathcal{O}, S_3) \in \mathcal{SOS}(x,t) \\ & \mathcal{B}(\Psi_l, \partial\mathcal{O}, T_1) \in \mathcal{SOS}(x,t) \\ & \mathcal{B}(\psi - \Psi_l, \partial\mathcal{O}, T_2) \in \mathcal{SOS}(x,t) \\ & \mathcal{B}(\Psi_u - \psi, \partial\mathcal{O}, T_3) \in \mathcal{SOS}(x,t), \end{aligned} \tag{4.10}$$

where $S = \{S_i\}_{i\in[3]_+}$, $S_i \subseteq \mathcal{SOS}(x)$ is defined as in 4.1, $T = \{T_i\}_{i\in[3]_+}$, and $T_j \subseteq \mathbb{R}[x]$ is defined as in 4.2. With a slight abuse of notation, $\mathcal{B}(\cdot) \in \mathcal{SOS}(x)$ implies that each polynomial in $\mathcal{B}(\cdot)$ is a SOS polynomial.

If the polynomial degrees are fixed, optimization problem (4.10) is convex and solvable using a semidefinite program via Theorem 3.3. The suboptimal controller is then synthesized by forming a hierarchy of SOS programs as described in Section 4.2. Henceforth, denote the solution to (4.10) as $(\Psi_u, \Psi_l, S, T, \varepsilon)$.

4.5 Analysis for Finite Horizon Problem

This section summarizes results for finite horizon problem that are analogous to those in Section 4.3. We show that the solutions in the SOS program hierarchy are uniformly bounded relative to the exact solutions, and the costs of using the approximate solutions as controllers are bounded above by the approximated value functions.

4.5.1 Properties of Approximated Desirability Functions

Proposition 4.3. *Given a solution* $(\Psi_u^d, \Psi_l^d, S^d, T^d, \varepsilon^d)$ *to* (4.10) *for a given degree d, the approximation error of the finite horizon desirability function is bounded as* $||\Psi^d - \Psi^*||_\infty \leq \varepsilon^d$*, where* Ψ^d *is either* Ψ_u^d *or* Ψ_l^d*.*

Proof. By Theorem 4.5, Ψ_l^d is the lower bound of Ψ^*, and Ψ_u^d is the upper bound of Ψ^*. So, $\varepsilon^d \geq \Psi_u^d - \Psi_l^d \geq 0$ and $\Psi_u^d \geq \Psi^* \geq \Psi_l^d$. Combining both inequalities, one has $\Psi_u^d - \Psi^* \leq \varepsilon^d$ and $\Psi^* - \Psi_l^d \leq \varepsilon^d$. Therefore, $||\Psi^d - \Psi^*||_\infty \leq \varepsilon^d$, where Ψ^d is either Ψ_u^d or Ψ_l^d. □

Proposition 4.4. *The hierarchy of SOS programs consisting of solutions to* (4.10) *with increasing polynomial degree produces a sequence of solutions* $(\Psi_u^d, \Psi_l^d, S^d, T^d, \varepsilon^d)$ *such that* $\varepsilon^{d+1} \leq \varepsilon^d$ *for all d.*

Proof. Polynomials of degree d form a subset of polynomials of degree $d+1$. Thus, at a higher polynomial degree $d+1$, a previous solution at a lower polynomial degree d is still a feasible solution when the coefficients for monomials with total degree $d+1$ is set to 0. Consequently, the optimal value ε^{d+1} cannot be larger than ε^d for all d. □

Although the *bound* on the pointwise error is non-increasing, the actual difference between Ψ and Ψ^* may increase between iterations.

Corollary 4.7. *Suppose* $||\Psi^d - \Psi^*||_\infty \leq \varepsilon^d$ *and* $||\Psi^{d+1} - \Psi^*||_\infty = \gamma^{d+1}$*. Then,* $\gamma^{d+1} \leq \varepsilon^d$*.*

Proof. By Proposition 4.4, $\varepsilon^{d+1} \leq \varepsilon^d$. Because $\gamma^{d+1} \leq \varepsilon^{d+1}$, $\gamma^{d+1} \leq \varepsilon^d$ □

4.5.2 Properties of Approximated Value Functions

Theorem 4.6. *For all* $(x, t) \in \bar{\mathcal{O}}$*,* V_u *is an upper bound of* V^* *such that*

$$0 \leq V_u - V^* \leq -\lambda \log\left(1 - \min\left\{1, \frac{\varepsilon}{\eta}\right\}\right) \tag{4.11}$$

where $\eta = e^{-\frac{||V^*||_\infty}{\lambda}}$*.*

Proof. By Corollary 4.6, $V_u \geq V^*$ and hence, $V_u - V^* \geq 0$. To prove the other inequality, by Proposition 4.1,

$$V_u - V^* = -\lambda \log \frac{\Psi_l}{\Psi^*} \leq -\lambda \log \frac{\Psi^* - \varepsilon}{\Psi^*} \leq -\lambda \log\left(1 - \frac{\varepsilon}{\eta}\right),$$

The last inequality holds because $\Psi^* \geq e^{-\frac{||V^*||_\infty}{\lambda}}$ by definition in (3.9). Since Ψ_l is the lower bound of Ψ^*, the right hand side of the first equality is always a positive number. Therefore, V_u is a point-wise upper bound of V^*. □

Corollary 4.8. *Let* $V_u^d = -\lambda \log \Psi_l^d$ *and* $V_u^{d+1} = -\lambda \log \Psi_l^{d+1}$*. If* $\Psi_u^d - \Psi^* \leq \varepsilon^d$ *and* $V_u^{d+1} - V^* = \gamma^{d+1}$*, then* $\gamma^{d+1} \leq -\lambda \log\left(1 - \min\left\{1, \frac{\varepsilon^d}{\eta}\right\}\right)$*.*

Proof. This result is given by Corollary 4.7 and Theorem 4.6. □

4.5.3 Bound on the Total Trajectory Cost

Theorem 4.7. *Given the control law* $u^\varepsilon = -R^{-1}G^T\nabla_x V_u$*,*

$$J_u \leq V_u \leq V^* - \lambda \log\left(1 - \min\left\{1, \frac{\varepsilon}{\eta}\right\}\right), \tag{4.12}$$

where $J_u = \mathbb{E}_{\omega_t}[\phi_T(x_T) + \int_0^T q(x_t) + \frac{1}{2}u_t^T R u_t \; dt]$*, the expected cost of the system when using the control law,* u^ε*.*

Proof. By Itô's formula,

$$dV_u(x_t) = L(V_u)(x_t)dt + \nabla_x V_u(x_t)B(x_t)d\omega_t,$$

where $L(V)$ is defined in (3.2). Then,

$$V_u(x_t) = V_u(x_0, 0) + \int_0^t L(V_u)(x_s)ds + \int_0^t \nabla_x V_u(x_s)B(x_s)d\omega_s. \tag{4.13}$$

Given that V_u is derived from polynomial function Ψ_l, the integrals are well defined, and we can take the expectation of (4.13) to get

$$\mathbb{E}[V_u(x_t)] = V_u(x_0, 0) + \mathbb{E}\left[\int_0^t L(V_u)(x_s)ds\right],$$

whereby the last term of (4.13) drops out because the noise is assumed to have zero mean. The expectations of the other terms return the same terms because they are deterministic.

Notice that from the definition of V_u, $\partial_t V_u = -\frac{\lambda}{\Psi_l}\partial_t\Psi_l$, $\nabla_x V_u = -\frac{\lambda}{\Psi_l}\nabla_x\Psi_l$, and $\nabla_{xx}V_u = \frac{\lambda}{\Psi_l^2}(\nabla_x\Psi_l)(\nabla_x\Psi_l)^T - \frac{\lambda}{\Psi_l}\nabla_{xx}\Psi_l$. So, $u^\varepsilon = \frac{\lambda}{\Psi_l}R^{-1}G^T\nabla_x\Psi_l$. Thus, from (3.2),

$$\begin{aligned} L(V_u) = -\frac{\lambda}{\Psi_l}\partial_t\Psi_l - \frac{\lambda}{\Psi_l}(\nabla_x\Psi_l)^T(f + \frac{\lambda}{\Psi_l}GR^{-1}G^T\nabla_x\Psi_l) \\ + \frac{1}{2}Tr\left(\left(\frac{\lambda}{\Psi_l^2}(\nabla_x\Psi_l)(\nabla_x\Psi_l)^T - \frac{\lambda}{\Psi_l}\nabla_{xx}\Psi_l\right)B\Sigma_\varepsilon B\right). \end{aligned}$$

Applying the assumption in (3.8) and simplifying yields

$$L(V_u) = -\frac{\lambda}{\Psi_l}\partial_t\Psi_l - \frac{\lambda}{\Psi_l}(\nabla_x\Psi_l)^T f - \frac{\lambda}{2\Psi_l^2}(\nabla_x\Psi_l)^T\Sigma_t\nabla_x\Psi_l - \frac{\lambda}{2\Psi_l}Tr\left((\nabla_{xx}\Psi_l)\Sigma_t\right).$$

From the first constraint in (4.10),

$$\begin{aligned} -\partial_t\Psi_l + \frac{1}{\lambda}q\Psi_l - f^T(\nabla_x\Psi_l) - \frac{1}{2}Tr\left((\nabla_{xx}\Psi_l)\Sigma_t\right) \le 0 \implies \\ -\frac{\lambda}{\Psi_l}\left(\partial_t\Psi_l + (\nabla_x\Psi_l)^T f\right) \le -q + \frac{\lambda}{2\Psi_l}Tr\left((\nabla_{xx}\Psi_l)\Sigma_t\right). \end{aligned}$$

Substituting this inequality into $L(V_u)$ and simplifying yields

$$\begin{aligned} L(V_u) &\le -q - \frac{\lambda}{2\Psi_l^2}(\nabla_x\Psi_l)^T\Sigma_t\nabla_x\Psi_l \\ &= -q - \frac{1}{2}\left(\nabla_x V_u\right)^T GR^{-1}G^T\left(\nabla_x V_u\right) \\ &= -q - \frac{1}{2}(u^\varepsilon)^T Ru^\varepsilon, \end{aligned}$$

where the first equality is given by the logarithmic transformation and the second equality is given by the control law $u^{\varepsilon} = -R^{-1}G^T \nabla_x V_u$. Therefore,

$$\begin{aligned}
\mathbb{E}_{\omega_t}[V_u(x_T, T)] &= V_u(x_0, 0) + \mathbb{E}_{\omega_t}\left[\int_0^T L(V_u)(x_s, s)ds\right] \\
&\leq V_u(x_0, 0) - \mathbb{E}_{\omega_t}\left[\int_0^T q(x_s) + \frac{1}{2}(u_s^{\varepsilon})^T R u_s^{\varepsilon} ds\right] \\
&= V_u(x_0, 0) - J(x_0, u^{\varepsilon}) + \mathbb{E}_{\omega_t}[\phi(x_T)],
\end{aligned}$$

where the last equality is given by (3.4). Therefore,

$$V_u(x_0, 0) - J(x_0, u^{\varepsilon}) \geq \mathbb{E}_{\omega_t}[V_u(x_T, T) - \phi(x_T)].$$

By definition, $V_u(x, T) \geq \phi(x)$ for all $(x, T) \in \Omega \times \{T\}$. Thus, $\mathbb{E}_{\omega_t}[V_u(x_T, T) - \phi(x_T)] \geq 0$. Consequently, $V_u(x_0, 0) - J(x_0, u^{\varepsilon}) \geq 0$, and $V_u(x_0, 0) \geq J(x_0, u^{\varepsilon})$. Theorem 4.6 gives the second inequality in the theorem. □

4.6 Extension to Deterministic Systems

This section discusses the extension of the SOS-based controller synthesis technique to compute suboptimal controller for deterministic nonlinear systems. The approach presented in this chapter would appear up to this point to be limited to systems that are linearly solvable, *i.e.,* those that satisfy condition (3.8). However, the proposed methods may be extended to a system, which does not satisfy these conditions by approximating the system with one that is linearly solvable. One example is to introduce stochastic forcing into an otherwise deterministic system.

We first construct a comparison theorem between HJB solutions to systems that share the same general dynamics, but with differing noise covariance. This comparison allows for the approximated value function of one system to bound the value function for another, providing pointwise bounds, and indeed SCLFs, for those that do not satisfy (3.8).

Proposition 4.5. *Suppose V^{a^*} is the solution to the HJB equation* (3.6) *with noise covariances Σ_a, and V^b is a supersolution to* (3.6) *with identical parameters except the noise covariance Σ^b, where $\Sigma_b - \Sigma_a \geq 0$, then $V^b \geq V^{a^*}$ for all $x \in \Omega$.*

Proof. From [66, Def. 2.2], V is a viscosity supersolution to the HJB equation (3.6) with noise covariance Σ if it satisfies

$$0 \leq -q - (\nabla_x V)^T f + \frac{1}{2}(\nabla_x V)^T G R^{-1} G^T (\nabla_x V) - \frac{1}{2} Tr\left((\nabla_{xx} V) B \Sigma B^T\right). \quad (4.14)$$

Since $\Sigma_b - \Sigma_a \geq 0$ the following trace inequality holds:

$$Tr\left((\nabla_{xx} V^a) B \Sigma_b B^T \right) \geq Tr\left((\nabla_{xx} V^a) B \Sigma_a B^T \right) .$$

Therefore, we have the inequality

$$\begin{aligned} 0 &\leq -q - \left(\nabla_x V^b\right)^T f + \frac{1}{2}\left(\nabla_x V^b\right)^T GR^{-1}G^T \left(\nabla_x V^b\right) - \frac{1}{2} Tr\left(\left(\nabla_{xx} V^b\right) B\Sigma_b B^T\right) \\ &\leq -q - \left(\nabla_x V^b\right)^T f + \frac{1}{2}\left(\nabla_x V^b\right)^T GR^{-1}G^T \left(\nabla_x V^b\right) - \frac{1}{2} Tr\left(\left(\nabla_{xx} V^b\right) B\Sigma_a B^T\right), \end{aligned}$$

which implies that V^b is in fact a viscosity supersolution to the system with noise covariance Σ^a (*i.e.,* V^b satisfies (4.14) for Σ^a). As V^b is a supersolution to the system with parameter Σ^a, then $V^b \geq V^{a^*}$. □

A particular class of such approximations arises from a deterministic HJB solution, which is not linearly solvable, but is approximated by one that is linearly solvable. Consider a deterministic system of the form

$$dx(t) = (f(x(t)) + G(x(t))u(t))\, dt \tag{4.15}$$

with cost function

$$J(x,u) = \phi(x(T)) + \int_0^T q(x(t)) + \frac{1}{2} u(t) R u(t)\, dt, \tag{4.16}$$

where $\phi, q, R, f, G,$ and the state and input domains are defined as in the stochastic problem in Section 4.1. Then, the HJB equation for the first exit problem is given by

$$0 = q + (\nabla_x V)^T f - \frac{1}{2} (\nabla_x V)^T GR^{-1}G^T (\nabla_x V) \tag{4.17}$$

and the optimal control is given by $u^* = -R^{-1}G^T \nabla_x V$. In general, (4.17) is not a linear PDE.

Corollary 4.9. *Let V^* be the value function that solves* (4.17)*, and V^u be the upper bound solution obtained from* (4.3)*, where all parameters are the same as* (4.17) *and Σ_t is not zero. Then, V^u is an upper bound for V^* over the domain (i.e., $V^* \leq V^u$).*

Proof. A simple application of Proposition 4.5, where Σ_a takes the form of a zero matrix, gives $V^* \leq V^u$. □

Interestingly, using the solution from (4.3) and the transformation $V_u = -\lambda \log \Psi_l$, the suboptimal controller $u^\varepsilon = -R^{-1}G^T \nabla_x V_u$ is a stabilizing controller for the deterministic system (4.15) if a simple condition is satisfied. This fact is shown using the Lyapunov theorem for deterministic systems introduced next [41].

Definition 4.3. Given the system (4.15) and cost function (4.16), a control Lyapunov function (CLF) is a proper positive definite function $\mathcal{V} \in C^1$ on a compact domain $\Omega \cup \{0\}$ such that

$$\begin{aligned} \mathcal{V}(0) = 0,\ \mathcal{V}(x) \geq \mu(|x|) \quad \forall\, x \in \Omega \backslash \{0\} \\ \exists\, u(x)\ s.t.\ (\nabla_x \mathcal{V})^T (f + Gu) \leq 0 \quad \forall\, x \in \Omega \backslash \{0\}, \end{aligned} \tag{4.18}$$

where $\mu \in \mathcal{K}$.

Theorem 4.8 ([41] Thm. 2.5)**.** *Given a system* (4.15) *and cost function* (4.16)*, if there exists a CLF V and a control u*(x) *satisfying Definition 4.3, then the controlled system is stable, and u is a stabilizing controller. Furthermore, if* $(\nabla_x V)^T (f + Gu) < 0$ *for all* $x \in \Omega \backslash \{0\}$*, the controlled system is asymptotically stable, and u is an asymptotically stabilizing controller.*

Verifying that the controller $u^\varepsilon = -R^{-1} G^T \nabla_x V_u$ is in fact stabilizing and that V_u is a CLF may be seen as follows.

Corollary 4.10. *Given the controller* $u^\varepsilon = -R^{-1} G^T \nabla_x V_u$*, if*

$$Tr\left((\nabla_{xx} V_u)\, \Sigma_t\right) \geq 0 \quad \forall\, x \in \Omega \backslash \{0\},$$

then u^ε *is a stabilizing controller for* (4.15)*. If*

$$Tr\left((\nabla_{xx} V_u)\, \Sigma_t\right) > 0 \quad \forall\, x \in \Omega \backslash \{0\},$$

then u^ε *is an asymptotically stabilizing controller for* (4.15)*.*

Proof. Recall that from the proof of Theorem 4.3, all conditions in Definition 4.3 are satisfied by V_u except (4.18). To show that V_u satisfies (4.18), rearrange (3.6) to yield the following:

$$\begin{aligned} (\nabla_x V_u)^T (f + Gu^\varepsilon) &= (\nabla_x V_u)^T f - (\nabla_x V_u)^T G R^{-1} G^T (\nabla_x V_u) \\ &\leq -q - \frac{1}{2} (\nabla_x V_u)^T G R^{-1} G^T (\nabla_x V_u) - \frac{1}{2} Tr\left((\nabla_{xx} V_u)\, \Sigma_t\right), \end{aligned}$$

where $\Sigma_t = B \Sigma_\varepsilon B^T$. Recall that q and R are positive definite. If $Tr\left((\nabla_{xx} V_u)\, \Sigma_t\right) \geq 0$ for all $x \in \Omega \backslash \{0\}$, then $(\nabla_x V_u)^T (f + Gu^\varepsilon) \leq 0$ implying that V^u is a CLF and u^ε is a stabilizing controller by Theorem 4.8. Furthermore, if $Tr\left((\nabla_{xx} V_u)\, \Sigma_t\right) > 0$ for all $x \in \Omega \backslash \{0\}$, u^ε is an asymptotically stabilizing controller. □

In the deterministic case, Σ_t is free variable that can be chosen to be small according to the equality (3.8). Therefore, (3.8) is no longer a constraint or an assumption, but it serves as a design principle for obtaining a CLF for system (4.15). Furthermore, given a Σ_t, the trace condition in Corollary 4.10 is easily enforced in (4.3) by adding one extra constraint in the optimization problem. Thus, the optimization problem (4.3) can also produce a CLF for the corresponding deterministic system, with analytical results from the Section 4.3, including a priori trajectory suboptimality bounds (Theorem 4.4), inherited as well.

4.7 Robust Nonlinear Optimal Controller Synthesis

Apart from approximating deterministic systems, the proposed technique may be extended to incorporate uncertainty in the problem data. Assume there exists unknown coefficients $a \in \mathcal{H}$ in $f(x), G(x), B(x)$, where $\mathcal{H} \subset \mathbb{R}^k$, $\mathcal{H} = \{a \mid g_i(a) \geq 0, g_i(a) \in \mathcal{R}(x), i \in [r]_+\}$ is a basic closed semialgebraic set describing the uncertainty set of a. The problem data is then defined by the expressions $f(x,a), G(x,a), B(x,a)$ for $x \in \Omega$, and $a \in \mathcal{H}$. In this case, the uncertain parameters may be considered as additional domain variables, defined over their own compact space.

Uncertainty of this form may be incorporated naturally into the optimization problem (4.3). Define the monomial set $\mathcal{X} = \{a^\alpha x^\beta\}_{\alpha \in [r_a]_+, \beta \in [r_x]_+}$. The optimization variables corresponding to the polynomials in S and T in (4.3) are then constructed out of $\mathcal{X}$ as

$$p(x,a) = \sum_{\alpha=1}^{r_a} \sum_{\beta=1}^{r_x} c_{\alpha,\beta} a^\alpha x^\beta.$$

Note that Ψ_u and Ψ_l are not themselves functions of a because they are upper and lower bounds for solutions of all possible a. The uncertainty set $\mathcal{H}$ is incorporated by defining a compact domain $\mathcal{M} = \Omega \times \mathcal{H}$ that takes the product of the original problem domain and the uncertainty set. The resulting optimization problem is therefore

$$\begin{aligned}
\min_{\Psi_l,\Psi_u,S,T} \quad & \varepsilon \\
s.t. \quad & -\frac{1}{\lambda}q\Psi_l + \mathcal{L}(\Psi_l, a) - \mathcal{D}(\mathcal{M}, S_1) \in \mathcal{SOS}(x,a) \\
& \frac{1}{\lambda}q\Psi_u - \mathcal{L}(\Psi_u, a) - \mathcal{D}(\mathcal{M}, S_2) \in \mathcal{SOS}(x,a) \\
& \varepsilon - (\Psi_u - \Psi_l) - \mathcal{D}(\mathcal{M}, S_3) \in \mathcal{SOS}(x,a) \\
& \mathcal{B}(\Psi_l - \mathcal{D}(\mathcal{H}, S_4), \partial\Omega, T_1) \in \mathcal{SOS}(x,a) \\
& \mathcal{B}(\psi - \Psi_l - \mathcal{D}(\mathcal{H}, S_5), \partial\Omega, T_2) \in \mathcal{SOS}(x,a) \\
& \mathcal{B}(\Psi_u - \psi - \mathcal{D}(\mathcal{H}, S_6), \partial\Omega, T_3) \in \mathcal{SOS}(x,a) \\
& -\partial_{x^i}\Psi_l - \mathcal{D}(\Omega \cap \{x^i \geq 0\}, S_7) \in \mathcal{SOS}(x,a) \\
& \partial_{x^i}\Psi_l - \mathcal{D}(\Omega \cap \{-x^i \geq 0\}, S_8) \in \mathcal{SOS}(x,a) \\
& \Psi_l(0) - 1 - \mathcal{D}(\mathcal{H}, S_9) \in \mathcal{SOS}[a] \\
& -\Psi_l(0) + 1 - \mathcal{D}(\mathcal{H}, S_{10}) \in \mathcal{SOS}[a],
\end{aligned} \tag{4.19}$$

where $S = \{S_i\}_{i\in[10]_+}$, and $T = \{T_i\}_{i\in[3]_+}$. The operator $\mathcal{L}$ now depends on the variable a. The resulting solutions to the optimizations (4.19), and the upper bound suboptimal value functions $V_u = -\lambda \log \Psi_l$, are found for all $a \in \mathcal{H}$, with the control law $u^\varepsilon = -R^{-1}G^T\nabla_x V_u$ now stabilizing for the entirety of the uncertainty set $\mathcal{H}$. Similar techniques have been studied previously for Lyapunov analysis, *e.g.,* [88, Ch. 4.3.3].

4.8 Numerical Examples

This section studies the computational characteristics of this method using two examples: a scalar system and a two-dimensional system. In the following problems, the optimization parser YALMIP [89] was used in conjunction with the semidefinite optimization package MOSEK [90]. In both examples, the continuous system is integrated numerically using Euler integration with step size of 0.005s during simulations.

4.8.1 Scalar Unstable System

Consider the following scalar unstable nonlinear system

$$dx = \left(x^3 + 5x^2 + x + u\right) dt + d\omega \tag{4.20}$$

on the domain $x \in \Omega = \{x \mid -1 \leq x \leq 1\}$. The noise model considered is Gaussian white noise with zero mean and variance $\Sigma_\varepsilon = 1$. The goal is to stabilize the system at the origin. The first exit cost functional is used to implement this. We choose the boundary at two ends of the domain to be $\Psi(-1) = 20e^{-10}$ and $\Psi(1) = 20e^{-10}$, which is equivalent to setting a high cost V at the boundary. At the origin, the boundary is set as $\Psi(0) = 1$. We set $q = x^2$, and $R = 1$ in the cost functional. In the one dimensional case, the origin, which is a boundary, divides the domain into two partitions, $x \leq 0$ and $x \geq 0$. Because of the natural division of the domain, the solutions for both domains can be represented by smooth polynomial respectively, and solved independently. The simulation is terminated when the trajectories enter the interval $[-0.005, 0.005]$ centered on the origin.

The desirability functions that result from solving (4.3) for varying polynomial degrees are shown in Figure 4.2. The true solution is computed by solving the HJB directly in Mathematica [91]. The sharp corner at the origin is expected because the HJB PDE solution is not necessarily smooth at the boundary, and in this instance the origin is a zero-cost boundary.

The approximation error ε for both partitions is shown in Figure 4.3a for increasing polynomial degree. As seen in the plots, the approximation improves as the polynomial degree increases. Polynomial degrees below 14 are not feasible, and thus this data is absent in the plots. The suboptimal solution converges faster for $x > 0$ than for $x < 0$ when the degree of polynomial increases because the true solution

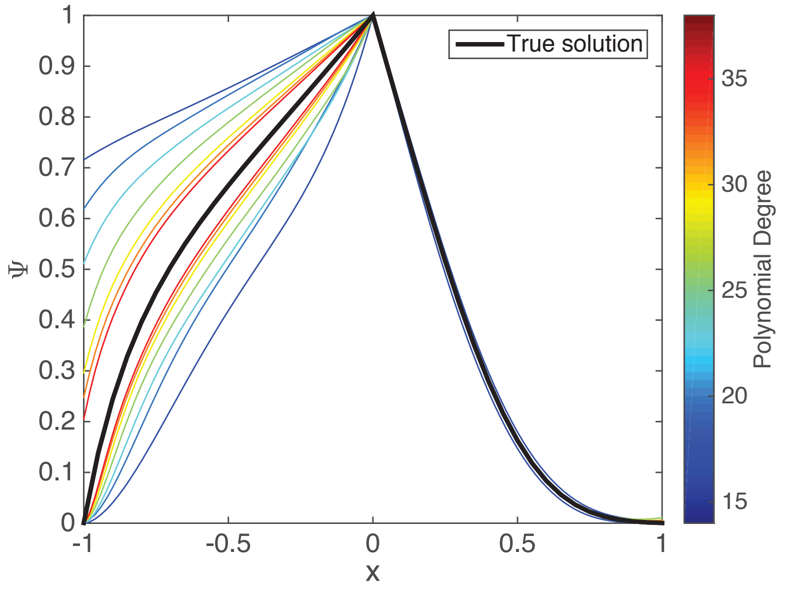
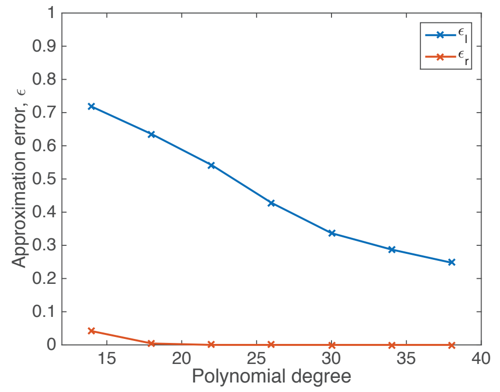
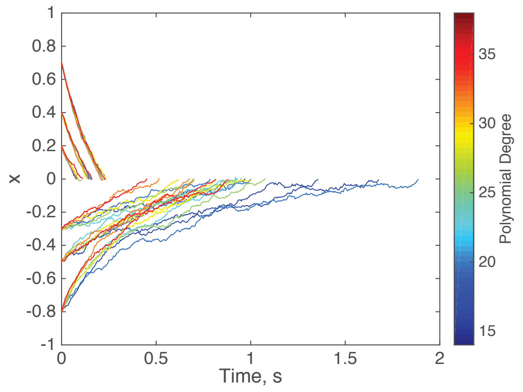


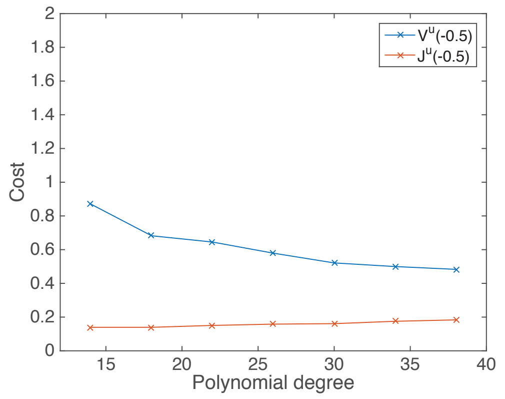
Figure 4.2: The desirability function of system (4.20) for varying polynomial degree. The true solution is the black curve.



(a)



(b)



(c)

Figure 4.3: Computational results of system (4.20). (a) Convergence of the objective function of (4.3) as the degree of polynomial increases. The approximation error for $x \leq 0$ is denoted as ε_l and the approximation error for $x \geq 0$ is denoted as ε_r. (b) Sample trajectories using controller computed from optimization problem (4.3) with different polynomial degrees starting from six randomly chosen initial points. (c) The comparison between J_u and V_u for different polynomial degrees, whereby J_u is the expected cost and V_u is the value function computed from optimization problem (4.3). The initial condition is fixed at $x_0 = -0.5$.

for $x > 0$ has a simple quadratic-like shape that can be easily represented as a low degree SOS function.

Figure 4.3b shows sample trajectories using the controller computed from optimization problem (4.3) for different polynomial degrees. The controllers are stabilizing for six randomly chosen initial points when the system is perturbed by random samples of Gaussian white noise with $\Sigma_\varepsilon = 1$. Unsurprisingly, the suboptimal solutions with low pointwise error result in the system converging towards the origin faster.

To compare between the cost function J_u and the approximated value function

V_u, a Monte Carlo experiment is illustrated in Figure 4.3c. For each polynomial degree that is feasible, the controller obtained from Ψ_l in optimization problem (4.3) is implemented in 30 simulations of the system subject to random samples of Gaussian white noise with $\Sigma_\varepsilon = 1$. The initial condition is fixed at $x_0 = -0.5$. In the figure, $V^u \geq J^u$ as expected, and the difference between the two decreases with increasing d.

4.8.2 Two Dimensional System

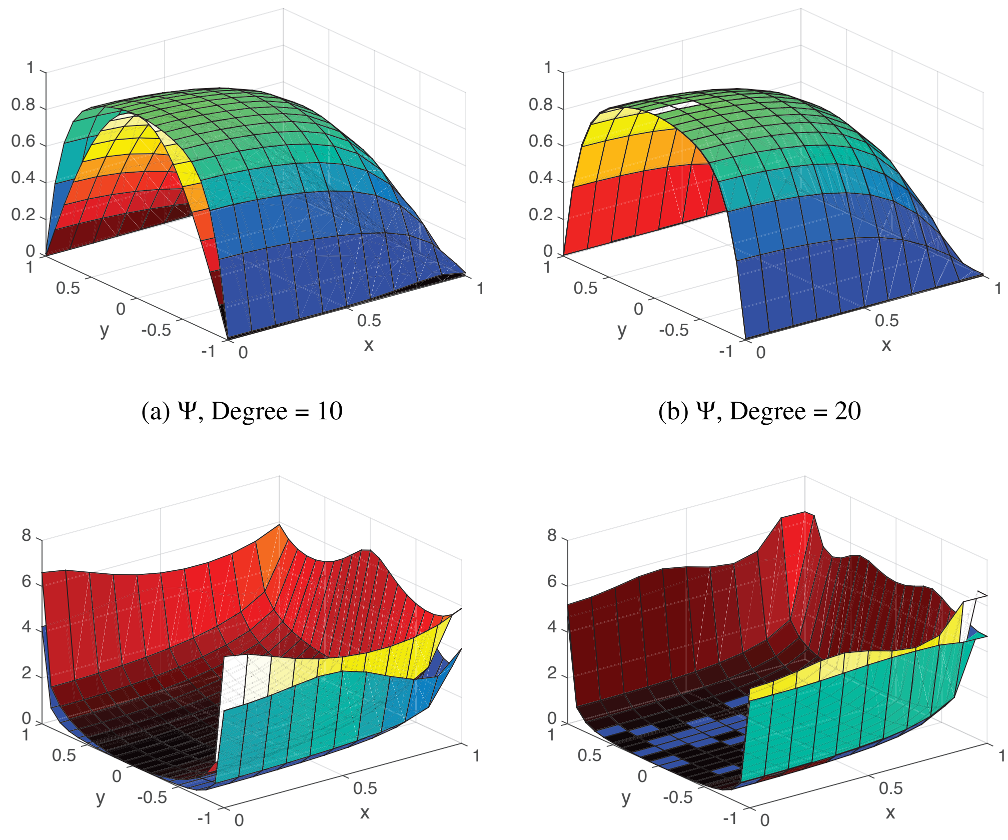
Consider a nonlinear 2-dimensional example with the following dynamics:

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \left(2 \begin{bmatrix} x^5 - x^3 - x + xy^4 \\ y^5 - y^3 - y + yx^4 \end{bmatrix} + \begin{bmatrix} x\, u_1 \\ y\, u_2 \end{bmatrix} \right) dt + \begin{bmatrix} x\, d\omega_1 \\ y\, d\omega_2 \end{bmatrix}. \tag{4.21}$$

The goal is to reach the origin at the boundary of the domain $\Omega = \{(x, y) \mid -1 \leq x \leq 1, -1 \leq y \leq 1\}$. This goal is implemented via a first exit cost functional. The control penalty is $R = I_{2\times 2}$, and state cost is $q(x) = x^2 + y^2$. The boundary conditions for the sides at $x = 1, x = -1, y = 1$, and $y = -1$ are set to $\phi(x, y) = 5$, while at the origin, the boundary has cost $\phi(0, 0) = 0$. The noise model considered is Gaussian white noise with zero mean and an identity covariance matrix.

The approximated desirability functions and their corresponding value functions are shown in Figure 4.4, with half of the domain $x \in [0, 1]$ shown in order to view the gaps between the upper and lower bound solutions. Figure 4.5a shows the convergence of the objective function of optimization problem (4.3) as the degree of polynomial increases. There is no data below degree of 10 because the optimization problem is not feasible in these cases. As shown in Figure 4.5b, sample trajectories starting from six different initial points shows that the controllers computed from Ψ_l for various degrees arrive at the origin. The trajectory is considered to reach the origin if it is within an Euclidean distance of 0.01 from the origin.

Similar to the scalar example, a Monte Carlo experiment is performed to compare between J_u and V_u. For each polynomial degree that yields a feasible solution, the controller obtained from Ψ_l in optimization problem (4.3) is implemented in 30 simulations of the system subject to random samples of Gaussian white noise with $\Sigma_\varepsilon = I_{2\times 2}$. The initial condition is fixed at $x_0 = (0.7, 0.7)$. Figure 4.5c shows the comparison between J_u and V_u for different polynomial degrees, whereby J_u is the expected cost and V_u is the value function computed from Ψ_l in optimization problem (4.3). As expected, $V^u \geq J^u$.



(a) Ψ, Degree = 10

(b) Ψ, Degree = 20

(c) V, Degree = 10

(d) V, Degree = 20

Figure 4.4: Approximated desirability functions and value functions for (4.21) when polynomial degrees are 10 and 20. In (a) and (b), the blue sheets are the upper bound solutions Ψ_u and the red sheets are the lower bound solutions Ψ_l. The corresponding value functions are shown in (c) and (d), respectively.

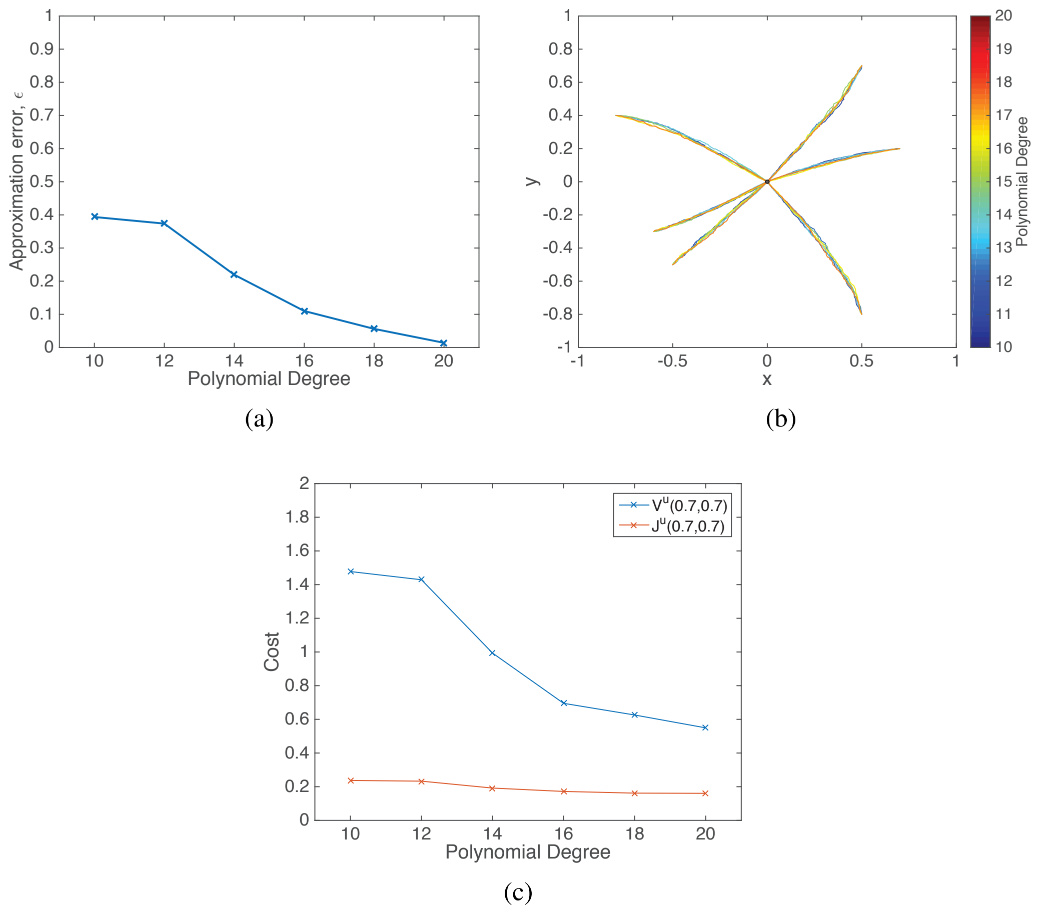


Figure 4.5: Computational results of system (4.21). (a) Convergence of the variables in the objective function of (4.3). (b) Sample trajectories using controller from optimization problem (4.3) with different polynomial degrees starting from six randomly chosen initial points. (c) The comparison between J_u, the expected cost, and V_u the value function for different polynomial degrees from optimization problem (4.3). The initial condition is fixed at $x_0 = (0.7, 0.7)$.

4.8.3 Using DSOS and SDSOS

Recently, new numerical approaches [92], [93] – diagonally dominant SOS (DSOS) and scaled diagonally dominant SOS (SDSOS) – have been developed to improve the computation burden of SOS programing. These approaches reformulate the SDP used to solve a SOS program into a linear program and a second order cone program, respectively. As a result, the computations are more efficient. However, the accuracy of the solution may decrease, and thus typically, a higher degree of polynomial is necessary for achieving a similar level of accuracy.

This section discusses some preliminary results of using DSOS and SDSOS to solve for the linear HJB equation for the two examples. For the scalar unstable system, the DSOS program is unable to find a feasible solution for polynomial degree up to 118, the maximum degree that is tested. For SDSOS programing, the smallest degree at which the solution exists is 74 with an approximation error ε of 0.846. The computation time is approximately 1.5 seconds, comparable with the computation time of the SOS program (approximately 1.1 seconds) on the same computer. The polynomial degree of the SOS program is 14 and the approximation error is 0.718 when the program first becomes feasible.

For the two dimensional example, DSOS is unable to find a feasible solution for degree up to 30. The computation time is 49 minutes when the polynomial degree is 30. Using SDSOS, a feasible solution is found when polynomial degree is 18 with an approximation error $\varepsilon = 0.516$ and computation time of 26 seconds. In constrast, the SOS program found a feasible solution with polynomial degree of 10 with an approximation error $\varepsilon = 0.394$ and computation time of 9 seconds. Based on these initial findings, the potential benefit of using DSOS and SDSOS to solve the linear HJB equation may be limited.

4.9 Summary

This chapter has proposed a new method to approximate the solution to a class of optimal control problems for stochastic nonlinear systems via SOS programming. Analytical results provide guarantees on the suboptimality of the trajectories that result when using the approximate solutions for controller design. Consequently, one can synthesize a suboptimal stabilizing controller for a large class of stochastic nonlinear dynamical systems.

The development of this work has been limited to nonlinear systems governed by polynomial functions. A number of avenues exist for incorporating non-polynomial nonlinearities apart from projecting the non-polynomial functions to a polynomial basis. First, it is possible to use a non-polynomial basis of approximating functions in conjunction with SOS programming. The only limitation to this approach is that the basis of such a set must be closed under addition, multiplication, and differentiation. An example is to consider signomials as the basis functions for approximation. Signomials are exponentials composed with linear functionals. Recent work by Chandrasekaran *et al.* [94] shows that one could solve a sequence of convex programs to achieve increasingly better approximations. Alternatively, it is possible to transform non-polynomial nonlinearities to polynomial nonlinearities with additional equality constraints, with an increase in the requisite number of optimization variables [38].

As is commonly seen when using SOS programming, the numerics of the SDP may be cumbersome in practice. There are a number of avenues for future work aimed at improving the practical performance. First, the monomials of the polynomial approximation can be chosen strategically in order to decrease computation time while achieving high accuracy. A promising future direction is the synthesis of the work presented here with that of [95], wherein the curse of dimensionality is avoided via the strategic choice of basis functions. To improve the numerical conditioning of these optimization techniques, a domain partitioning technique is studied in [86], wherein the alternating direction method of multipliers is used to enable both parallelization and a solution representation that varies in resolution over the domain. In addition, there exists a growing body of literature towards increasing the numeric stability and scalability of SOS techniques [92], [93], [96]–[98].

The next chapter will introduce a low rank tensor-based approach that avoids the curse of dimensionality, but at the expense of analytical guarantees presented in this chapter.

Chapter 5

HIGH DIMENSIONAL OPTIMAL CONTROLLER SYNTHESIS USING LOW-RANK TENSOR DECOMPOSITION

This chapter presents a numerical technique to efficiently solve the Hamilton-Jacobi-Bellman (HJB) equation for a class of stochastic affine nonlinear dynamical systems in high dimensions. The HJB solution provides a globally optimal controller to the associated dynamical system [48]. The linear form of the linear HJB equation enables the use of off-the-shelf numerical PDE solvers to compute a solution of the linear HJB equation. However, the curse of dimensionality quickly causes the problem to become intractable for systems with modest dimensions [22] because the number of degrees of freedom required to solve the optimal control problem grows exponentially with dimension. Yet robots or other engineering systems commonly have at least six degrees of freedom.

The curse of dimensionality, commonly found in robotic systems, prevents one from solving the HJB equation naively. In this chapter, the curse is avoided by representing the linear HJB equation efficiently using low rank tensor decomposition (see review in Section 3.8). Then, an alternating least squares (ALS) based technique finds an approximate solution to the linear HJB equation. This work is an extension of [95], where the idea of using ALS and tensor decomposition to solve the HJB equation was first considered for the first exit problem.

Previous research has considered multiple approaches to alleviate this curse of dimensionality. These techniques include using sparse grid [99], Taylor polynomial approximation [100], max-plus method [101], and model reduction [102]. In contrast to the previous work, this paper focuses on using a *low rank tensor decomposition* technique to represent and solve the linear HJB equation. The complexity of this approach grows linearly with the number of dimensions [81]. Others [103] have considered using a tensor decomposition based technique. However, this technique solves for an optimal controller by using value iteration, instead of the HJB equation. Machine learning techniques [104] and optimizations based on Lax and Hopf formulas [105] are also used to solve high dimensional HJB equations, but they currently only apply to parabolic PDEs.

The major contributions of this chapter are the improvement of the ALS algorithm from the original work [95] and the development of the SeALS tool in MATLAB [32]. A straightforward implementation of the ALS algorithm results in ill-conditioned matrices that prevent approximation to a high order of accuracy. The ill-conditioning issue is resolved by computing the solution sequentially and introducing boundary condition rescaling. Both of these additions reduce the condition number of matrices in the ALS-based algorithm. A new MATLAB tool, Sequential Alternating Least Squares (SeALS) that implements the new method is developed. SeALS can solve high dimensional linear HJB equations more accurately with shorter computation time.

In addition to the improvements of the ALS algorithm, this chapter also extends the use of low rank tensor decomposition technique to problems with other cost functionals, namely finite horizon and infinite horizon cost functionals described in Section 3.4. A time stepping algorithm and inverse power method are brought together with low rank tensor decomposition framework in order to solve the finite horizon and infinite horizon problems respectively.

SeALS is used to compute optimal controllers for multiple simulated examples, including balancing an inverted pendulum, landing a Vertical Takeoff and Landing (VTOL) aircraft, and stabilizing a quadcopter. The ability to compute the solution of 12-dimensional linear HJB equation of a quadcopter on a personal laptop suggests that SeALS has a great potential for use in robotics applications. Some contents of this work appeared in [35].

The rest of this chapter is organized as follows. Section 5.1 formulates the problem and discusses the underlying assumptions. Section 5.2 presents the method to convert the linear HJB equation into a CP tensor representation. Sections 5.3–5.5 discuss the controller synthesis procedure for first exit, finite horizon, and infinite horizon problems respectively. Section 5.6 discusses limitations of the original ALS algorithm, describes the improvements using sequential computation and boundary condition rescaling, and presents a summary of the improved SeALS algorithm. The implementation of the algorithms may be found in [32]. Conclusions and recommendation for future work are given in Section 5.8. The implementation of the algorithms described in previous sections for different examples are deferred to Appendix A for ease of reading. The examples include a vertical takeoff and landing (VTOL) aircraft and a quadcopter.

5.1 Problem Formulation

This chapter proposes a numerical technique to synthesize an optimal controller for system (3.1) in high dimensions (*i.e.,* n is large) with respect to cost functionals described in Section 3.4.1.

The systems dynamics and the cost functionals are assumed to be governed by separable functions. A function $f(x)$ is separable if it can be represented as $f(x) = \sum_{l=1}^{r} s_i \prod_{i=1}^{d} f_i^l(x_i)$.

Assumption 5.1. System (3.1) and the cost functionals (*i.e.,* f, G, B, ϕ, and q) are described by separable functions.

In numerical implementation, Ω is always defined as a compact bounded domain that is a subset of $\mathbb{R}^n$. As a result, if a function in the problem formulation is not separable, it can be approximated as a separable function, for example, the Fourier series or a polynomial that can be made arbitrarily accurate by the Stone-Weierstrass Theorem [87].

More precisely, this chapter focuses on using low rank tensor decomposition to numerically solve the associated linear HJB equations (3.10) with respect to the three cost functionals.

Remark. In this chapter, we will let integer d denote the dimension of the state vector sin (3.1) to be consistent with the code implementation in [32]. The symbol n_i represents the number of points for the i-th dimension instead.

5.2 Low Rank Tensor Representation

This section shows how the linear HJB equation (3.10) can be numerically solved using low rank tensor decomposition. Depending on the cost functionals, the specific implementation might differ. However, the conversion from the linear HJB equation (3.10) into CP tensor representation is the same across all cost functionals. Later sections present the techniques specific to each cost functional and provide the complete algorithms. This section focuses on converting (3.10) into CP tensor representation beginning with state space discretization described next.

5.2.1 State Space Discretization

Given the domain $\Omega \in \mathbb{R}^d$, discretize the domain Ω by n_i grid points in the i-th dimension, and represent the discretized domain using a multi-dimensional tensor $T \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$. Each point $(m_1, m_2, \ldots, m_d)$ in T corresponds to a grid point in the discretized domain, denoted by $(X_1(m_1), X_2(m_2), \ldots, X_d(m_d))$. For example, Ω with $d = 2$ is an ordinary rectangle, whereby T is a $n_1 \times n_2$ matrix. Depending on the choice of discretization as described in Section 3.7, the specific location of $X_i(m_i)$ may differ. Let $Grid(\Omega)$ represents the discretization of Ω.

5.2.2 Operator-Tensor Conversion

Given the discretized domain, to represent the operator $\mathfrak{L}$ by a tensor operator $\mathbb{L}$, first approximate each functions within the matrix-valued function q, f, B, and Σ_ε in $\mathfrak{L}$ with CP tensor functions as described in (3.8). These functions form matrices that are composed of corresponding tensor functions, denoted *matrix-valued tensor functions*.

Next, convert the matrix-valued tensor functions of q, f, B and Σ_ε into tensor operators. For each tensor function in a given matrix-valued tensor function, diagonalize the basis functions according to

$$\sum_{l=1}^{r_F} \bigotimes_{i=1}^{d} F_i^l \rightarrow \sum_{l=1}^{r_F} \bigotimes_{i=1}^{d} diag(F_i^l), \tag{5.1}$$

where $diag(F)$ is a diagonal matrix with vector F as the diagonal. The matrix-valued tensor functions in this new form are called *matrix-valued tensor operators*. Let q_{op}, f_{op}, B_{op}, and $\Sigma_{\varepsilon,op}$ denote the matrix-valued tensor operators of q, f, B, and Σ_ε respectively.

Now, approximate the gradient ∇_x and the Hessian ∇_{xx} in $\mathfrak{L}$ by matrix-valued tensor operators $\mathbb{D}_1$ and $\mathbb{D}_2$, respectively. The tensor decomposition of the partial derivative in k-th dimension is given by

$$\nabla_k = I_1 \otimes \ldots I_{k-1} \otimes D \otimes I_{k+1} \otimes \cdots \otimes I_d, \tag{5.2}$$

where D is a difference matrix and I_j is the identity matrix for all $j \in [d]_+$. The matrix-valued tensor operator $\mathbb{D}_1$ of the gradient ∇_x is formed with entries ∇_k, while the matrix-valued tensor operator $\mathbb{D}_2$ of the hessian ∇_{xx} is formed with entries $\nabla_{k,j} = \nabla_k \cdot \nabla_j$.

Lastly, multiply and add all matrix-valued tensor operators to form the tensor operator $\mathbb{L}$

$$\mathbb{L}(\cdot) = -\frac{1}{\lambda} q_{op}(\cdot) + f_{op}^T \mathbb{D}_1(\cdot) + \frac{1}{2} Tr(\mathbb{D}_2(\cdot) B_{op} \Sigma_{\varepsilon,op} B_{op}^T).$$

As a result, the linear HJB equation (3.10) takes one of the following forms:

$$\text{First exit:} \quad 0 = \mathbb{L}F \tag{5.3a}$$

$$\text{Finite horizon:} \quad -F_+ = \mathbb{L}F \tag{5.3b}$$

$$\text{Infinite horizon:} \quad -cF = \mathbb{L}F, \tag{5.3c}$$

where F is a tensor function that corresponds to the desirability function Ψ, $\mathbb{L}$ is a tensor operator that corresponds to $\mathfrak{L}$, and F_+ represents the discrete time stepping. Typically, $F_+ = (F(i+1)-F(i))/h$, where i is the time iteration and h is the time step, but the F at the right hand side of the equation can be $F(i+1)$, $F(i)$, or other more sophisticated forms depending on the time integration scheme being employed. For example, $F(i+1)$ is used for the backward Euler scheme. The boundary conditions are discretized accordingly as well, and the details will be discussed in the individual sections for each cost functional. Henceforth, let

$$\mathbb{L} = \sum_{l=1}^{r_{\mathbb{L}}} \bigotimes_{i=1}^{d} L_i^l.$$

Furthermore, let *ComputeTensorOperator* be the function that converts a given linear HJB operator $\mathfrak{L}$ and discretization $Grid(\Omega)$ into a tensor operator $\mathbb{L}$.

Given the tensor operator $\mathbb{L}$, the rest of this chapter discusses how to solve the three classes of linear HJB equation (3.10) using the CP tensor representation. Before moving on to the approach for first exit problem, we briefly describe how to compute the controller given F, the solution of (3.10). Recall that the optimal controller is given by $u = -R^{-1}G^T\nabla_x V = \lambda R^{-1}G^T\frac{\nabla_x\Psi}{\Psi}$. The tensor F is the discretization of Ψ, and $\nabla_x\Psi$ can be computed by numerically differentiating F. Thus, u for a given x can be computed with the given formula. If x is not a value on the $Grid(\Omega)$, the value of Ψ and $\nabla_x\Psi$ are interpolated using the neighboring points. The implementations of the controllers on a few engineering examples are presented in Appendix A. Next, the details on how to solve the three classes of linear HJB equation (3.10) using the CP tensor representation are discussed.

5.3 Controller Synthesis for First Exit Problem

This section presents the approach to solve the linear HJB equation (3.10a) for the first exit problem. The linear HJB equation (3.10a) is first discretized and represented using CP tensor as described in Section 5.2. Then, the boundary conditions ψ are incorporated into the CP tensor formulation (5.3a).

To solve (5.3a) with the associated boundary conditions (3.11a), we will first form a linear tensor equation $\mathbb{A}F = G$ whereby $\mathbb{A}$ and G will incorporate both $\mathbb{L}$ and the boundary conditions (3.11a), and F is the tensor function in (5.3a) representing the desirability function Ψ. Then, this linear tensor equation is solved using the ALS algorithm (Algorithm 1). Once an approximated desirability function, F, is obtained, the controller is computed from F according to $u = -R^{-1}G^T\nabla_x V$, where $V = -\lambda \log F$. Algorithm 2 summarizes the approach. Each step is described in more detail below.

First, the tensor function G is constructed from the boundary condition ψ by *ComputeG* in Algorithm 2 analogously to the construction of the tensor operator $\mathbb{L}$ for $\mathfrak{L}$ in *ComputeTensorOperator*. For Dirichlet boundary conditions, the elements in G that correspond to the boundary points $x \in \partial\Omega$ is set to $e^{-\frac{\phi(x)}{\lambda}}$. The rest of the elements of the tensor function G in the interior $x \in \Omega$ are set to zero.

To illustrate the construction, consider the case where the boundary of Ω is set to be a constant a. For each dimension k, create a tensor function

$$Q_k = I'_1 \otimes \cdots \otimes I'_{k-1} \otimes C \otimes I_{k+1} \otimes \cdots \otimes I_d, \tag{5.4}$$

where $I'_j \in \mathbb{R}^{n_j}$, $I'_j = \begin{pmatrix} 0 & 1 & \cdots & 1 & 0 \end{pmatrix}^{\mathrm{T}}$ for $j \in [k-1]_+$, $C \in \mathbb{R}^{n_k}$, $C = \begin{pmatrix} a & 0 & \cdots & 0 & a \end{pmatrix}^{\mathrm{T}}$, and $I_j = \mathbb{1}_{n_j}^{\mathrm{T}}$ is a vector of all ones for $j = k+1, \ldots, d$. Repeat the process for $k \in [d]_+$. Then, $G = \sum_{k=1}^{d} Q_k$.

A similar approach can be applied to more complicated Dirichlet boundary values and Neumann boundary conditions. For a periodic boundary condition at the k-th dimension, Q_k is set to be a zero tensor function.

Then, operator $\mathbb{A}$ is constructed from $\mathbb{L}$ and the boundary conditions ψ. We first illustrate the procedure for Dirichlet boundary conditions. Recall that $\mathbb{L} = \sum_{l=1}^{r_{\mathbb{L}}} \bigotimes_{i=1}^{d} L_i^l$.

Algorithm 2 First Exit Problem

```
Input: Linear operator $\mathfrak{L}$, boundary conditions $\psi$, domain $\Omega$, accuracy tolerance $\varepsilon$,
    and initial rank $r_0$.
Output: Tensor function $F$.
1: $\mathbb{L} :=$ ComputeTensorOperator($\mathfrak{L}$, Grid($\Omega$))
2: $G :=$ ComputeG($\psi$, Grid($\Omega$))
3: $\mathbb{A} :=$ ComputeA($\mathbb{L}$, $\psi$, Grid($\Omega$))
4: $\mathbb{A} :=$ CompressTensor($\mathbb{A}$)
5: $F :=$ ALS($\mathbb{A}$, $G$, $\varepsilon$, $r_0$)
```

Let $\bar{\mathbb{A}} = \sum_{l=1}^{r_A} \bigotimes_{i=1}^{d} A_i^l$, where $r_A = r_{\mathbb{L}}$, and $A_i^l = L_i^l$ for all $l \in [r_{\mathbb{L}}]_+$ and $i \in [d]_+$. Given a Dirichlet boundary condition in the k-th dimension, replace the first and last row in the matrices $\{A_k^l\}_{l \in [r_A]_+}$ by zeros. The operator now acts as zero on the boundary in the k-th dimension. Then, create a new tensor operator

$$P_k = I_1' \otimes \cdots \otimes I_{k-1}' \otimes C \otimes I_{k+1} \otimes \cdots \otimes I_d, \tag{5.5}$$

where $I_j' \in \mathbb{R}^{n_j \times n_j}$ is the identity matrix but the first and last rows are all zeros for $j \in [k-1]_+$, C is a zero matrix except for two ones in the first and last diagonal elements, and $I_j \in \mathbb{R}^{n_j \times n_j}$ is the identity matrix for $j = k+1, \ldots, d$. The operator P_k acts as an identity on the boundary in the k-th dimension. Repeat this process for $k \in [d]_+$. Let $\mathbb{A} = \bar{\mathbb{A}} + \sum_{k=1}^{d} P_k$. Then, the new operator $\mathbb{A}$ sets the boundary values of F given a G in $\mathbb{A}F = G$.

So far, only Dirichlet boundary conditions has been considered. A slight modification in the current procedure also enables Neumann boundary conditions and periodic boundary conditions to be included. For Neumann conditions at the k-th dimension, the first and last row of C in P_k are given by the first and last row of the difference matrix, respectively, calculating the directional derivative at the boundary. If using the finite difference discretization scheme, for periodic boundary condition at the k-th dimension, the first row of $\{A_k^l\}_{l \in [r_A]_+}$ is replaced by zeros. The zero matrix C has 1 and -1 as first and last element in the first row. Under spectral differentiation, the terms $\{A_k^l\}_{l \in [r_A]_+}$ are not modified, and the P_k is a zero tensor for periodic boundary condition at the k-th dimension.

The operator $\mathbb{A}$ often has a high separation rank, and therefore *CompressTensor* compresses $\mathbb{A}$ to a low rank approximation using Algorithm 1 and the identity operator as discussed in Section 3.8.

Lastly, given $\mathbb{A}$ and G, *ALS* solves for F in $\mathbb{A}F = G$ using Algorithm 1. The obtained solution F is the approximate solution to the linear HJB equation (3.10a) given as a tensor function (5.3a), as in Definition 3.8. Once F is computed, the control input is given by $u = -R^{-1}G^T \nabla_x V$, where $\nabla_x V = -\lambda \frac{\nabla_x F}{F}$. Recall that $\nabla_x F$ can be computed by multiplying (5.2) with F for each dimension. Therefore, given a state $x \in \Omega$, we can recover the value of $\nabla_x F$ at x and compute $u = -R^{-1}G^T \nabla_x V$ by simple linear algebra operations as described in Section 3.8.

Figure 5.1 shows an implementation of the controller produced by Algorithm 2 on a quadcopter example. The goal of the quadcopter is to reach $p_x = 1$. The algorithm for this example uses an improved ALS algorithm, SeALS, that is described later in Section 5.6. For more detailed descriptions of this example (Example A.5) and other examples of first exit problem, refer to Appendix A.

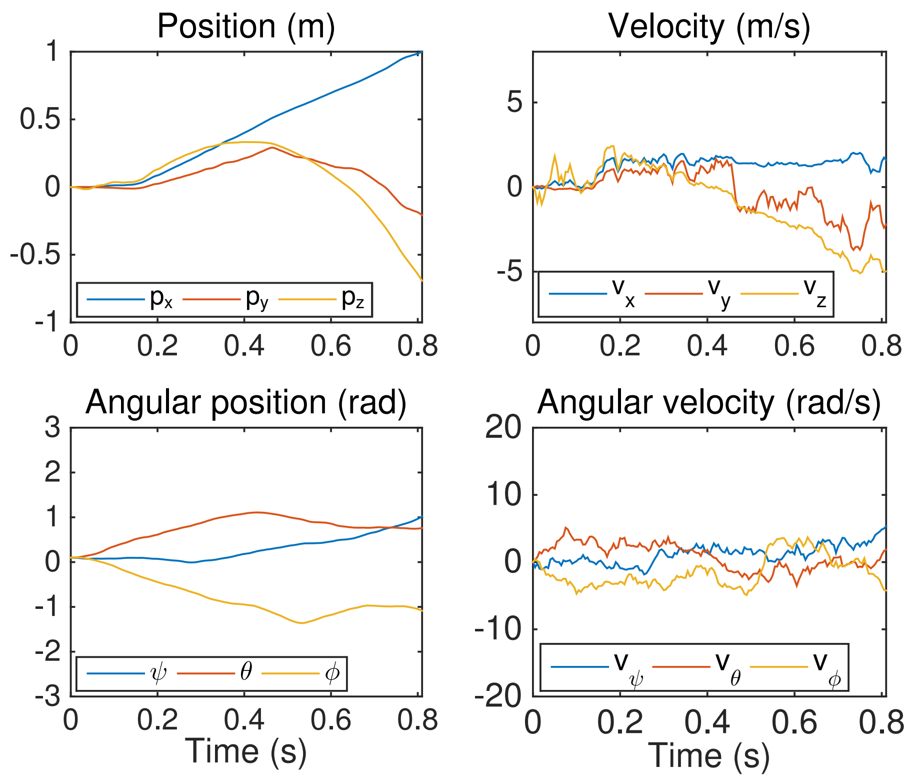


Figure 5.1: A simulation of a quadcopter controlled using the controller produced by Algorithm 2 and SeALS (Algorithm 7). The quadcopter reaches $p_x = 1$ as desired.

5.4 Controller Synthesis for Finite Horizon Problem

This section discusses the technique to solve the finite horizon problem using (3.10b) with boundary condition (3.11b) in the CP tensor framework. The linear HJB equation (3.10b) is first discretized and represented using a CP tensor as described in Section 5.2. Then, backward Euler is implemented together with the ALS algorithm (Algorithm 1). This section begins with an overview of the finite horizon algorithm (Algorithms 3 and 4), and follows with descriptions of each step in the algorithm. Many functions in Algorithms 3 and 4 are the same as Algorithm 2, and thus refer to Section 5.3 for details of these functions. Only functions that are specific to Algorithm Algorithms 3 and 4 are explained in this section.

In general, Algorithms 3 and 4 compute the sequence $\{F_i\}_{i=1}^{T/h}$ that forms the solutions to (3.10b) given the linear operator $\mathfrak{L}$, the set of boundary conditions ψ, the time increment h, the end time T, accuracy tolerance ε, and initial rank r_0. The algorithm first forms the tensor $\mathbb{L}$ and initial condition F_0 from the given discretization. Then, depending on if it's a forward Euler iteration or a backward Euler iteration, it forms $\mathbb{B}$ the tensor form of the $(I + Fh)$ term in (3.16) or $(I - Fh)$ term in (3.17). The function *ComputeA* incorporates the boundary conditions into $\mathbb{B}$ to form the final operator $\mathbb{A}$ for the backward Euler iterations. At each step of the iterations, $F_i = \mathbb{A}F'_{i-1}$ is computed for forward Euler, and ALS algorithm is used to solve the equation $\mathbb{A}F_i = F'_{i-1}$ for backward Euler.

The algorithm step is described with more details next. The function *ComputeInit* computes the tensor function F_0 that represents the final time condition in the boundary conditions (3.11b) by Definition 3.8 using discretization *Grid*(Ω). The function *ComputeTensorOperator* is discussed in detail in Section 5.3. After $\mathbb{L}$ is obtained, for forward Euler, the algorithm create the tensor form of the $(I + Fh)$ term in (3.16) by assigning $\mathbb{B} = \mathbb{I} + \mathbb{L}h$, where h is the time increment, $\mathbb{I} \triangleq \bigotimes_{i=1}^{d} I$ is the identity tensor operator, and $I \in \mathbb{R}^{n_i \times n_i}$ is the identity matrix. The sign in front of $\mathbb{A}$ should be negative based on (3.10b), but it is positive because (3.10b) is solved backward in time. Similarly, for backward Euler, the tensor form of the $(I - Fh)$ term in (3.17) is created.

Next, *ComputeA* (described in Section 5.3) incorporates the boundary condition into $\mathbb{B}$ and forms the final tensor operator $\mathbb{A}$ that is used for the time stepping iteration. The algorithm performs the time stepping iterations based on forward or backward Euler integration beginning with the initial value F_0.

For each iteration, the function *UpdateBoundary* forms a new tensor function F'_{i-1} that replaces the value at the boundary of F_{i-1} with the boundary value of F_i given by the Dirichlet boundary condition. More precisely, let $F_{i-1} = \sum_{l=1}^{r} \bigotimes_{j=1}^{d} f_j^l$.

Algorithm 3 Finite Horizon Problem - Forward Euler

```
Input: Linear operator $\mathfrak{L}$, boundary conditions $\psi$, domain $\Omega$, end time $T$, and time
    increment $h$.
Output: Tensor functions, $F_i$ for $i = 1, 2, \ldots, T/h$.
 1: $F_0 := ComputeInit(\psi, Grid(\Omega))$
 2: $\mathbb{L} := ComputeTensorOperator(\mathfrak{L}, Grid(\Omega))$
 3: $\mathbb{B} =: \mathbb{I} + \mathbb{L}h$
 4: $\mathbb{A} := ComputeA(\mathbb{B}, \psi, Grid(\Omega))$
 5: for $i = 1, 2, \ldots, T/h$ do
 6:     $F'_{i-1} := UpdateBoundary(F_{i-1}, \psi)$
 7:     $F_i := \mathbb{A}F'_{i-1}$
 8: end for
```

Algorithm 4 Finite Horizon Problem - Backward Euler

```
Input: Linear operator $\mathfrak{L}$, boundary conditions $\psi$, domain $\Omega$, accuracy tolerance $\varepsilon$,
    end time $T$, time increment $h$, and initial rank $r_0$.
Output: Tensor functions, $F_i$ for $i = 1, 2, \ldots, T/h$.
1: $F_0 := ComputeInit(\psi, Grid(\Omega))$
2: $\mathbb{L} := ComputeTensorOperator(\mathfrak{L}, Grid(\Omega))$
3: $\mathbb{B} =: \mathbb{I} - \mathbb{L}h$
4: $\mathbb{A} := ComputeA(\mathbb{B}, \psi, Grid(\Omega))$
5: for $i = 1, 2, \ldots, T/h$ do
6:     $F'_{i-1} := UpdateBoundary(F_{i-1}, \psi)$
7:     $F_i := ALS(\mathbb{A}, F'_{i-1}, \varepsilon, r_0)$
8: end for
```

Let $\bar{F}_{i-1}$ be the same as F_{i-1}. Given a Dirichlet boundary condition at the k-th dimension, the first and last elements of $\{f_k^l\}_{l=1}^r$ in $\bar{F}_{i-1}$ are replaced with zeros. Then, create a new tensor function

$$Q_k = I'_1 \otimes \cdots \otimes I'_{k-1} \otimes C \otimes I_{k+1} \ldots I_d,$$

where $I'_j \in \mathbb{R}^{n_j}$, $I'_j = \begin{pmatrix} 0 & 1 & \ldots & 1 & 0 \end{pmatrix}^{\mathrm{T}}$ for $j = 1, \ldots, k-1$, $C \in \mathbb{R}^{n_k}$, $C = \begin{pmatrix} a_k & 0 & \ldots & 0 & b_k \end{pmatrix}^{\mathrm{T}}$, a_k and b_k are the boundary values of F_i in the k-th dimension, $I_j \in \mathbb{R}^{n_j}$, and $I_j = \begin{pmatrix} 1 & 1 & \ldots & 1 & 1 \end{pmatrix}^{\mathrm{T}}$ is a vector of all ones for $j = k+1, \ldots, d$. Repeat the process for $k = 1, \ldots, d$. Then, set $F'_{i-1} = \bar{F}_{i-1} + \sum_{k=1}^{d} Q_k$, where F'_{i-1} is the same as F_{i-1}, but the boundary values of F'_{i-1} are now the boundary values of F_i.

At this point, we have shown how to incorporate the Dirichlet boundary condition. For other types of boundary conditions including Neumman boundary condition and periodic boundary condition, the same algorithm applies, but the specific implementations of *ComputeA* and *UpdateBoundary* will be different as previously discussed in Section 5.3. Similarly, treatments of the boundary conditions for spectral differentiation scheme and the finite difference scheme may differ as well.

Lastly, once F'_{i-1} is obtained, the forward Euler algorithm computes $F_i = \mathbb{A}F'_{i-1}$. On the other hand, the backward Euler algorithm solves for F_i in $\mathbb{A}F_i = F'_{i-1}$ using *ALS*. This process is repeated until the iteration arrives at the final time T.

Performance of Algorithms 3 and 4 can be improved with simple modifications such as applying Algorithm 1 or other rank reduction algorithm [106] for tensors after Step 1 and Step 5 to reduce the rank of $\mathbb{B}$ and F_i respectively (*i.e.,* implement

CompressTensor). This modification will make sure the rank of the tensors do not grow too quickly after each operation, and thus reduce computation time.

Apart from computation time, another improvement that is implemented is rescaling. As i increases (*i.e.,* as time decreases from end time T), the cost-to-go tends to increase. This increase results in small magnitude of the desirability function, which then loses its precision due to numerical truncation near zero. To preserve the accuracy, the function F_i is rescaled at every iteration as shown in Algorithm 5 for the forward Euler case. The backward Euler case would have a similar construction except that line 8 would be $F'_i := ALS(\mathbb{A}, F'_{i-1}, \varepsilon, r_0)$ instead.

More precisely, for each iteration i (line 6-11 in Algorithm 5), the algorithm keeps track of two variables F^s_i, the shape function, and s_i, the scaling constant, where $F_i = s_i F^s_i$. The iteration begins with $s_0 = 1$ and $F^s_0 = F_0$. Then, instead of solving for F_i using F_{i-1}, it uses F^s_{i-1}. The scaling constant is chosen to be the value of F'_i at the origin because for most control problems, the origin is the point with the lowest cost (*i.e.,* largest desirability function value). Other options for scaling constant include the maximum of F'_i or the norm of F'_i. At the end of the algorithm, we obtain the shape functions F^s_i for $i = 1, 2, \ldots, T/h$ that can be used to compute the controller. Note that the controller is given by $u = \lambda R^{-1} G^T \frac{\nabla_x \Psi}{\Psi}$. As a result, the scaling constants are not necessary for the computation of the controller.

As with any finite horizon problem, the choice of the initial condition (*i.e.,* the desirability function at the end time T or the terminal cost function ϕ) will determine

Algorithm 5 Finite Horizon Problem - Scaling

```
Input: Linear operator $\mathfrak{L}$, boundary conditions $\psi$, domain $\Omega$, end time $T$, and time
    increment $h$.
Output: Shape functions $F^s_i$, and scaling constants $s_i$ for $i = 1, 2, \ldots, T/h$.
 1: $F^s_0 := ComputeInit(\psi, Grid(\Omega))$
 2: $s_0 := 1$
 3: $\mathbb{L} := ComputeTensorOperator(\mathfrak{L}, Grid(\Omega))$
 4: $\mathbb{B} =: \mathbb{I} + \mathbb{L}h$
 5: $\mathbb{A} := ComputeA(\mathbb{B}, \psi, Grid(\Omega))$
 6: for $i = 1, 2, \ldots, T/h$ do
 7:     $F'_{i-1} := UpdateBoundary(F^s_{i-1}, \psi)$
 8:     $F'_i := \mathbb{A}F'_{i-1}$
 9:     $s_i := s_{i-1}F'_i(0)$
10:     $F^s_i := F'_i / s_i$
11: end for
```

if the controlled system is stable. Methods to choose these functions are not the main focus of this thesis. Refer to Jadbabaie's PhD thesis [107] for a more detailed discussions.

Figure 5.2 shows an implementation of the controller produced by Algorithm 4 with scaling on a simple pendulum stabilization example. The goal is to stabilize the pendulum upright, where the tilt angle $x_1 = 0$ is zero. For more detailed descriptions of this example (Example A.2) and comparison of Algorithms 3 and 4 for finite horizon problem, refer to Appendix A.

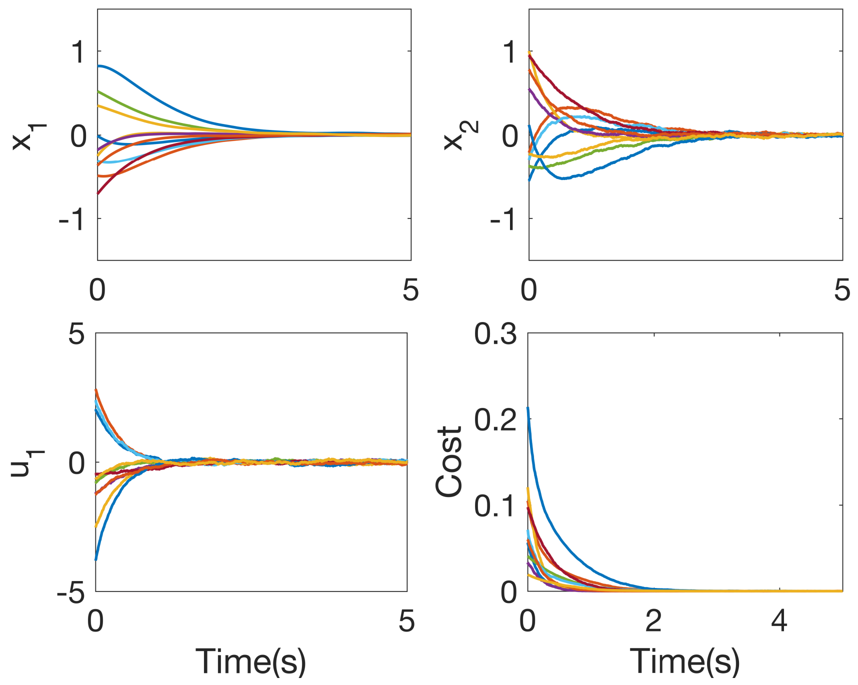


Figure 5.2: State, input, and cost trajectories for 10 random trials of a simple pendulum stabilization using the controller from Algorithm 4 with scaling. All states and inputs converge to the origin by $T = 5$ seconds.

5.5 Controller Synthesis for Infinite Horizon Problem

The infinite horizon problem (3.10c) is an eigenvalue problem, where c is the smallest eigenvalue and Ψ is the corresponding eigenfunction. As a result, (5.3c) is an eigenvalue problem in tensor form, where F is the eigentensor for eigenvalue c. A well known iterative technique that computes the smallest eigenvalue and the associated eigenvector of a matrix is the inverse power iteration [108]. The infinite horizon problem (3.10c) is solved using a similar iterative algorithm summarized in Algorithm 6. This algorithm is equivalent to the inverse power iteration for a matrix

Algorithm 6 Infinite Horizon Problem

```
Input: Linear operator $\mathfrak{L}$, boundary conditions $\psi$, domain $\Omega$, initial tensor function
    guess $F_0$, accuracy tolerance $\varepsilon$, and initial rank $r_0$.
Output: Eigentensor $F'$, eigenvalue $c'$
1: $\mathbb{L} := ComputeTensorOperator(\mathfrak{L}, Grid(\Omega))$
2: $\mathbb{A} := ComputeA(\mathbb{L}, \psi, Grid(\Omega))$
3: $F := 0, F' := F_0, c := 1, c' := 0$
4: while $|c - c'| > \varepsilon$ do
5:     $c := c', F := F'$
6:     $F' := ALS(-\mathbb{A}, F, \varepsilon)$
7:     $c' := 1/\|F'\|$
8:     $F' := c'F'$
9: end while
```

when the system is one dimensional.

The first two steps of Algorithm 6 also appear in Algorithms 2–4. Algorithm 6 begins to differs starting from line 3, where it computes an eigenvalue and eigenfunction pair for the tensor operator using inverse power iteration. At each iteration, F' is computed by solving $-\mathbb{A}F' = F$ using Algorithm 1. Then, F' is normalized and the process repeats. When the changes in F' is small, the iteration terminates. The last F' is an eigenfunction, and the last c' is the eigenvalue of the tensor operator $-\mathbb{A}$. The choice of initial tensor function F_0 guess determines the convergence rate.

Figure 5.3 shows an implementation of the controller produced by Algorithm 6 on a VTOL aircraft example. The goal of the VTOL is to stabilize at the origin. For more detailed descriptions of this example (Example A.4) and other examples of infinite horizon problem, refer to Appendix A.

5.6 Issues of ALS and Sequential Alternating Least Squares (SeALS)

The success of ALS algorithm (Algorithm 1) in computing a solution is essential for synthesizing a controller as seen in Algorithms 2–4 and 6. However, in many cases, Algorithm 1 may fail due to bad numerical conditioning. This section begins with describing an example that is used to illustrate the main points of this section. Then, the issue of Algorithm 1 is introduced followed by descriptions of the techniques to mitigate the issue. Lastly, a new algorithm (Algorithm 7) is introduced in which the techniques to alleviate ill conditioning of Algorithm 1 are included.

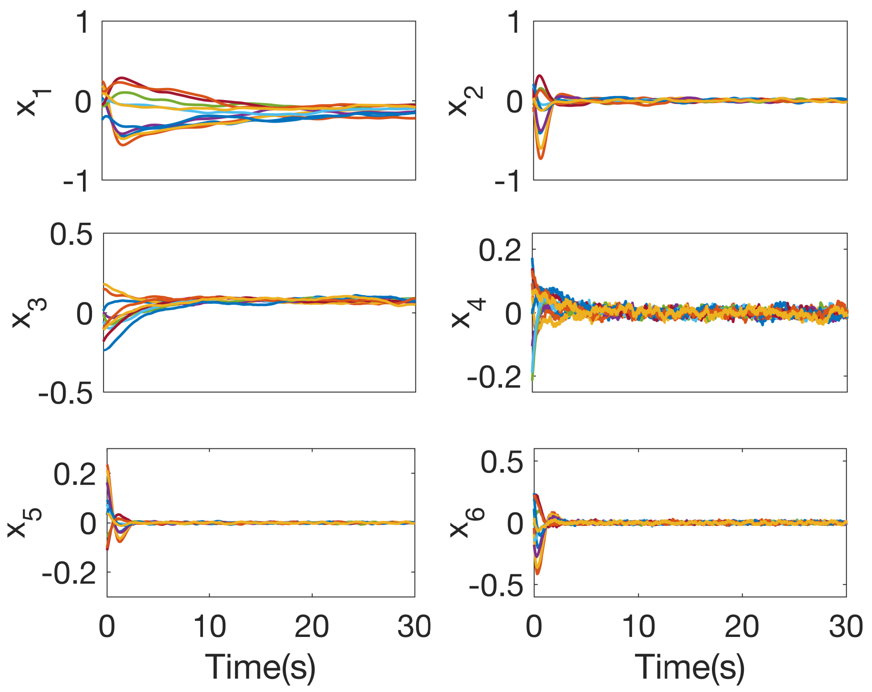


Figure 5.3: State trajectories for 10 random trials of a VTOL aircraft using the controller from Algorithm 6. All states stabilize to zero.

5.6.1 An Illustrative Example

We first consider a two-dimensional polynomial system with the first exit cost functionals (3.3), whereby the solution can be computed for comparison as shown in Section 4.8.

Example 5.1 (Two Dimensional Polynomial System)**.** The dynamics of this system is given by

$$dx_1 = (2(x_1^5 - x_1^3 - x_1 + x_1 x_2^4) + x_1 u_1)\, dt + d\omega_1$$
$$dx_2 = (2(x_2^5 - x_2^3 - x_2 + x_2 x_1^4) + x_2 u_2)\, dt + d\omega_2$$

on the domain $\Omega = \{(x_1, x_2) | -1 \leq x_1 \leq 1, -1 \leq x_2 \leq 1\}$. The noise is set to $\Sigma_\varepsilon = I$. The cost functions are set to $q(x) = x_1^2 + x_2^2$ and $R = 2I$, to represent that the goal is to reach the origin. The boundary conditions at $x_1 = \pm 1$ and $x_2 = \pm 1$ are set to $\phi(x_1, x_2) = 5$, and the boundary condition at the origin is set to $\phi(0, 0) = 0$. We use uniform finite difference scheme for discretization with a sixth order of accuracy for the derivatives.

A MacBook Pro with 2.5 GHz i5 processor and 4 GB ram-memory is used to perform the computation in MATLAB.

5.6.2 Ill-Conditioning of ALS

When solving for the solution of (5.3a), the associated matrix $\mathcal{M}$ in (3.21) often becomes ill-conditioned. In other words, κ the condition number of $\mathcal{M}$ exceeds 10^{13}, where $\kappa = \frac{\sigma_{max}}{\sigma_{min}}$, and σ_{max} and σ_{min} are the maximal and minimal singular value of $\mathcal{M}$ respectively. A poorly conditioned linear equation will result in an inaccurate estimation of the solution, independent of the algorithm used to find the solution. Therefore, the accuracy of the solution F in (3.21) will be limited by the condition number of the matrix $\mathcal{M}$ [109]. This effect may prevent ALS from iterating further to produce a lower residual, resulting in a less accurate solution.

We identify two sources that cause $\mathcal{M}$ to be ill-conditioned, the operator $\mathbb{A}$ and the solution F. According to (3.22), $M_{i,j}$ is a function of both $\mathbb{A}$ and F. The majority of the matrices in $\mathbb{A}$ originate from discretizing the differential operators in $\mathfrak{L}$ using finite differencing, which tend to have coefficients with large magnitude. Particularly, elements in $\mathbb{A}$ that are near the boundary $\partial\Omega$ tend to have large values because of finite differencing on the edges. However, other parts of $\mathbb{A}$ that corresponds to the boundary condition tends to be small because they consists of identity operators. As a result, the matrix $\mathcal{M}$ that contains the tensor terms of $\mathbb{A}$ usually has a high condition number.

Furthermore, the magnitude difference among the tensor terms of the most currently computed solution F generally increases as its rank increases. New tensor terms that are added by the algorithm are generally smaller than previous tensor terms in F because the new terms are added to account for the residual of the previous tensor terms. A regularizer added to the cost function, as described in Section 3.8.2, prevents ill-conditioning of the solution F. However, if the regularizer α is too large, the resulting solution F gives a large residual. Recall that the residual of the current solution is given by

$$\frac{||\mathbb{A}F - G||}{\sqrt{\prod_{i=1}^{d} n_i}}.$$

Therefore, limited by a moderate α to prevent a large residual, different terms $M_{i,j}$ in $\mathcal{M}$ that contain different tensor terms of F often have large magnitude differences, causing $\mathcal{M}$ to be ill-conditioned. Figure 5.4 illustrates an example of an ill-conditioned matrix $\mathcal{M}$ for Example 5.1 that shows the effects from both $\mathbb{A}$ and F.

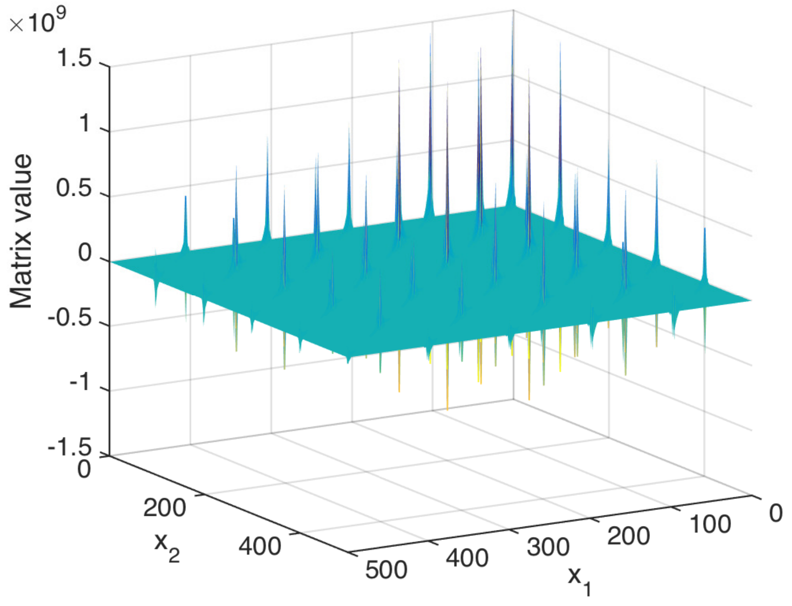


Figure 5.4: An example of an ill-conditioned matrix $\mathcal{M}$ in (3.21) for Example 5.1. The axes x_1 and x_2 denote the indices of the matrix $\mathcal{M}$, and the vertical axis is the value of $\mathcal{M}$. The peaks originate from the operator $\mathbb{A}$ and the solution F. The condition number of this matrix is 1.02×10^{13}.

5.6.3 Improvements on ALS

To mitigate the ill-conditioning issue in Algorithm 1, we introduce two new improvements.

Sequential Computation of Solution

The linearity of (3.20) can be used to reduce the magnitude differences among the elements of $\mathcal{M}$ in (3.22) originating from the current solution F. The key idea is to subtract dominant tensor terms of F in equation (3.20) whenever $\mathcal{M}$ becomes ill-conditioned, and keep iterating with the remaining smaller terms. Large magnitude differences in F are therefore avoided, allowing the algorithm to realize a lower residual.

More precisely, note that

$$G = \mathbb{A}F = \mathbb{A}\sum_{l=1}^{r_F} F^l = \sum_{l=1}^{r_F} \mathbb{A}F^l \Longleftrightarrow G - \sum_{l=1}^{p-1} \mathbb{A}F^l = \sum_{l=p}^{r_F} \mathbb{A}F^l = \mathbb{A}\sum_{l=p}^{r_F} F^l,$$

where F^l represents the l-th tensor term in the tensor function F. Let $F_p = \sum_{l=p}^{r_F} F^l$

and $G_p = G - \sum_{l=1}^{p-1} \mathbb{A}F^l$. Then, the previous equation becomes

$$\mathbb{A}F_p = G_p,$$

which is a linear equation of the form (3.20) that Algorithm 1 can solve. Intuitively, this technique removes the dominant terms in F from the equation allowing the algorithm to compute other smaller terms in F, avoiding ill-conditioning of $\mathcal{M}$. Thus, if the current computed solution $\hat{F}$ causes $\mathcal{M}$ to be ill-conditioned, we record the $\hat{F}$ as F_j, reset the G to $G - \mathbb{A}\hat{F}$, and restart the algorithm using the new G. As a result, we obtain a sequence of solutions F_j in which the sum of the sequence returns the full approximate solution F.

Figure 5.5 shows the performance of the ALS with sequential solutions for Example

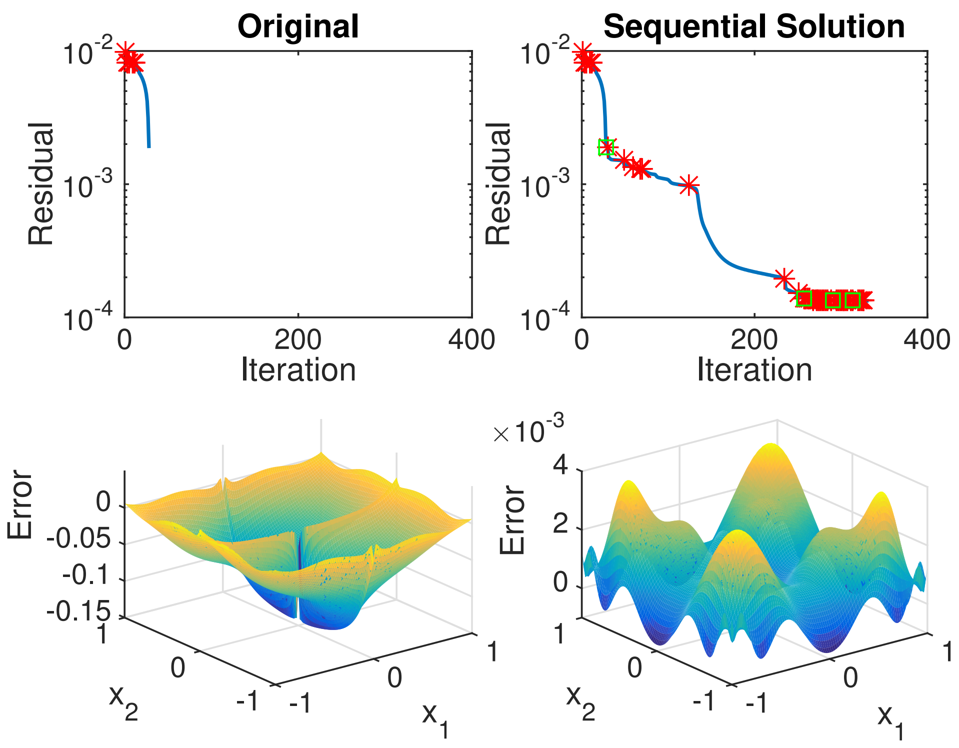


Figure 5.5: Computational results of using Algorithm 1 on Example 5.1 with and without sequential solutions. Each dimension has $n_i = 101$ discretization points. The first row shows the residual $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} n_i}}$. decreases in each iteration, and the second row plots the error between the computed solution and the true solution. A red star indicates when a new tensor term is added. The solution F is reset at a green square. The modified ALS achieves a more accurate solution. The original ALS quits when $\mathcal{M}$ is ill-conditioned. The algorithm with sequential solutions is terminated when the residual stops improving.

5.1. The modified ALS realizes a lower residual $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} n_i}}$, achieving a more accurate solution. The original ALS quits when $\mathcal{M}$ is ill-conditioned. The residual of the ALS with sequential computation stagnates when the computation reaches the MATLAB precision. At every reset of F, G is reset by subtracting $\hat{F}$ from the current G. Each subtraction reduces the magnitude of G, and eventually, the magnitude of G becomes so small that the magnitude difference between $\mathbb{A}$ and G is on the order of 10^{13}, approximately the MATLAB precision. In this case, the algorithm can no longer compute a high accuracy solution.

Boundary Condition Rescaling

In addition to the sequence of subtractions, the linearity of (3.20) allows for rescaling of elements in $\mathcal{M}$ to decrease its condition number. Rescaling reduces the magnitude differences resulting from the operator $\mathbb{A}$ at the start of Algorithm 1 (before line 1). Specifically, the elements in A_i^l that correspond to the boundary conditions are rescaled so that the overall condition number of $\mathcal{M}$ is decreased. The specific rescaling constants can be any constants that effectively decrease the condition number of $\mathcal{M}$. For example, the rescaling constant in dimension i can be computed from the weighted summation of the matrices $\{A_i^l\}_{l=1}^{r_A}$

$$s_i = \frac{\sum_{l=1}^{r_A} A_i^l \left\|A_i^l\right\|}{\sum_{l=1}^{r_A} \left\|A_i^l\right\|}$$

to rescale boundary conditions in dimension i to approximately the same magnitude as the elements not corresponding to the boundary conditions. Note that G is rescaled to ensure that (3.20) still holds. Different elements in A_i^l can have different rescaling constants.

More precisely, to scale the Dirichlet boundary conditions of $\partial\Omega$ in the k-th dimension, the first and last diagonal elements of the matrix C in (5.5) are scaled with s_k. Then, the first and last elements of C in (5.4) are scaled with s_k in the k-th dimension to ensure that the scaled terms A and G still satisfy the equation $\mathbb{A}F = G$. For Neumann and periodic boundary conditions, the rescaling method essentially follows the same idea.

The performance of this boundary rescaling technique is demonstrated with Example 5.1 and the result is shown in Figure 5.6. By rescaling the boundary conditions, the modified ALS algorithm can solve to a lower residual $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} n_i}}$, achieving a more accurate solution (see second row of Figure 5.6) before $\mathcal{M}$ becomes ill-conditioned. Both runs quit once $\mathcal{M}$ becomes ill-conditioned.

Furthermore, Figure 5.7 shows that sequential computation combined with boundary condition rescaling produces a more accurate solution and decreases computation time dramatically from about 77 minutes (580 iterations) to about 7 minutes (190 iterations) before the residuals stabilized. The residuals for both cases stabilize once the computations reach MATLAB precision. The separation rank of the computed solutions are 17 and 126 for algorithm with and without operator rescaling respectively. Note the residual is lower for the algorithm without boundary condition rescaling (see first row of Figure 5.7), but the solution is more accurate for the algorithm with boundary condition scaling (see second row of Figure 5.7). This observation is not unexpected because the residual is calculated after the boundary

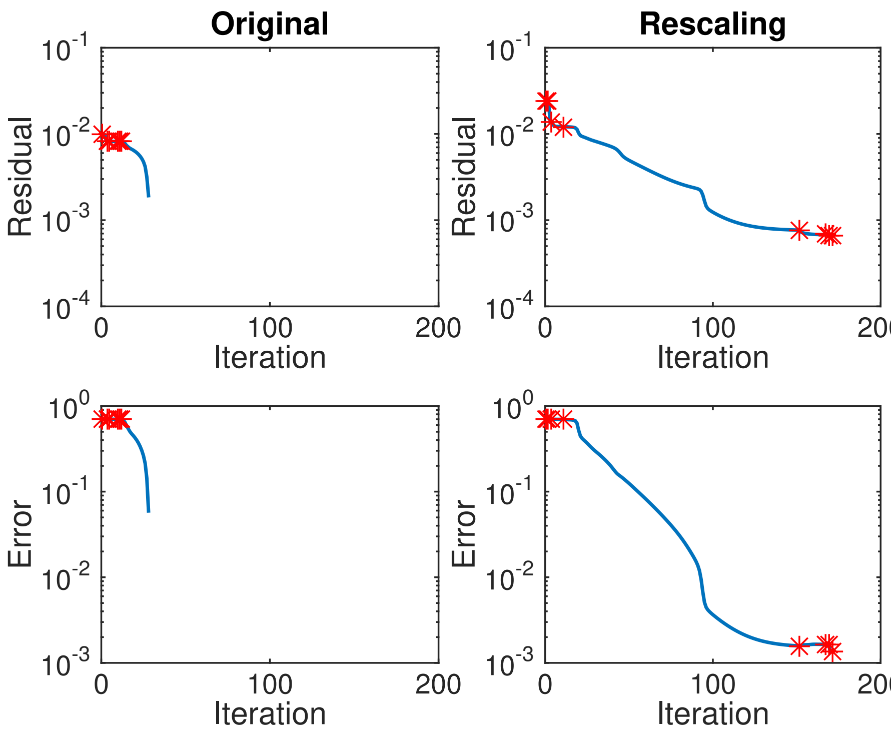


Figure 5.6: Computational results of using Algorithm 1 on Example 5.1 with and without boundary condition rescaling. Each dimension has $n_i = 101$ discretization points. The first row shows that the residual $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} n_i}}$ decreases in each iteration, and the second row plots the mean squared error between the computed solution and the true solution in each iteration. A red star indicates when a new tensor term is added. Boundary condition rescaling yields a lower residual before $\mathcal{M}$ becomes ill-conditioned, and thus achieves a more accurate solution. Both runs quit once $\mathcal{M}$ becomes ill-conditioned.

conditions are rescaled. Thus, when comparing the two, one should consider that the residuals are not exactly equivalent.

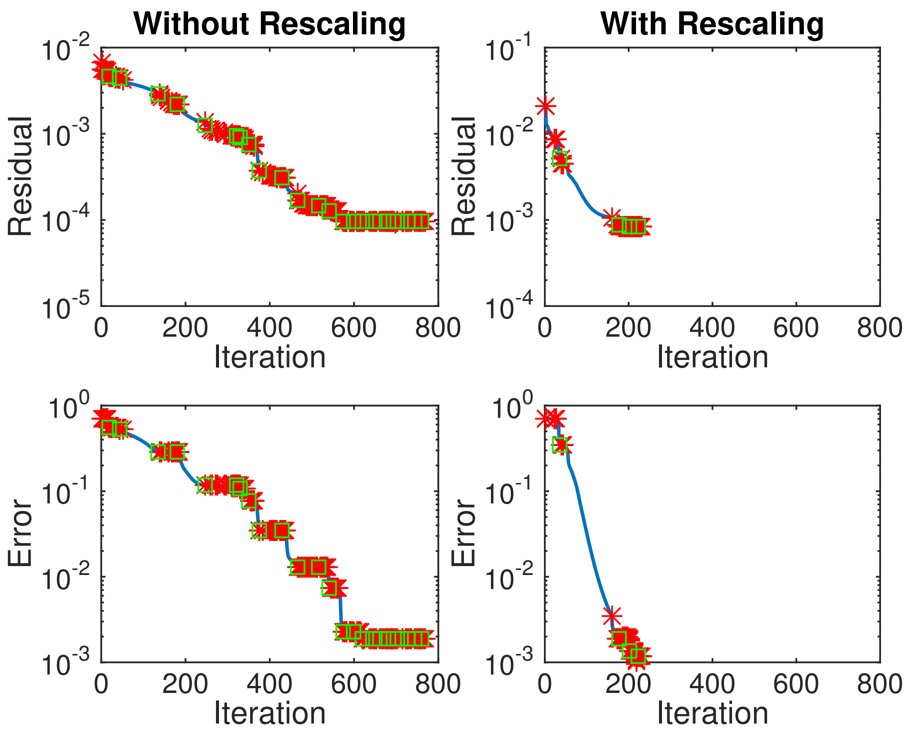


Figure 5.7: Computational results of implementing sequential computation on Example 5.1 with and without boundary condition rescaling. We chose $n_i = 151$ discretization points for each dimension to distinguish the two cases better. The first row shows how the residual $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} n_i}}$ decreases with each iteration, and the second row shows the mean squared error between the computed solution and the true solution with each iteration.

5.6.4 Sequential Alternating Least Squares (SeALS)

Combining the sequential solution and boundary condition rescaling, we arrive at the SeALS algorithm (Algorithm 7), which can compute more accurate solutions for the linear HJB equation in a shorter amount of time.

Given a tensor operator $\mathbb{A}$ and a tensor function G computed by *ComputeA* and *ComputeG* in Algorithm 2, SeALS implements the original ALS algorithm with two additional steps. First, before solving (3.21), elements in $\mathbb{A}$ and G that correspond to the boundary conditions are rescaled by *RescaleTensors* as described in previous section. Second, when $\mathcal{M}$ is ill-conditioned, the current F is recorded as F_j. Then G is reset to $G - \mathbb{A}F$, and F is reset to a preconditioned random rank 1 tensor term.

In the end, the algorithm returns F, which is the sum of all previously recorded F_j.

This SeALS algorithm is implemented in MATLAB [32]. A more detailed description and other additional features of SeALS are available in the tool's user's guide [110].

The first column of Figure 5.8 shows the solution obtained by Algorithm 2 and SeALS for Example 5.1. The error from the true solution is on the similar order of magnitude, 10^{-3}, as the uncertainty of the true solution. The solution is therefore considered accurate. The computation was performed with $n_i = 151$ discretization points in each dimension. A uniform finite difference scheme with sixth order accuracy derivatives is used for discretization. The residual tolerance for compressing the operator was set to 10^{-6} decreasing the rank from 73 to 6. The algorithm is terminated when the residual stagnates. The first five basis functions $\{F_i^l\}_{l=1}^5$ in each dimension are shown on the second column of Figure 5.8. The functions are smooth with peaks at the origin due to the zero-cost. Residual convergence and

Algorithm 7 Sequential ALS (*SeALS*) Algorithm

```
Input: Tensor operator $\mathbb{A}$, tensor function $G$, accuracy tolerance $\varepsilon$, scaling constants
    $a$, initial rank $r_0$
Output: Tensor function $F$
 1: $F := RandomTensor(r_0)$
 2: $\mathbb{A}, G := RescaleTensors(\mathbb{A}, G, a)$
 3: $res := ComputeResidual(\mathbb{A}, F, G)$
 4: $j := 1$
 5: while $res > \varepsilon$ do
 6:     $res' := res$
 7:     for $k = 1, 2, \ldots, d$ do
 8:         $F, l_k := SolveNormal(\mathbb{A}, F, G)$
 9:     end for
10:     $res := ComputeResidual(\mathbb{A}, F, G)$
11:     if $l_k$ is True for any $k \in [d]_+$ then
12:         $F_j := F$ and $j := j + 1$
13:         $G := G - \mathbb{A}F$
14:         $F := RandomTensor(1)$
15:     else if $|res - res'| < \varepsilon$ then
16:         $F^r := PreRandomTensor(1)$
17:         $F := F + F^r$
18:     end if
19: end while
20: $F := \sum_j F_j$
```

the normalization constants s_l^F is shown in Figure 5.9. The residual stabilizes in the end when the computation reaches the MATLAB precision. The algorithm is thus terminated. The rank of the solution before the stagnant residual is around 10, and the total iteration time is 72 seconds. Figure 5.10 shows ten simulations of the system controlled using the controller produced from Algorithm 2 and SeALS. All trajectories reach the origin as expected.

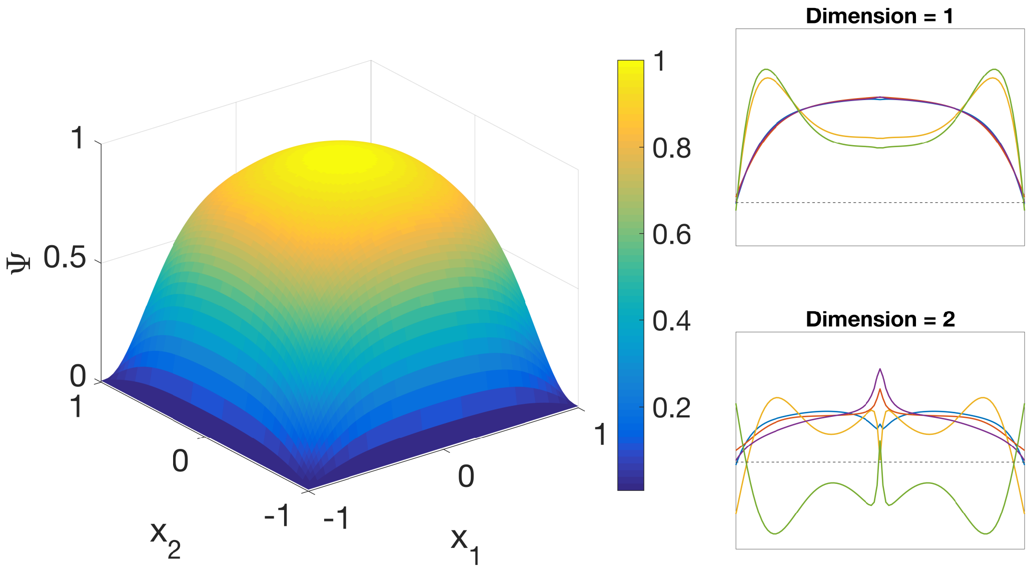


Figure 5.8: Computed solution (left) and the basis functions in each dimension (right) for Example 5.1. The uncertainty of the true solution is of magnitude 10^{-3}, the same magnitude as the error. The obtained solution is therefore considered accurate.

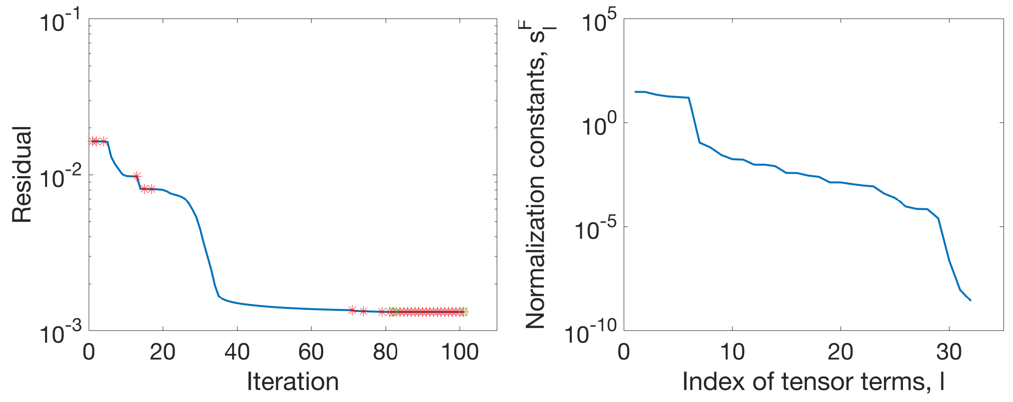


Figure 5.9: Residual $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} n_i}}$ (left) and the normalization constants s_l^F (right) for Example 5.1. The algorithm is terminated when the residual stagnates.

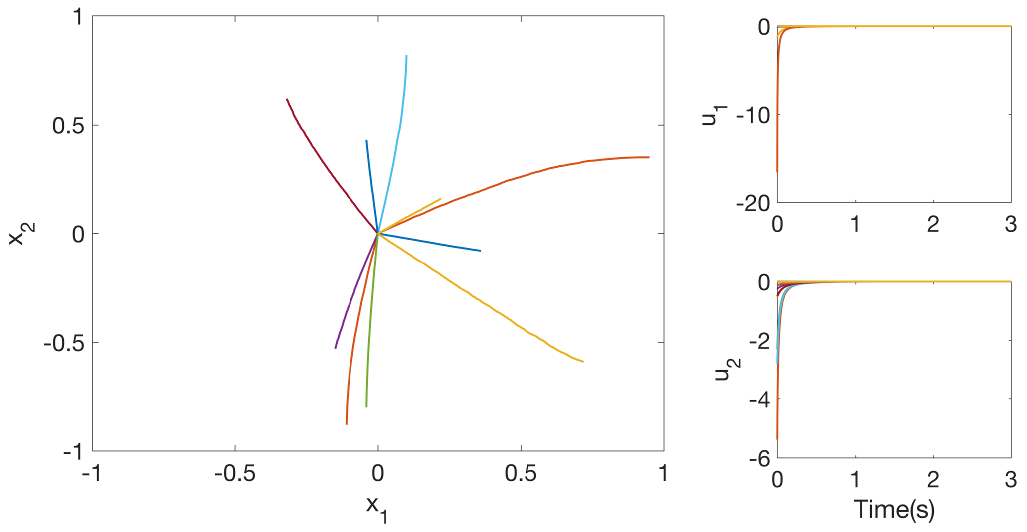


Figure 5.10: Ten simulations of Example 5.1 controlled using the controller produced by Algorithm 2 and SeALS (Algorithm 7) and their corresponding control signals. All trajectories reach the origin as expected.

5.7 Comparison with SOS-based Approach

This section briefly compares the SOS based technique introduced in Chapter 4 with the low rank tensor decomposition based technique introduced in this chapter using Example 5.1. This example is chosen because it is a two dimensional example that the SOS based approach can also solve for the approximate solution as shown in Section 4.8.

Figure 5.11 shows that the controllers resulting from both techniques stabilize the system with noise in Example 5.1 for ten randomly chosen initial states. Figure 5.12 shows the corresponding control inputs. The degree of polynomial is 24 for the SOS based approach, and the noise is the same for both controllers. Note the differences between the two approaches for the control input trajectories. As a result of these differences, the cost for using the controller computed from the SOS based method is lower than the cost of using the controller computed from the tensor based method. Figure 5.13 shows the cost comparison using the two approaches for the ten simulations. The point on the top right corner correspond to the initial state at $(0.95, 0.35)$, which is close to the boundary of computation for both approaches. For the tensor based approach, the cost becomes higher because the solution becomes highly oscillatory near the boundary due to numerical artifact of imposing boundary conditions. For the SOS based approach, the solution is a smooth polynomial function throughout the entire domain.

However, the computation time for the SOS based controller is approximately 43 minutes for polynomial optimization with 24 degrees that results in an accuracy, ε, of 0.0025. If the entire hierarchy of SOS programs is considered, the computation takes approximately 1.5 hours. In contrast, the tensor based approach takes only 72 seconds to achieve a residual of 0.0013. Nonetheless, the SOS based approach provide strong performance and stability guarantees that the tensor based approach does not have.

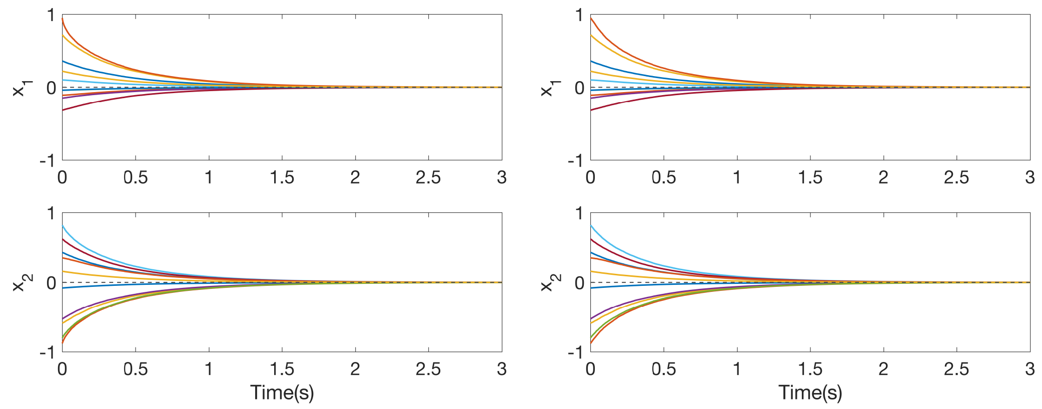


Figure 5.11: The state trajectories of ten simulations of Example 5.1 using the controllers computed from SOS programs (right) in Chapter 4 and tensor based Algorithm 2 with SeALS (Algorithm 7) (left). The noise is the same for both controllers. All trajectories reach the origin as expected.

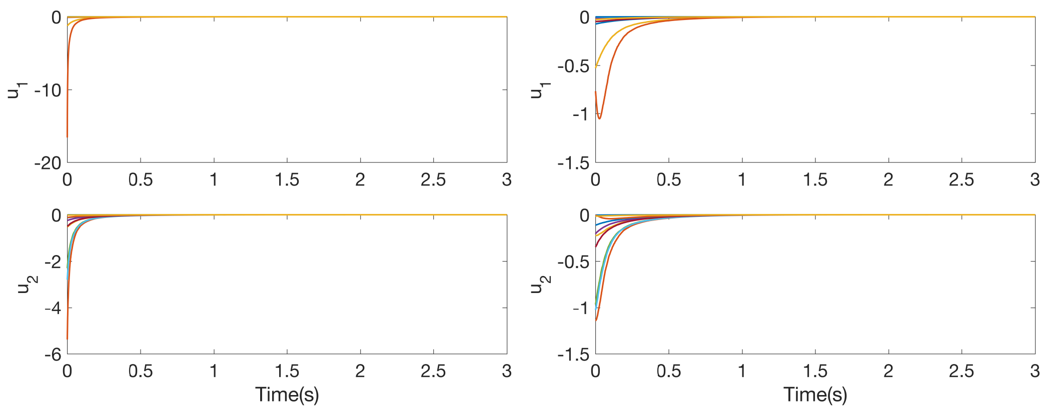


Figure 5.12: The control input trajectories of ten simulations of Example 5.1 using the controllers computed from SOS programs (right) in Chapter 4 and tensor based Algorithm 2 with SeALS (Algorithm 7) (left). The noise is the same for both controllers.

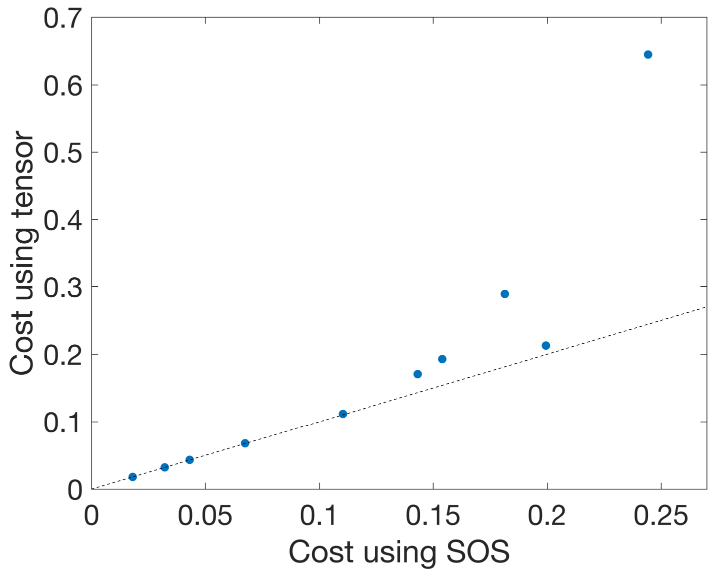


Figure 5.13: The cost comparison of simulations of Example 5.1 using the controllers computed from SOS programs in Chapter 4 and tensor based Algorithm 2 with SeALS (Algorithm 7). Each point is one simulation with the same noise for both controllers.

5.8 Summary

This chapter presents a method to solve the high dimensional linear HJB equation for three classes of cost functionals and a large class of stochastic affine nonlinear dynamical systems. This work significantly improves upon [95], where the concept of tensor decomposition and ALS are considered for solving the linear HJB equation for the first exit problem. The original framework's ill-conditioning issue is mitigated through sequential computation of solutions and boundary condition rescaling. Consequently, the new algorithm that incorporates both methods achieves significantly lower error compared to the original implementation, resulting in more accurate solutions and better controllers. Furthermore, this technique is extended into finite horizon and infinite horizon problems by combining time stepping and inverse power iteration with the ALS algorithm. A MATLAB tool that implements the algorithms discussed in this chapter is developed [32]. A few engineering examples, including stabilizing a quadcopter, are presented in Appendix A to illustrate the performance of the low rank tensor decomposition based method. The ability to compute the solution to the linear HJB equation for a quadcopter with twelve dimensions using a personal laptop and produce a controller that achieves the objective is a strong indicator of this technique's great potential for implementation in other robotics and engineering systems.

Future work includes optimizing the MATLAB tool for speed, and incorporating other numerical techniques that improve the algorithm's stability, including adding artificial diffusion [111]. Different kinds of differentiation schemes such as upwind finite difference scheme [112] and adaptive grid [99] could be incorporated for a more stable and accurate calculation. Including other forms of tensor representation as alternatives in the MATLAB tool such as tensor train [79] and function train [80] could be considered as well. Apart from ALS, many algorithms [113] based on low rank tensor decomposition could potentially solve the HJB equation, and thus a future direction is to explore these other techniques with respect to solving the linear HJB equation.

One main limitation of this technique is numerical phenomena associated with the desirability function. The desirability function is given by the exponential of the negative of the value function. Therefore, the desirability function could easily become a very small number for a modest value of the corresponding value function. As a result, the controller that is proportional to the inverse of the desirability function is extremely sensitive to any small inaccuracy in the derivatives of the desirability function. This numerical limitation could potentially be alleviated by intelligently scaling the desirability function (*e.g.,* scaling in Algorithm 5) or by working directly with the value function instead of the desirability function. The latter requires solving the nonlinear HJB equation, and one future extension of this work is to expand the use of low rank tensor decomposition to the nonlinear PDE setting.

Apart from numerical and algorithmic improvement, a future direction of this work is deriving analytical properties of the controller given by the method proposed in this chapter, and providing performance guarantees on the controller. This extension may necessitate tensor based algorithms that have useful convergence properties and error bounds. In particular, in order to obtain the trajectory cost bound, the approximate value function has to be a pointwise upper bound of the true value function. This requirement necessitates tensor based algorithms that can enforce inequality constraints – a challenging yet interesting research direction.

Alternatively, this tool can be used as a method to provide approximate basis functions for the sums of squares (SOS) based technique described in Chapter 4 to solve for the optimal controller in high dimensions. The number of monomials in the SOS technique grows exponentially with dimension. However, the SOS technique provides a performance guarantees that the tensor based technique does not provide. To benefit from both, the tensor based technique can provide the SOS technique with a smaller set of monomials, and the SOS technique can compute for a suboptimal controller with performance guarantees.

Apart from solving the HJB equation, this tool may be adapted to solve general linear PDEs in high dimensions. In particular, the Fokker-Planck equation [114] used for uncertainty propagation and state estimation of a high dimensional system is a naturally linear PDE that could benefit from the technique presented in this chapter. Other researchers have investigated this idea with successes in engineering applications such as orbital mechanics [115]–[118], and the SeALS algorithm could potentially improve the performance of these computations.

In addition to solving linear PDE and the HJB equation, the low rank tensor decomposition framework could potentially be applied to other types of Hamilton-Jacobi equations including the Hamilton-Jacobi-Isaacs equation [119] or more generally, other dynamic programming problems, such as reinforcement learning. Essentially, the low rank tensor decomposition based framework is an efficient representation for large scale objects, for example, a high dimensional complex function. Therefore, it can have a significant impact in alleviating the curse of dimensionally that is ubiquitous in many fields of study by providing efficient representations and computations.

Appendix A

IMPLEMENTATION OF CHAPTER 5'S ALGORITHMS

This appendix presents examples of the applications of Algorithms 2–4 and 6 to solve first exit, finite horizon, and infinite horizon problems respectively using examples including those from [95] such as Vertical Takeoff and Landing (VTOL) aircraft and quadcopter. This appendix begins with the descriptions of the dynamics for each example. The specific cost functional and its associated linear HJB equation and boundary conditions are described in the individual sections later. Following the dynamics descriptions are three sections that discuss the implementation of Algorithms 2–4 and 6, respectively, using the examples.

Remark. The computation time given does not indicate the best possible computation time of this technique, because the computer and the implementation are not fully optimized for speed.

A.1 Example List

This section summarizes the example dynamical systems that are used for the rest of this section.

Example A.1 (Linear System)**.** The dynamics of a four dimensional linear system is given by

$$dx = (Ax + Bu)\,dt + B d\omega$$
$$A = \begin{pmatrix} 0.1609 & -0.9014 & 0.0971 & -0.2241 \\ -0.6151 & 0.3481 & 1.3786 & 2.7022 \\ -0.2390 & 0.9480 & -0.9167 & 0.5202 \\ 0.6150 & -1.7299 & -1.0357 & -0.4597 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix},$$

where $x \in \mathbb{R}^4$ represents the states, and $u \in \mathbb{R}^3$ represents tha control inputs. The A matrix has eigenvalues $0.4084 \pm 2.0834i$, -0.0997, and -1.5845. Thus, the system is unstable without a controller.

Example A.2 (Simple Pendulum)**.** The dynamics of a simple pendulum is given by

$$\begin{aligned} dx_1 &= (x_2 + 0.01u)\,dt + d\omega \\ dx_2 &= (sin(x_1) + u)\,dt + d\omega, \end{aligned}$$

where x_1 is the angle of the pendulum, x_2 is the angular velocity.

Example A.3 (Inverted Pendulum on a Moving Cart)**.** The dynamics of an inverted pendulum on a moving cart adapted from [120] is given by

$$\begin{aligned} dx_1 &= x_2\,dt + d\omega \\ dx_2 &= \frac{\frac{g}{l}\sin(x_1) - \frac{1}{2}m_r x_2^2 \sin(2x_1) - \frac{m_r}{ml}\cos(x_1)u}{\frac{4}{3} - m_r\cos^2(x_1)}\,dt + d\omega, \end{aligned}$$

where x_1 is the angle from upright position of the pendulum, x_2 is the angular velocity, $g = 9.8m/s^2$ is gravitational acceleration, and $l = 0.5m$ is the length of the pendulum. The system has mass ratio $m_r = \frac{m}{m+M}$, where $m = 2kg$ is the pendulum mass and $M = 8kg$ is the cart mass.

Example A.4 (Vertical Takeoff and Landing (VTOL) Aircraft)**.** The dynamics of a VTOL aircraft (from [121]) in the translation plane (p_x, p_y) with tilt angle θ are given by

$$\begin{aligned} dp_x &= v_x\,dt \\ dv_x &= -\sin\theta\,(u_1\,dt + d\omega_1) + \varepsilon\cos\theta\,(u_2\,dt + d\omega_2) \\ dp_y &= v_y\,dt \\ dv_y &= (-g + \cos\theta)(u_1\,dt + d\omega_1) + \varepsilon\sin\theta\,(u_2\,dt + d\omega_2) \\ d\theta &= v_\theta\,dt \\ dv_\theta &= u_2\,dt + d\omega_2, \end{aligned}$$

where v_x, v_y, and v_θ are translational and angular velocities, g is the gravitational constant, $\varepsilon = 0.01$, u_1 is the main thrust, and u_2 is the angular moment. The ground is at $y = 0$. Let $x = \begin{pmatrix} p_x & v_x & p_y & v_y & \theta & v_\theta \end{pmatrix}$.

Example A.5 (Quadcopter)**.** The quadcopter (from [23]) moves in three-dimensional space (p_x, p_y, p_z) with orientation given by the yaw angle ψ, pitch angle θ and roll angle ϕ. The control inputs $u = \begin{pmatrix} \mu & \tau_\psi & \tau_\theta & \tau_\phi \end{pmatrix}^\mathrm{T}$ are the main thrust μ, the yawing moment τ_ψ, the pitching moment τ_θ and the rolling moment τ_ψ. The dynamics are given by (3.1) in which

$$f = \begin{pmatrix} v_x & v_y & v_z & v_\psi & v_\theta & v_\phi & 0 & 0 & -g & 0 & 0 & 0 \end{pmatrix}^\mathrm{T}$$

$$B = \begin{pmatrix} 0_{6\times1} & 0_{6\times3} \\ \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta & \\ \cos\phi\sin\theta\sin\psi - \cos\psi\sin\theta & 0_{3\times3} \\ \cos\theta\cos\phi & \\ 0_{3\times1} & I_{3\times3} \end{pmatrix},$$

where $x = \begin{pmatrix} p_x & p_y & p_z & \psi & \theta & \phi & v_x & v_y & v_z & v_\psi & v_\theta & v_\phi \end{pmatrix}^{\mathrm{T}}$, $0_{m\times n}$ represents a m by n zero matrix, $I_{m\times n}$ represents a m by n identity matrix, and g is the gravitational constant.

A.2 First Exit Problem

Algorithm 2 and SeALS (Algorithm 7) are demonstrated using three examples from [95]. Note that simulations are performed with the indicated noise, whereas simulations in [95] were noiseless. A MacBook Pro with 2.5 GHz i5 processor and 4 GB RAM is used in the inverted pendulum and quadcopter examples, and a quadcore computer with 3.0 Ghz i7 processor and 64 GB RAM is used in the VTOL aircraft example. The regularizer α is set to 10^{-12} for all numerical examples.

A.2.1 Inverted Pendulum on a Moving Cart

Consider Example A.3. The goal is to keep the pendulum upright. Let the goal region be $\Lambda = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \leq 0.18,\ |x_2| \leq 0.3\}$. Then, the domain is chosen to be $\Omega = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \leq \pi,\ |x_2| \leq 11\} \backslash \Lambda$ because the problem is formulated as a first exit problem. The cost functions are set to $q = 0.1x_1^2 + 0.05x_2^2$ and $R = 0.02$ so that the system will exit the domain near the origin. For the exterior boundary of Ω, the state x_1 has a periodic boundary condition, and $\phi(x_1, \pm 11) = 10$ for x_2. The states at the interior boundary of Ω is set to zero. The noise is set at $\Sigma_\varepsilon = 10I$.

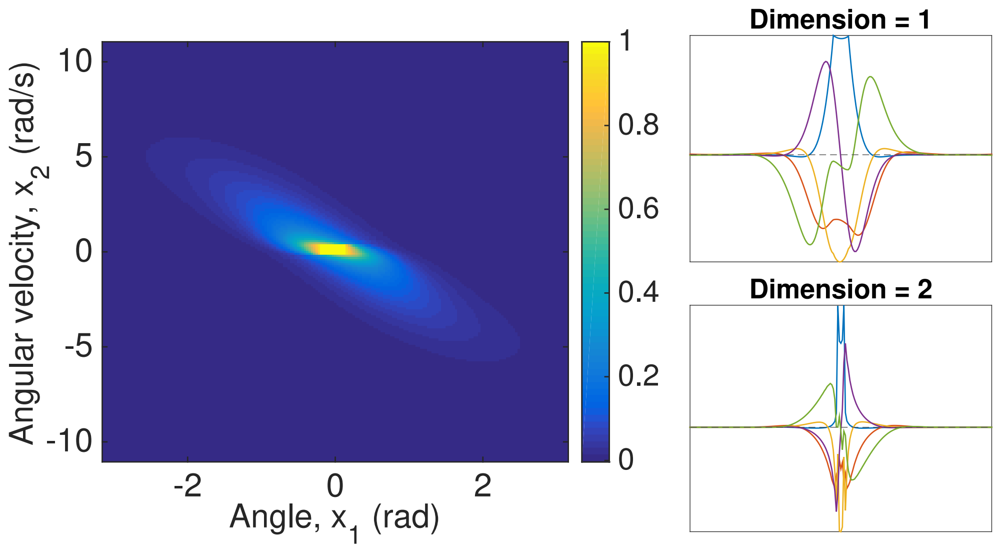


Figure A.1: The computed desirability function (left) and the basis functions $\{F_i^l\}_{l=1}^5$ in each dimension (right) for Example A.3.

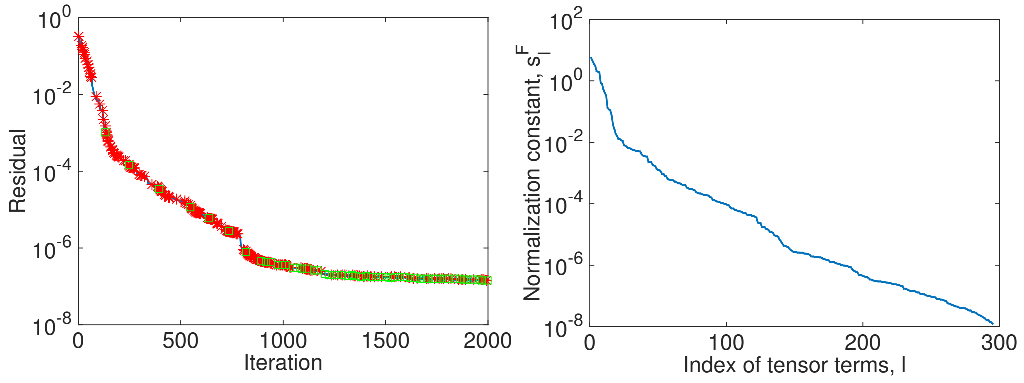


Figure A.2: Residual $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} n_i}}$ (left) and the normalization constants s_l^F (right) for Example A.3. The algorithm quits when it exceeds the prescribed maximum number of iterations 2000.

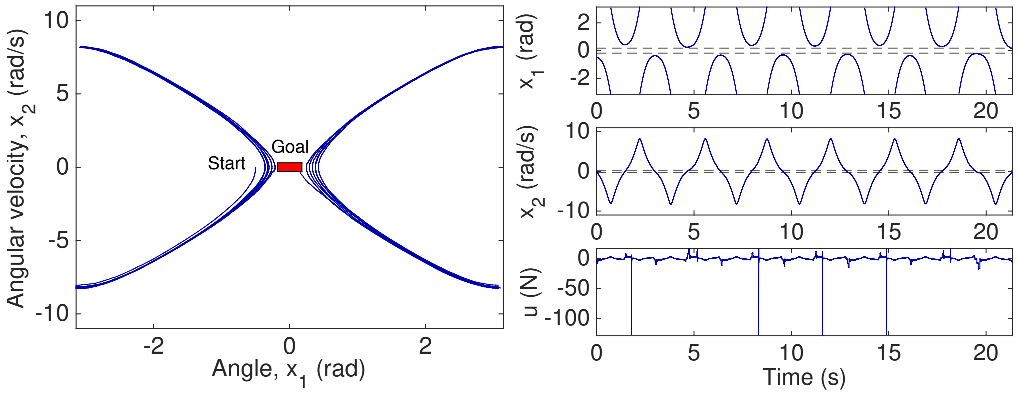


Figure A.3: A simulation of Example A.3 using the controller computed from the solution of Algorithm 2 and SeALS (Algorithm 7). Left plot shows the state trajectory in space, and right plots shows the state and control trajectories in time. The red box is the goal region. The angle x_1 is periodic.

A uniform finite difference scheme with sixth order accuracy derivatives is used for discretization. For each dimension, there are $n_i = 201$ discretization points. The operator was compressed from rank 29 to 11 for a relative residual 10^{-7}. Maximum rank of the solution was set to 20 for speed. Upon reaching the maximum rank, the algorithm records the current solution and restarts with a new rank one tensor (see Algorithm 7 line 12-14). Once the computation completes, the final solution is the sum of all recorded solutions. The final solution and the first five basis functions $\{F_i^l\}_{l=1}^5$ in each dimension are shown in Figure A.1.

Residual $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} n_i}}$ and normalization constants s_l^F are shown in Figure A.2. The algorithm quits with a separation rank of 230 when it exceeds a prescribed 2000 iterations, with a total run time of 37 hours, achieving a lower residual compared to [95]. Shorter runs can be achieved by optimizing the SeALS tool for speed or by quitting earlier when the basis weights are small enough for the specific application. A simulation is shown in Figure A.3 in which the pendulum reaches the goal region as desired when using a controller computed by SeALS.

A.2.2 Vertical Takeoff and Landing (VTOL) Aircraft

Next, consider Example A.4, which has six state dimensions. The domain Ω is defined by $p_x \in [-4, 4]$, $p_y \in [0, 2]$, $v_x \in [-8, 8]$, $v_y \in [-1, 1]$, $v_\theta \in [-5, 5]$, and a periodic $\theta \in [-\pi, \pi]$. The cost functions are $q(x) = 1$ and $R = 2I$.

The goal is to land the aircraft at $y = 0$, and the aircraft should reach $p_y = 0$ with moderate velocities v_x, v_y and v_θ and deviations in p_x and θ. Therefore, boundary conditions $\Psi|_{\partial\Omega} = 0$ are imposed for all non-periodic states except $p_y = 0$. At the i-th dimension, let S_i be the map from a state in the domain at the i-th dimension to a normalized state in the range of -1 and 1 such that the position order of the state

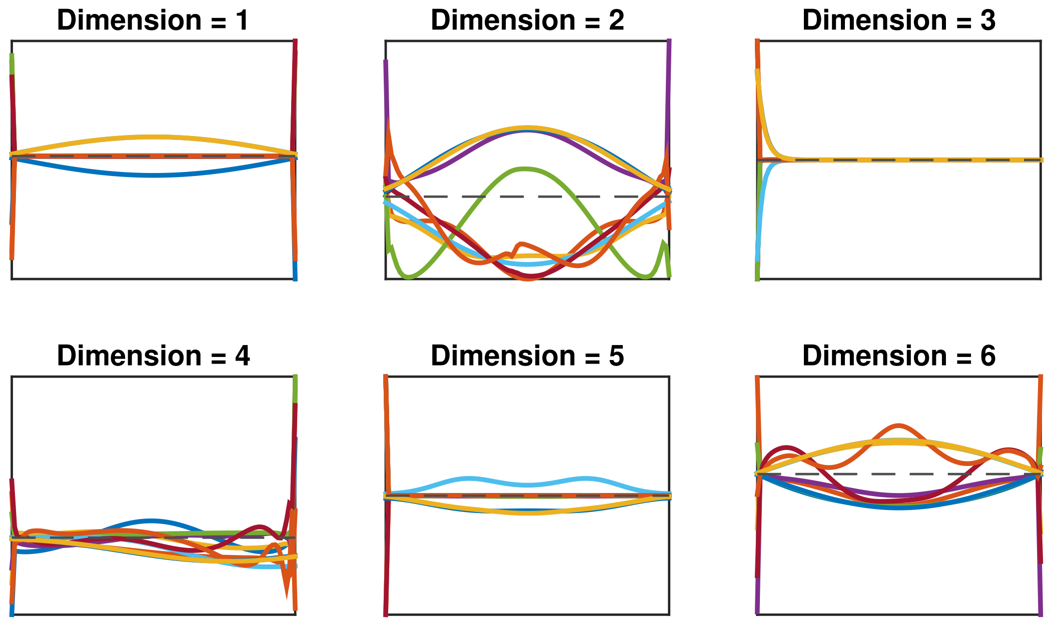


Figure A.4: The first ten basis functions in each dimension for Example A.4.

in the domain is preserved. More precisely, given x_i, the state at the i-th dimension,

$$S_i(x_i) = -1 + 2\frac{x_i - x_i^{min}}{x_i^{max} - x_i^{min}}$$

, where x_i^{min} and x_i^{max} are the boundary values of the i-th dimension. Then, at $p_y = 0$, $\Psi|_{p_y=0}(x) = \prod_{i=1}^{d}(1 - S_i(x_i)^2)$. The noise is set at $\Sigma_\varepsilon = 3I$.

A uniform finite difference scheme with second order accuracy derivatives is used for discretization. For each dimension, there are $n_i = 100$ discretization points. The operator was compressed from rank 50 to 13 for a relative error 10^{-5}. The algorithm took 30 hours, and the separation rank is 134. Figure A.4 presents the first ten basis functions $\{F_i^l\}_{l=1}^{10}$ in each dimension. Figure A.5 shows residual of the SeALS tool and the normalization constants s_l^F of the basis functions. Figure A.6 shows a simulation of the VTOL aircraft using controller computed from the solution produced by Algorithm 2 and SeALS (Algorithm 7). The controller successfully lands the aircraft ($p_y = 0$) with moderate speed and horizontal deviations, comparable to [95] despite the noise.

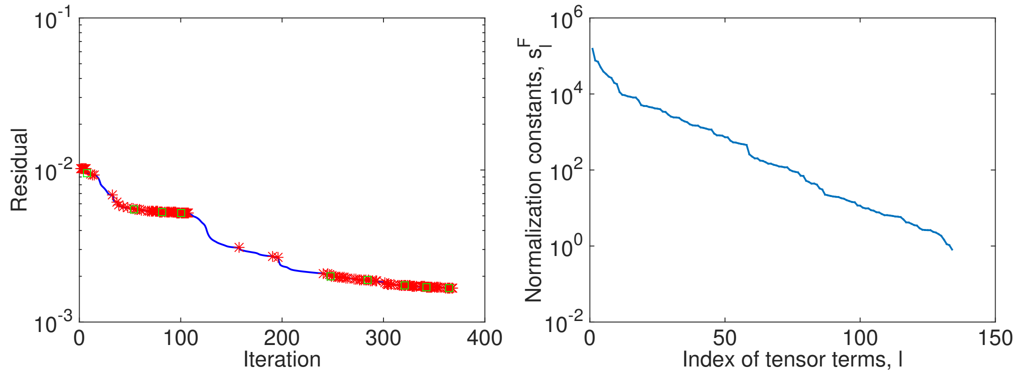


Figure A.5: Residual $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} n_i}}$ (left) and normalization constants s_l^F (right) for Example A.4.

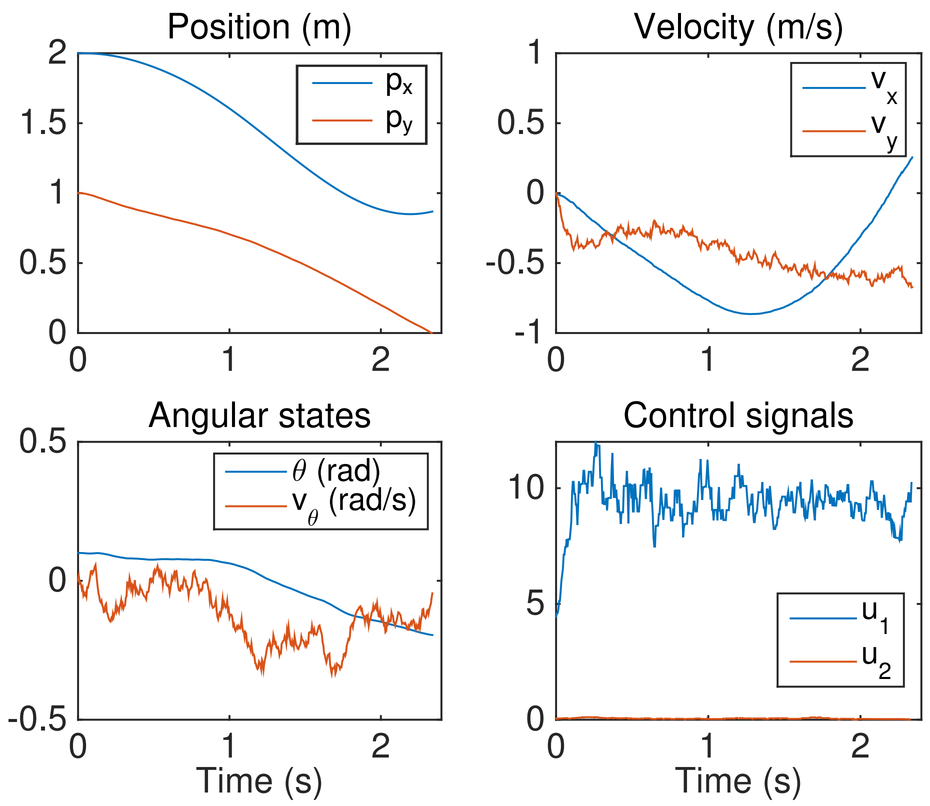


Figure A.6: A simulation of Example A.4 using the controller computed from the solution produced by Algorithm 2 and SeALS (Algorithm 7). The controller successfully lands the aircraft ($p_y = 0$) with moderate speed and horizontal deviations despite the presence of noise.

A.2.3 Quadcopter

Lastly, consider Example A.5 that has 12 state dimensions. The domain Ω is given by $p_x, p_y, p_z \in [-1, 1]$, periodic $\psi, \theta, \phi \in [-\pi, \pi]$, $v_x, v_y, v_z \in [-8, 8]$ and $v_\psi, v_\theta, v_\phi \in [-10\pi, 10\pi]$. The cost functions are $q = 2$ and $R = 2I$. The goal is to reach $p_x = 1$, and thus impose boundary conditions $\Psi|_{\partial\Omega} = 0$ except $p_x = 1$, where we set $\Psi|_{p_x=1}(x) = \prod_{i=1}^{d}(1 - S_i(x_i)^2)$. The noise is set at $\Sigma_\varepsilon = 100I$.

A uniform finite difference scheme with second order accuracy derivatives is used for discretization. For each dimension, there are $n_i = 100$ discretization points. The operator was compressed from rank 270 to 25 with a relative error 10^{-5}. The algorithm took 62 hours, and the separation rank is 52. Figure A.7 shows a sample simulation of the quadcopter controlled using the solution produced by Algorithm 2 and SeALS (Algorithm 7). The trajectory reaches $p_x = 1$ as desired, comparable to [95] despite the noise. The corresponding control signals are shown in Figure A.8.

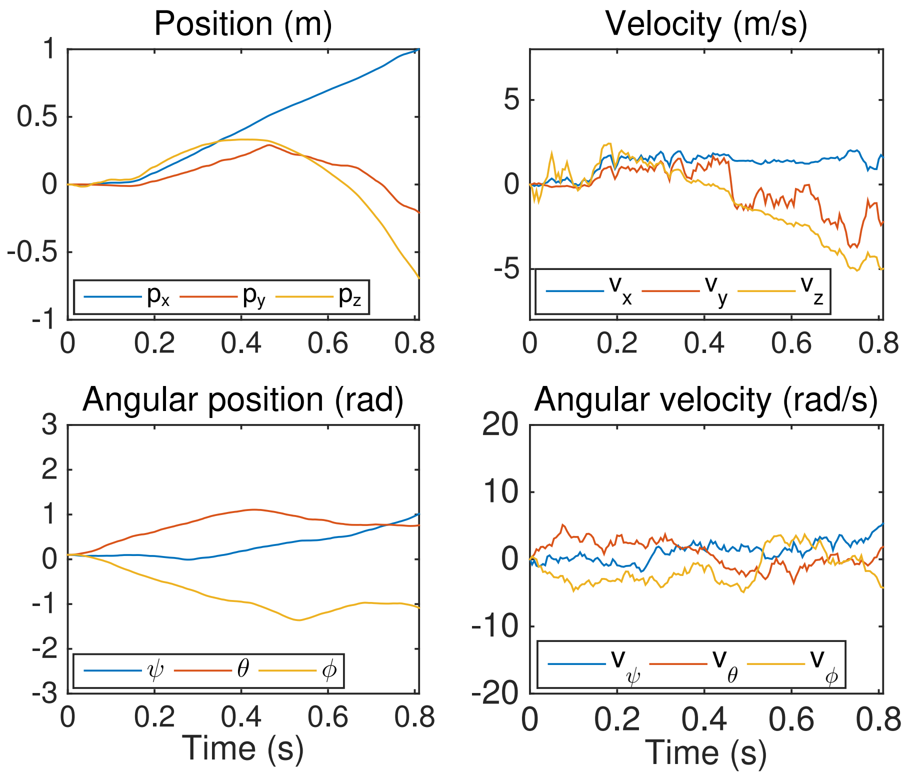


Figure A.7: A simulation of Example A.5 controlled using the controller produced by Algorithm 2 and SeALS (Algorithm 7). The quadcopter reaches $p_x = 1$ as desired. This figure is the same as Figure 5.1.

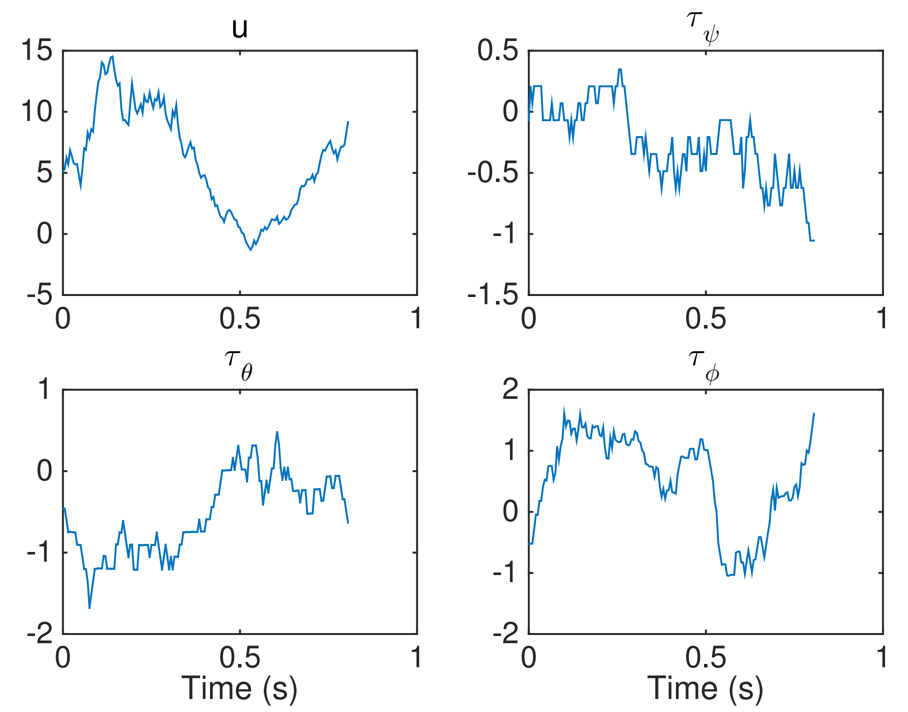


Figure A.8: The control signals for the simulation of Example A.5.

A.3 Finite Horizon Problem

Algorithms 3 and 4 with the scaling modification are demonstrated using Example A.2. Henceforth, when Algorithms 3 and 4 are mentioned, they refer to the algorithms with the scaling modification. A MacBook Pro with 2.9 GHz i5 dual-core processor and 16 GB RAM is used for all computations. The regularizer α is set to 10^{-12}. The goal is to move the unstable system to the origin within the given time $T = 5s$. The time integration is implemented with step size of 0.001 seconds. The domain is chosen to be $\Omega = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \leq \pi, |x_2| \leq 5\}$. The cost functions are set to $q = 0.1x_1^2 + 0.05x_2^2$ and $R = 0.02$. At the boundary of Ω, the state x_1 has a periodic boundary condition, and $\Psi(x, t)$ is set to zero when $x_2 = \pm 5$. The noise is set at $\Sigma_\varepsilon = I$. At $T = 5$ seconds, final time condition is $V(x, T) = 0.06x^\mathrm{T}x$ or equivalently, $\Psi(x, T) = e^{-3x^\mathrm{T}x}$. A spectral difference scheme is used for discretization. The domain is discretized into a 101 by 103 grid.

First, consider the results of using Algorithm 3. The operator $\mathbb{A}$ was compressed from rank 8 to 4 for a relative residual 10^{-6}. The total run time is 13 minutes. Figures A.11–A.13 shows the basis functions and the desirability function at three different times, namely $t = 0, 2.5,$ and 5 seconds. Figures A.10–A.13 show that the desirability function becomes smaller at earlier time because the cost to go (*i.e.,* the value function) is large when the remaining time from the end time $T = 5$ seconds is large. Figure A.9 gives a summary of the separation rank during each iteration of Algorithm 3. The separation rank grows as the computation progresses backward in time.

Simulation results are shown in Figure A.14 in which the system moves to the origin as desired when using a controller computed by Algorithm 3. Figure A.14 shows 10 simulations of the system with initial states randomly chosen and the corresponding input trajectories and the cost trajectories. The average finite horizon cost over the 10 simulations is 0.0271.

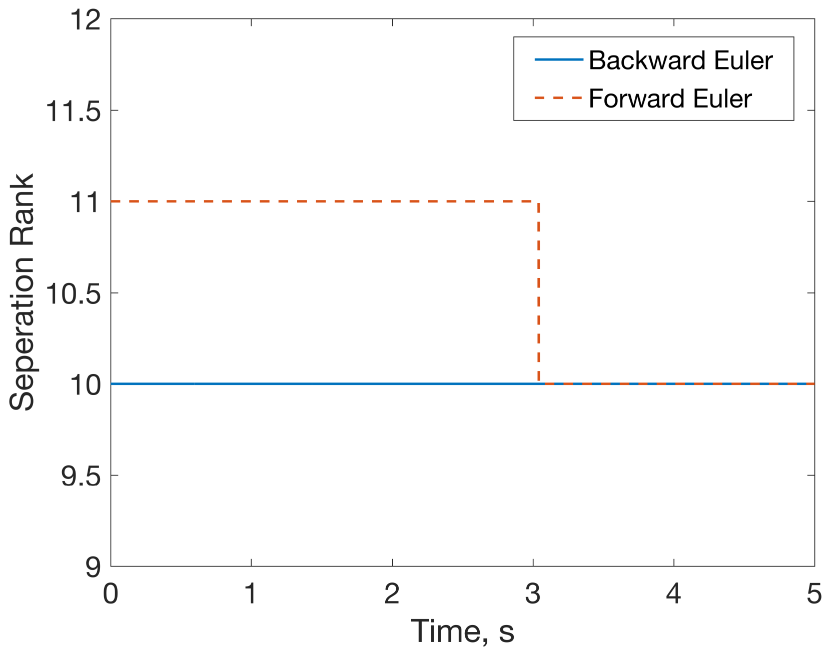


Figure A.9: Seperation rank of the desirability function for Example A.2 computed from Algorithms 3 and 4. The separation rank grows as the computation progresses backward in time for the forward algorithm (*i.e.,* Algorithm 3), but not the backward algorithm (*i.e.,* Algorithm 4).

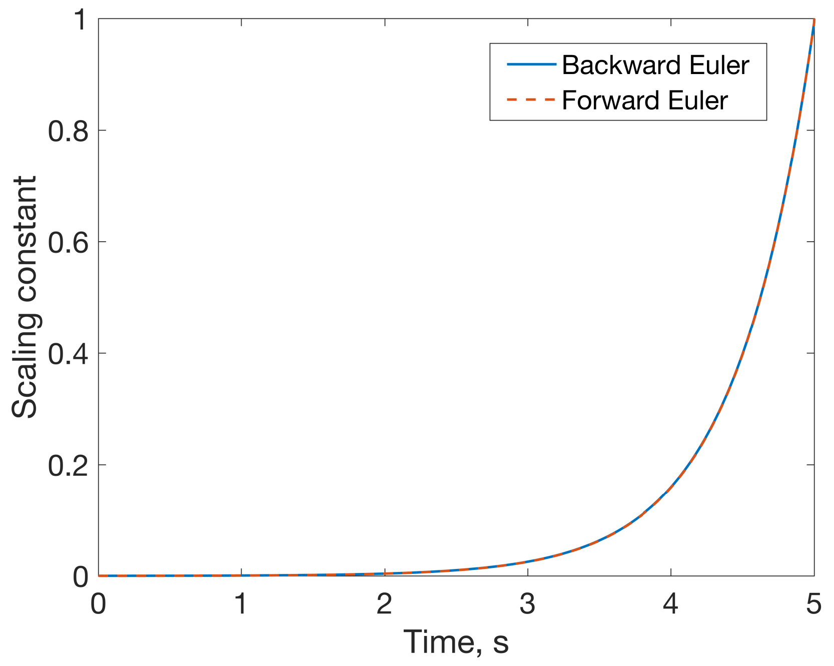


Figure A.10: Scaling constants of the desirability function for Example A.2 computed from Algorithms 3 and 4. The scaling constants decreases as expected when the computation progresses backward in time.

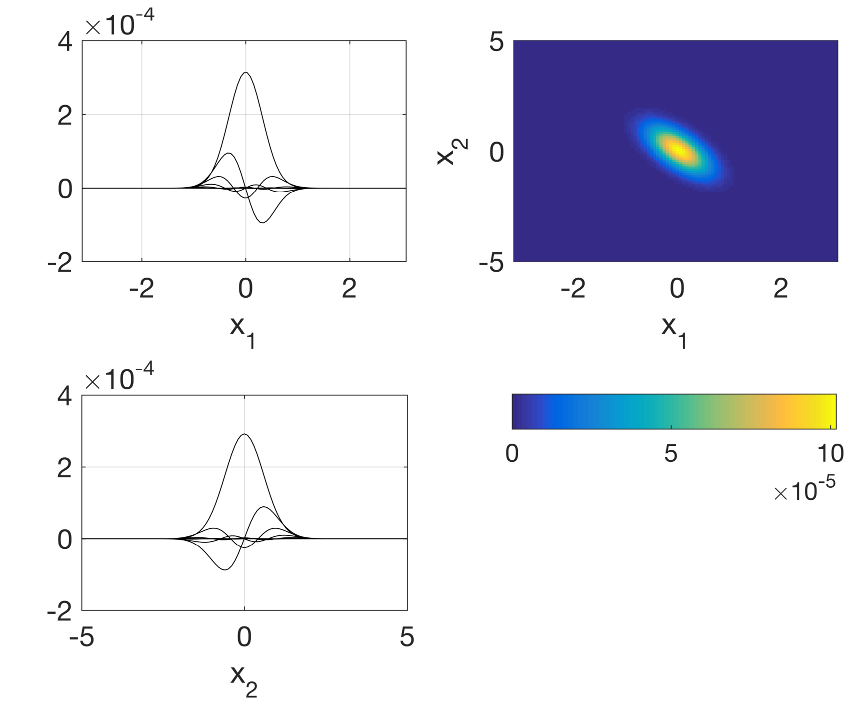


Figure A.11: Basis functions and desirability function of Example A.2 computed from Algorithm 3 at time $t = 0$ seconds.

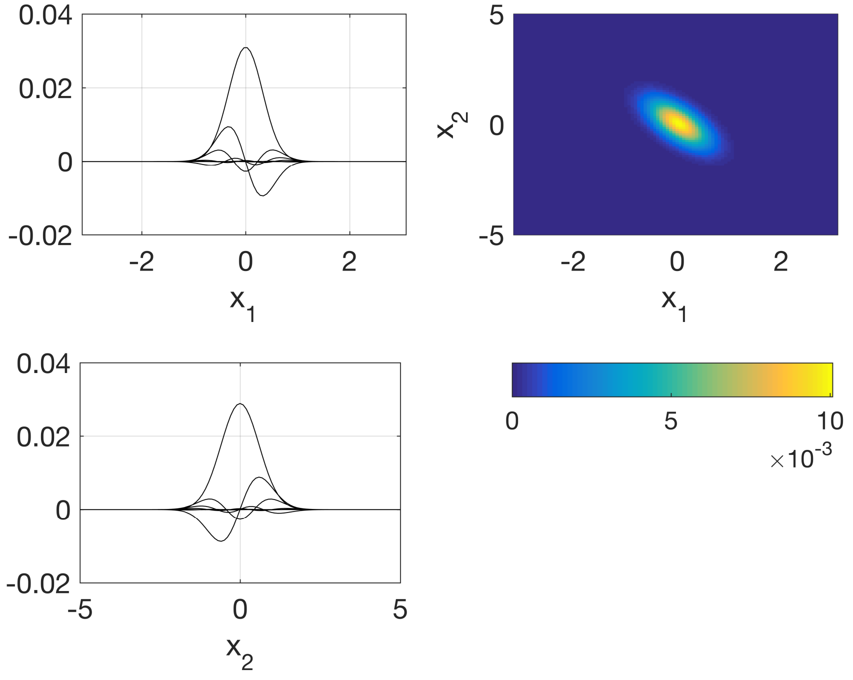


Figure A.12: Basis functions and desirability function of Example A.2 computed from Algorithm 3 at time $t = 2.5$ seconds.

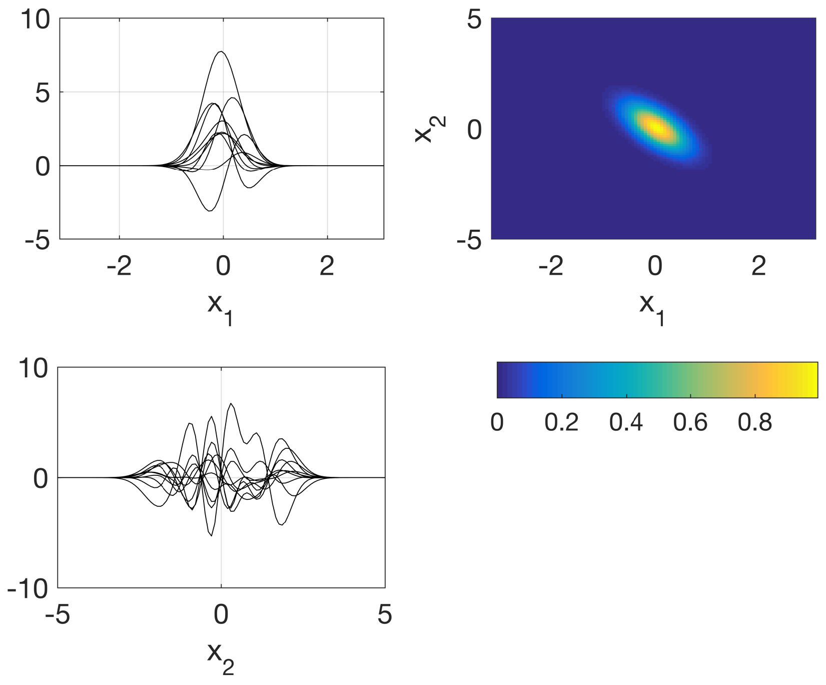


Figure A.13: Basis functions and desirability function of Example A.2 for Algorithms 3 and 4 at the final time $t = 5$ seconds.

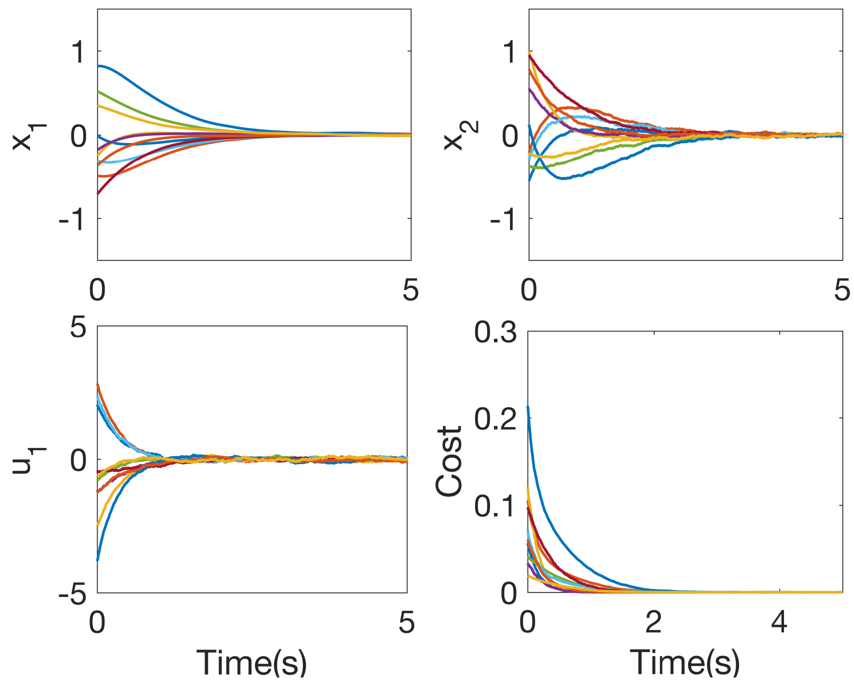


Figure A.14: State, input, and cost trajectories for 10 random trials of Example A.2 using the controller from Algorithm 3. All states and inputs converge to the origin by $T = 5$ seconds.

The backward algorithm, Algorithm 4, shows a similar performance. The operator was compressed from rank 11 to 5 for a relative residual 10^{-6}. The total run time is 64 minutes, a longer time than that of the forward algorithm as it involves solving a tensor least square problem at every time step instead of tensor multiplications. The backward algorithm has the same final time condition (Figure A.13) as the forward algorithm. Figures A.15 and A.16 shows the basis functions and the desirability function at the other two different times, namely $t = 2.5$, and 5 seconds. Unlike the solution of the forward algorithm in Figures A.11 and A.12, the shape of the basis functions and the seperation rank do not change.

Simulation results using a controller computed by Algorithm 4 are shown in Figure A.17 in which the system moves to the origin as desired. As before, Figure A.17 shows 10 simulations of the system with initial states randomly chosen and the corresponding input trajectories and the cost trajectories. Although the basis functions for solutions from Algorithms 3 and 4 are not the same at all time, the simulation results in Figures A.14 and A.17 are identical. The average finite horizon cost over the 10 simulations is 0.0271.

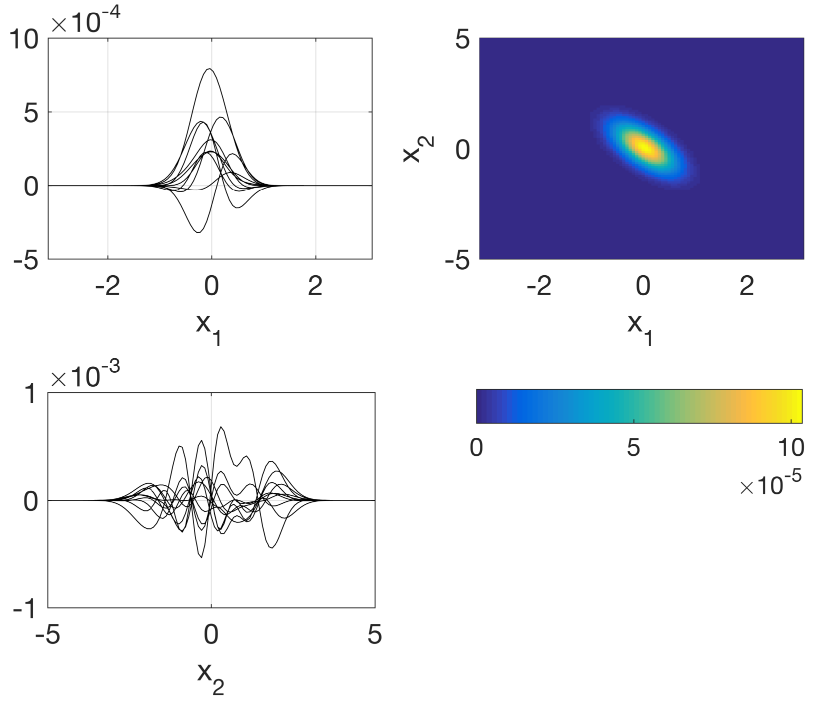


Figure A.15: Basis functions and desirability function of Example A.2 computed from Algorithm 4 at time $t = 0$ seconds.

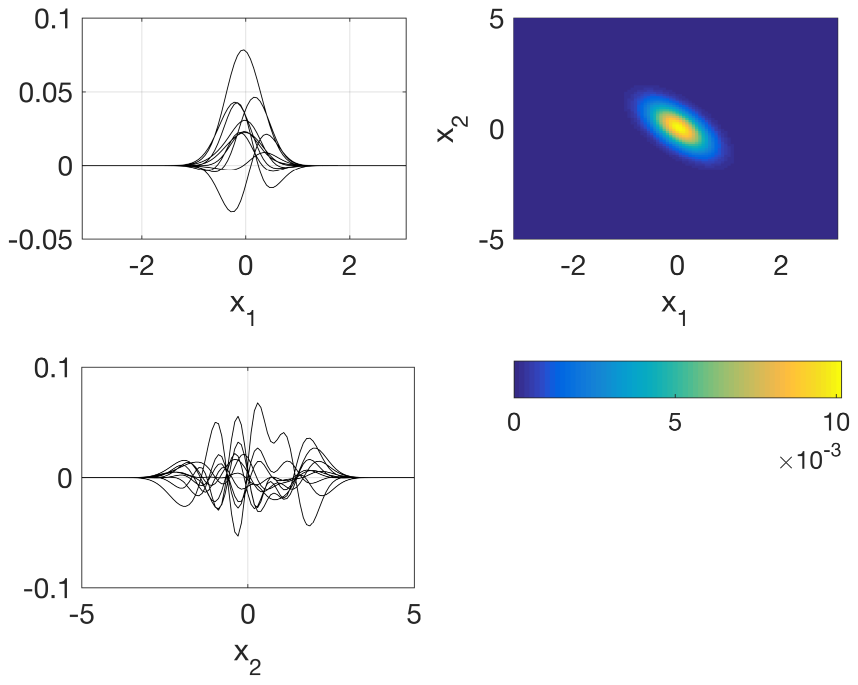


Figure A.16: Basis functions and desirability function of Example A.2 computed from Algorithm 4 at time $t = 2.5$ seconds.

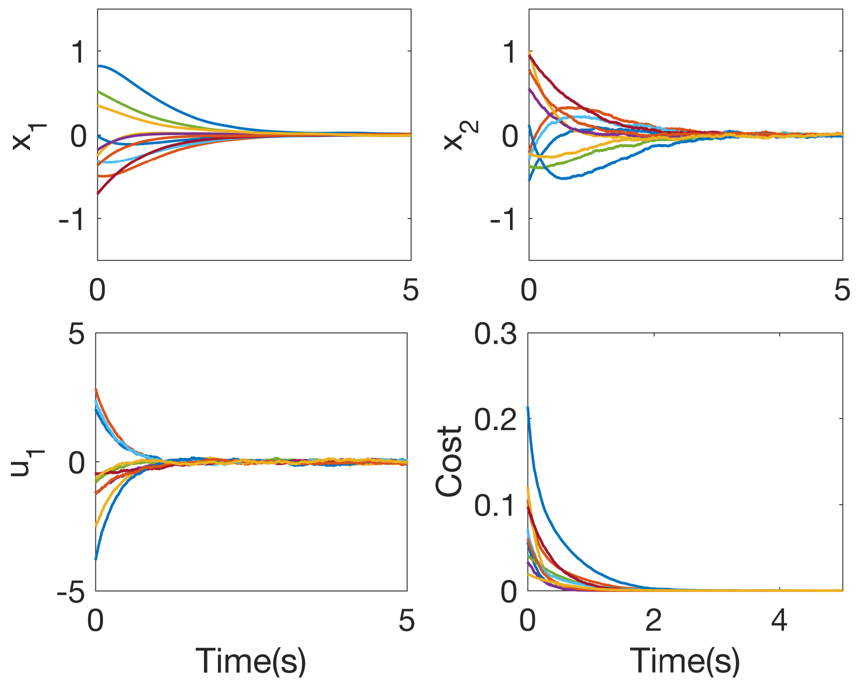


Figure A.17: State, input, and cost trajectories of 10 random trials for Example A.2 using the controller from Algorithm 4. All states and inputs converge to the origin at $T = 5$ seconds.

A.4 Infinite Horizon Problem

Algorithm 6 is demonstrated using three different examples. A MacBook Pro with 2.9 GHz i5 dual-core processor and 16 GB RAM is used for all computations. The regularizer α is set to 10^{-12} for all numerical examples. The simulations are implemented using Euler integration with 0.0005s time step. For all examples, Algorithm 6 is initialized with the value function of the linearization of the nonlinear systems. The value function of a linear system can be computed by solving the associated algebraic Ricatti equation, explained further in the next section. Given a value function, the corresponding desirability function is not necessarily a separable function. Thus, it is approximated with a low rank tensor using the ALS algorithm.

A.4.1 Linear Stochastic Optimal Control (LSOC)

First, we provide a brief overview of linear stochastic optimal control that is used to compare with the performance of the controller given by Algorithm 6.

Consider the following linear stochastic dynamical system,

$$dx = (Ax + Bu)\, dt + Bd\omega, \tag{A.1}$$

where $x \in \mathbb{R}^d$ represents the states, and $u \in \mathbb{R}^p$ represents tha control inputs. Given the infinite horizon cost (3.5), where $q(x) = x^{\mathrm{T}}Qx$, the optimal cost of (A.1) is $Tr(\Sigma_\varepsilon BP_{ss}B^{\mathrm{T}})$. The value of P_{ss} is obtained by solving the following algebraic Ricatti equation

$$A^{\mathrm{T}}P_{ss} + P_{ss}A - \frac{1}{2}P_{ss}BR^{-1}B^{\mathrm{T}}P_{ss} + Q = 0.$$

Note that the associated value function is $V(x) = x^{\mathrm{T}}P_{ss}x$. The optimal controller is thus $u(x) = -2R^{-1}B^{\mathrm{T}}P_{ss}x$. For a more thorough treatment of the optimal control of linear systems, refer to [122].

A.4.2 Linear System

Example A.1 is used to compare the technique proposed in this thesis with the LSOC. The goal is to stabilize the unstable linear system at the origin. The domain is chosen to be $\Omega = \{x \in \mathbb{R}^4 \mid \forall\, i, -5 \leq x_i \leq 5\}$. The cost functions are set to $q = x^{\mathrm{T}}x$ and $R = I$. At the boundary of Ω, $\Psi(x)$ is set to zero. The noise is set at $\Sigma_\varepsilon = I$. Spectral difference scheme is used for discretization. For each dimension i, there are $n_i = 51 + 2(i-1)$ discretization points. The operator was compressed from rank 47 to 23 for a relative residual 10^{-5}.

The algorithm terminates with a separation rank of 9 when the difference in eigenvalues is smaller than 10^{-7}. The total number of iterations is 14, and the total run time is approximately 5 hours. Shorter runs can be achieved by optimizing the algorithm for speed or by quitting earlier when the accuracy is good enough for the specific application. The convergence of the eigenvalue c' in Algorithm 6 is given in Figure A.18. The eigenvalue converges to 2.4772, which is close to the correct solution 2.4779. Simulation results are shown in Figure A.19 in which the system is stabilized as desired when using the controller computed by Algorithm 6. Figure A.19a shows 10 simulations of the system with initial states randomly chosen. Figure A.19b shows the corresponding input trajectories and the cost trajectories. The average infinite horizon cost over the 10 simulations is 0.1381.

Figure A.20 shows the simulation results when using the controller given by LSOC. The same noise values and initial conditions as in Figure A.19 are used. The average infinite horizon cost over the 10 simulations is 0.1733. The average difference between the cost is 0.0351. By comparing Figures A.19 and A.20, Algorithm 6 returns a controller that has relatively similar performance to the LSOC technique. In fact, the controller of Algorithm 6 for this specific example seems to give better transient cost than the controller of LSOC.

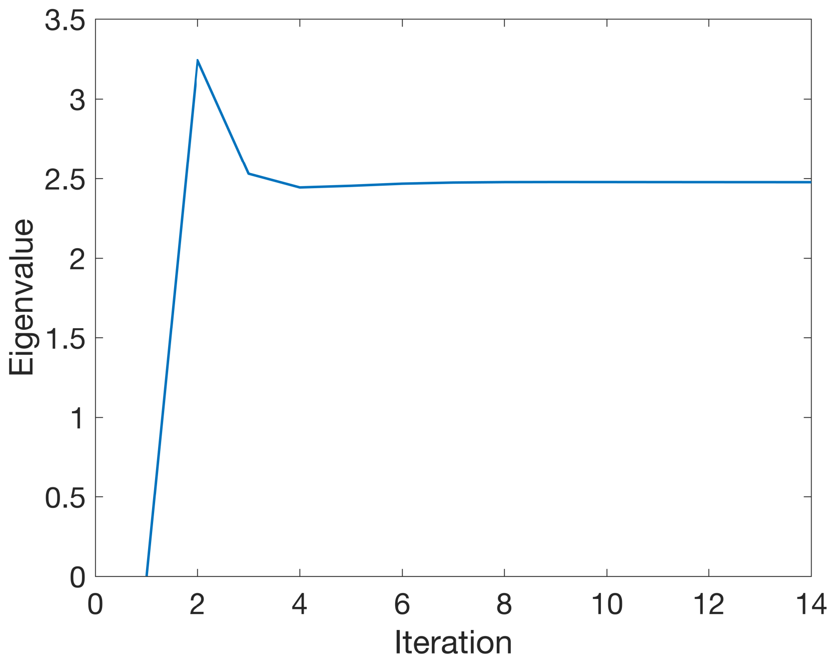
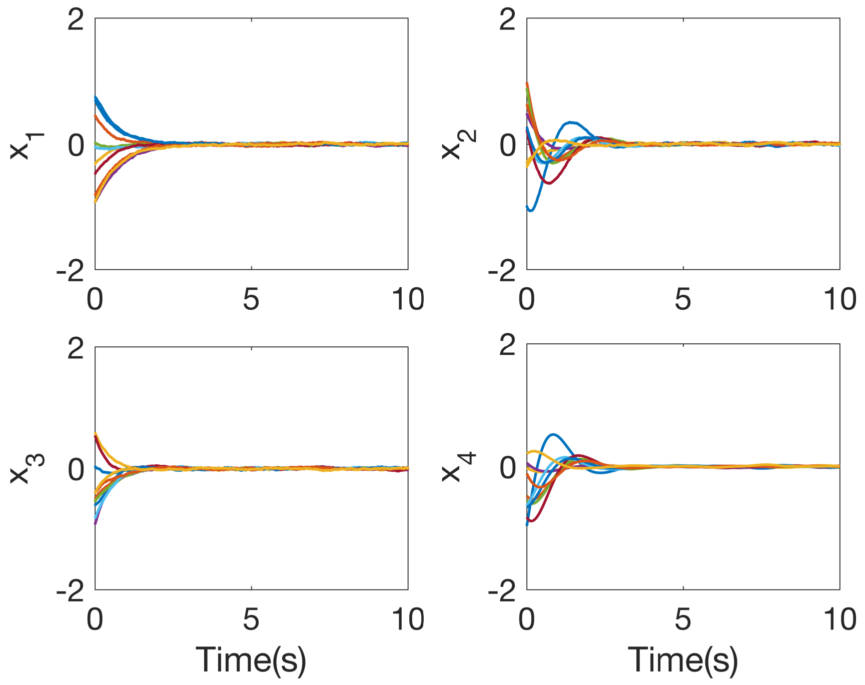
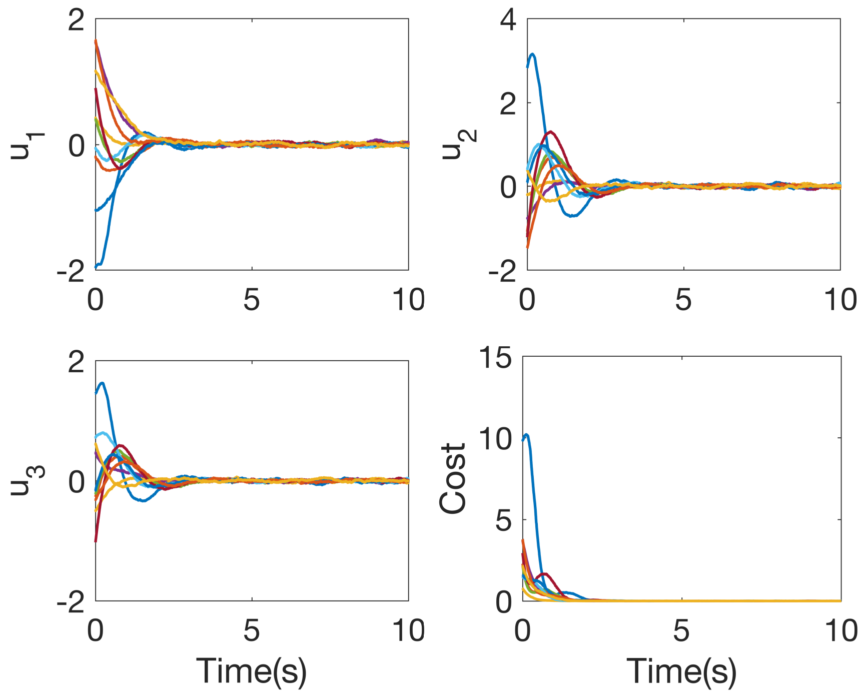


Figure A.18: Convergence of eigenvalue of Example A.1 when implementing Algorithm 6.

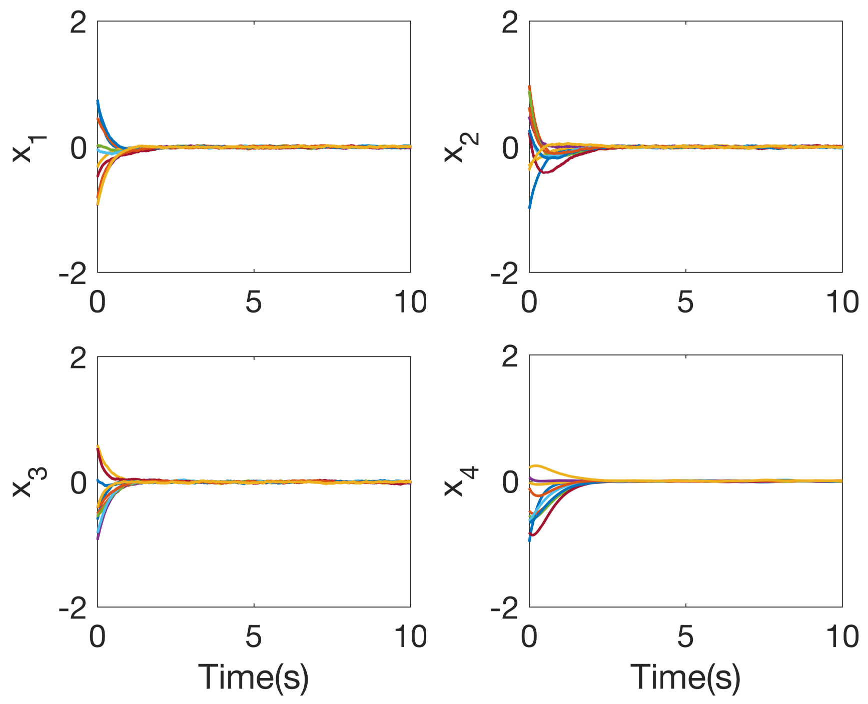


(a) State trajectories

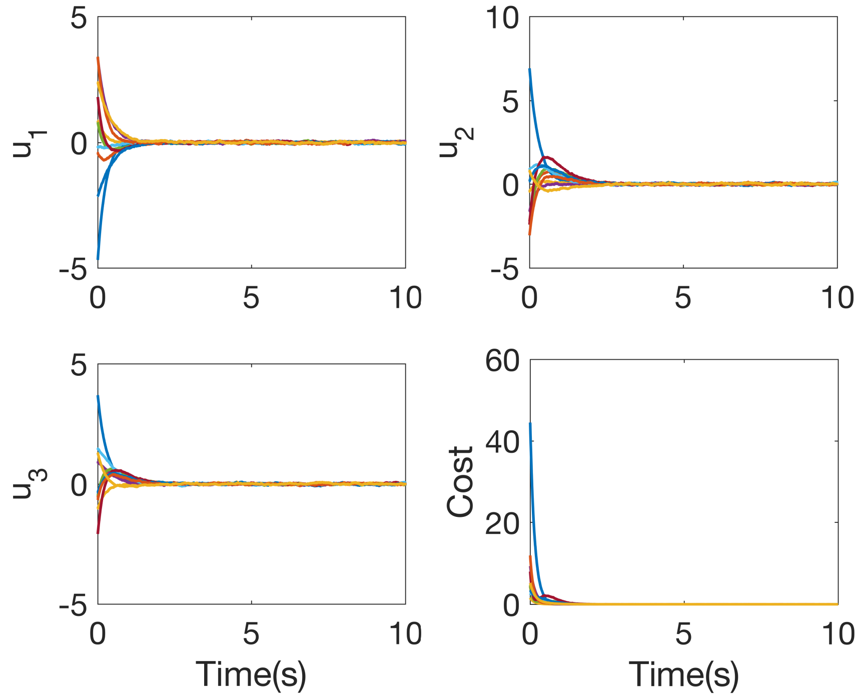


(b) Input and cost trajectories

Figure A.19: State, input, and cost trajectories for 10 random trials of Example A.1 using the controller from Algorithm 6. All states and inputs stabilize to zero.



(a) State trajectories



(b) Input and cost trajectories

Figure A.20: State, input, and cost trajectories for 10 random trials of Example A.1 using the controller from LSOC. All states and inputs stabilize to zero.

A.4.3 Simple Pendulum

Consider a simple nonlinear system in Example A.2. Similar as before, the goal is to stabilize the unstable system at the origin. The domain is chosen to be $\Omega = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \leq \pi, |x_2| \leq 5\}$. The cost functions are set to $q = 0.1x_1^2 + 0.05x_2^2$ and $R = 0.02$. At the boundary of Ω, the state x_1 has a periodic boundary condition, and $\Psi(x)$ is set to zero when $x_2 = \pm 5$. The noise is set at $\Sigma_\varepsilon = I$. A spectral difference scheme is used for discretization. The domain is discretized into 101 by 103 grid. The operator was compressed from rank 8 to 4 for a relative residual 10^{-6}.

The algorithm terminates with a separation rank of 12 when the difference in eigenvalues is smaller than 10^{-7}. The total number of iterations is 5, and the total run time is 37 seconds. The convergence of the eigenvalue is given in Figure A.21. The eigenvalue converges to 0.0366. Simulation results are shown in Figure A.22 in which the system is stabilized as desired when using a controller computed by Algorithm 6. Figure A.22a shows 10 simulations of the system with initial states randomly chosen and the corresponding input trajectories and the cost trajectories. Figure A.22b shows the same simulations but with a magnified view. The average infinite horizon cost over the 10 simulations is 0.0028.

Figure A.23 shows the simulation results when using the controller given by the LSOC of the linearized systems. Note that the controller is computed from the linearized system, but it is implemented on the nonlinear system. The same noise values and initial conditions as in Figure A.22 are used. The average infinite horizon cost over the 10 simulations is 0.0033. Again, by comparing Figures A.22 and A.23, Algorithm 6 returns a controller that has a relatively similar, but slightly better, performance to the LSOC technique.

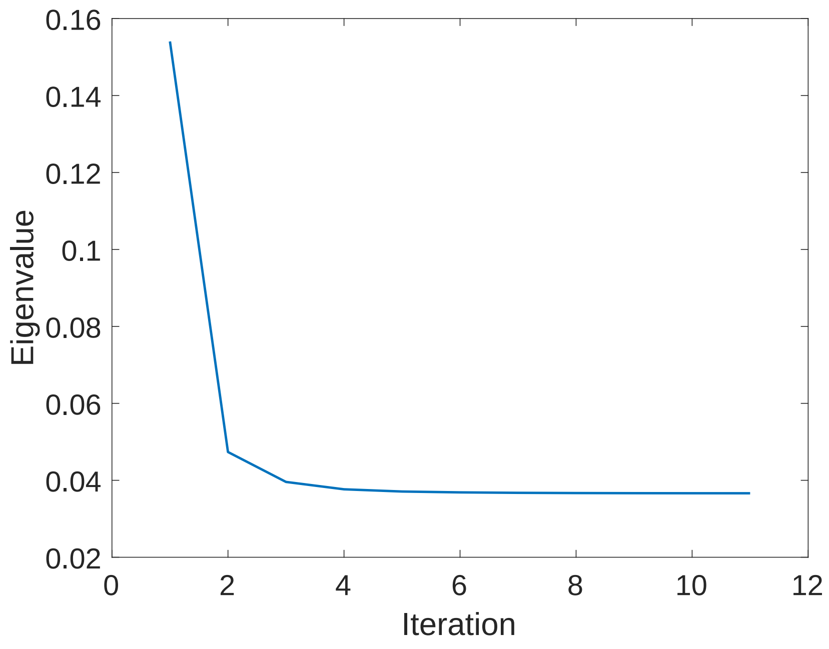
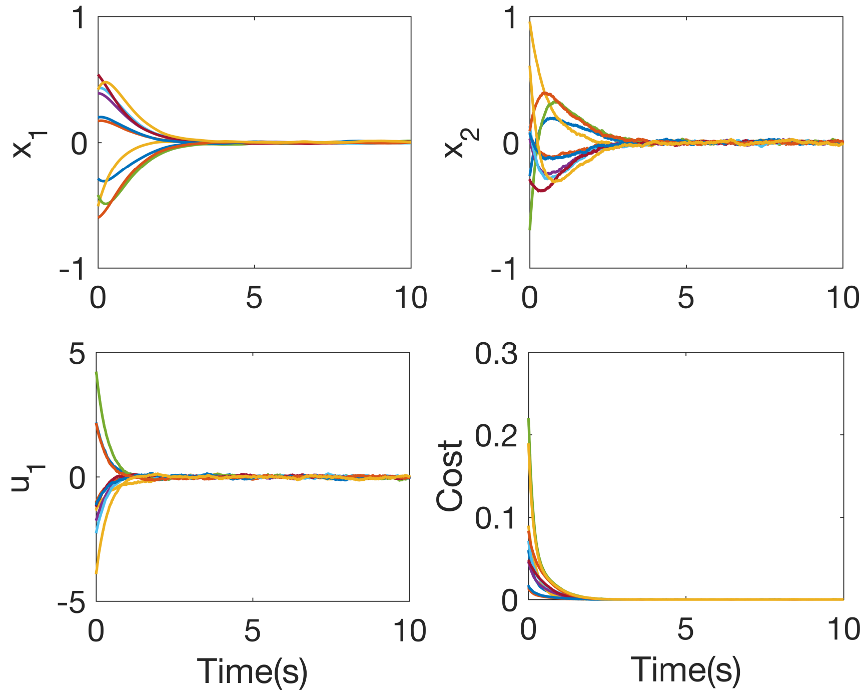
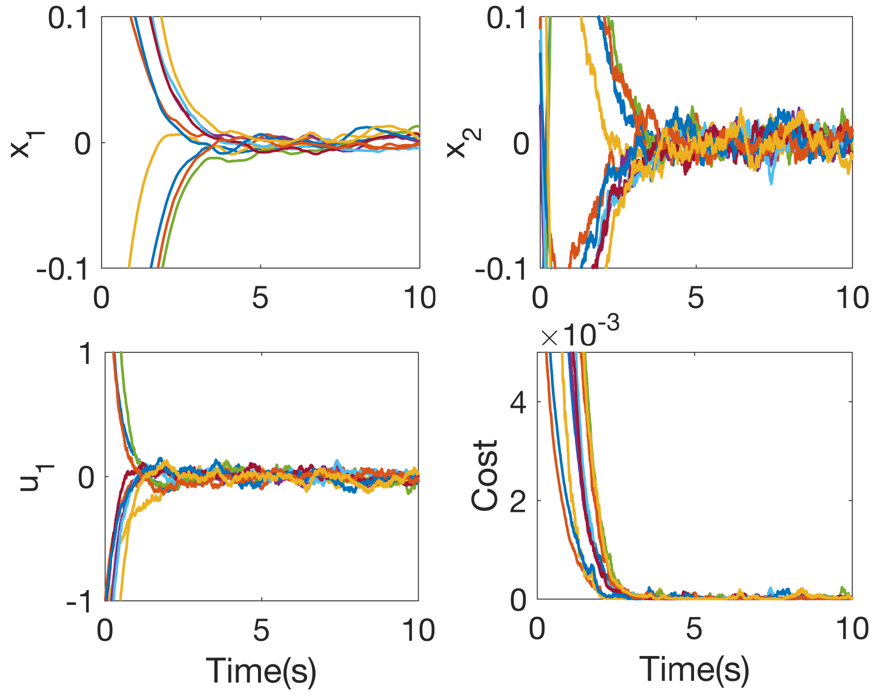


Figure A.21: Convergence of eigenvalue of Example A.2 when implementing Algorithm 6.

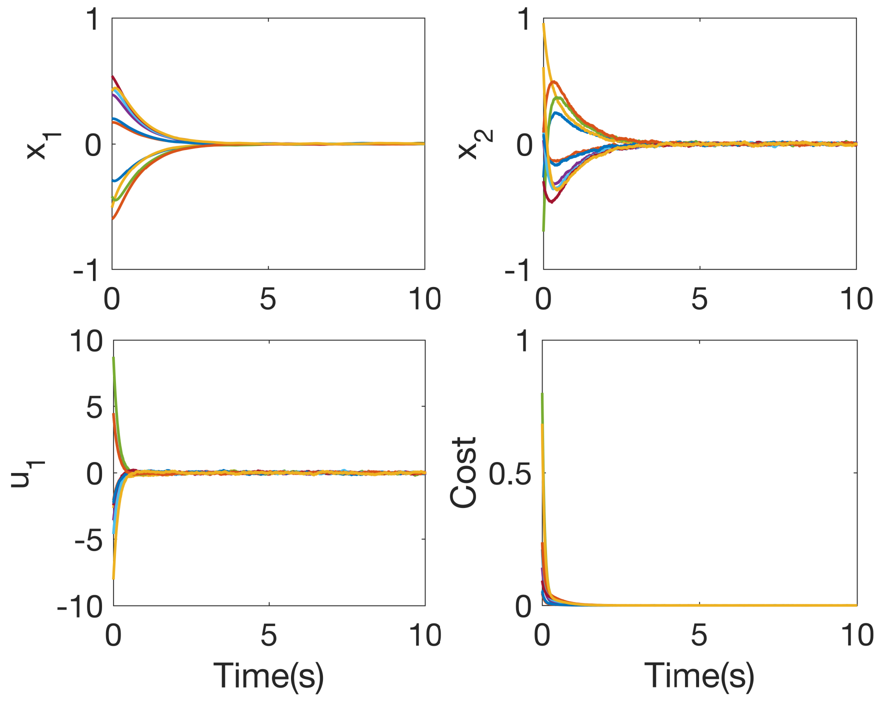


(a) Full view

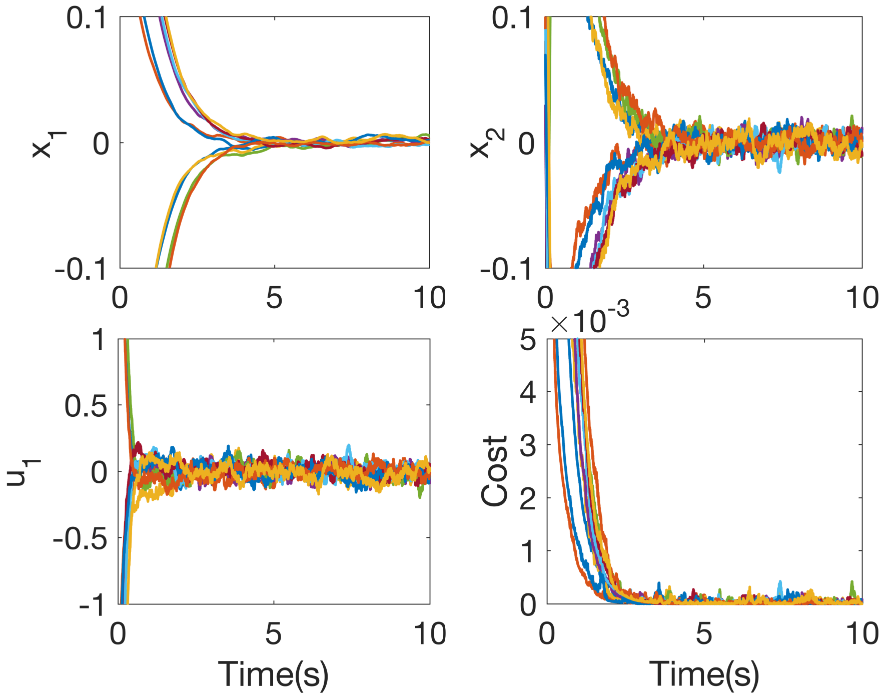


(b) Magnified view

Figure A.22: State, input, and cost trajectories for 10 random trials of Example A.2 using the controller from Algorithm 6. All states and inputs stabilize to zero.



(a) Full view



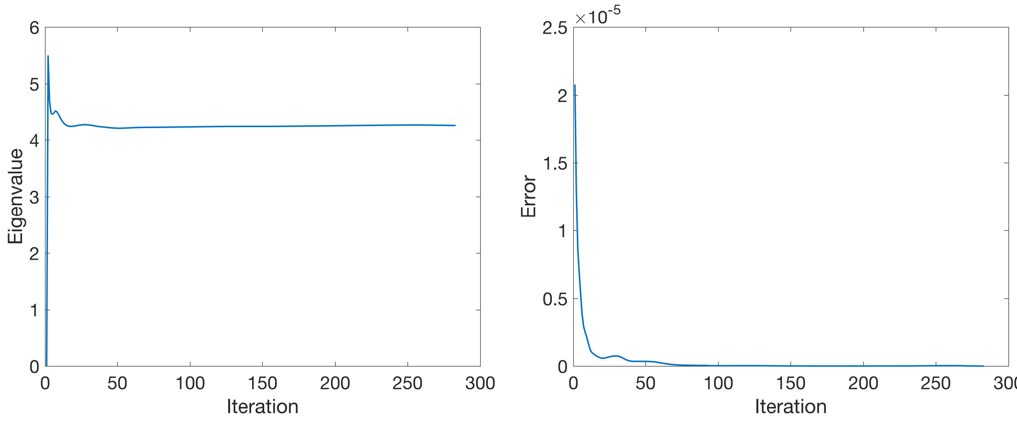
(b) Magnified view

Figure A.23: State, input, and cost trajectories for 10 random trials of Example A.2 using the controller from LSOC. All states and inputs stabilize to zero.

A.4.4 VTOL

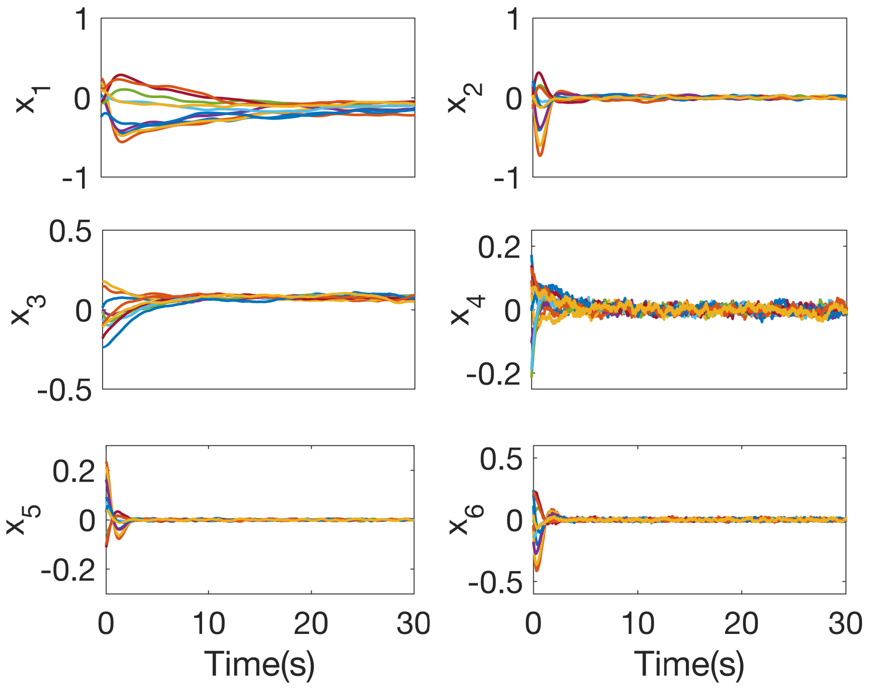
Lastly, consider Example A.4, which is a six dimensional nonlinear system. The domain Ω is defined by $p_x \in [-5, 5]$, $p_y \in [-5, 5]$, $v_x \in [-5, 5]$, $v_y \in [-5, 5]$, $v_\theta \in [-5, 5]$, and a periodic $\theta \in [-\pi, \pi]$. The cost functions are $q(x) = x^{\mathrm{T}}x$ and $R = I$. The goal is again to stabilize the system at the origin. Boundary conditions $\Psi(x)|_{x\in\partial\Omega} = 0$ is imposed for all non-periodic states. The noise is set at $\Sigma_\varepsilon = I$. Spectral difference scheme is used for discretization. The domain is discretized into a $101 \times 103 \times 105 \times 107 \times 107 \times 101$ grid. The operator was compressed from rank 48 to 20 for a relative residual 10^{-6}.

The algorithm terminates with a separation rank of 15 when the difference in eigenvalues is smaller than 10^{-8}. The total number of iterations is 283, and the total run time is 11 hours. The convergence of the eigenvalue is given in Figure A.24. The eigenvalue converges to 4.2586. As seen in Figure A.24, the algorithm could be terminated earlier than 283 iterations when both the error $\|F - F'\|$ and the difference in eigenvalue are small enough for a given application. Simulation results are shown in Figure A.25 in which the system is stabilized as desired when using a controller computed by Algorithm 6. Figure A.25a shows 10 simulations of the system with initial states randomly chosen, and Figure A.25b shows the corresponding input trajectories. The average infinite horizon cost over the 10 simulations is 0.0386.

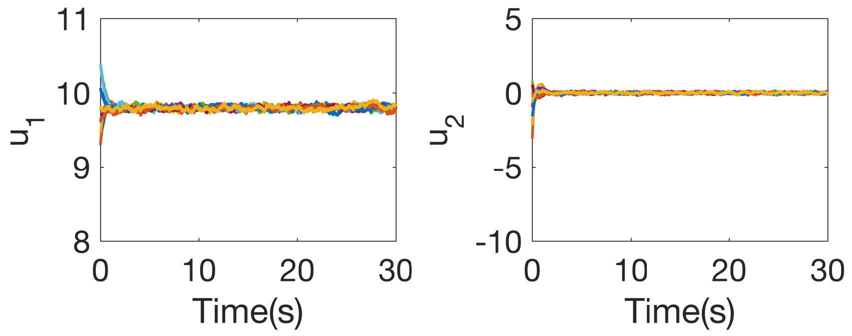


(a) Eigenvalue (b) Error

Figure A.24: Convergence of eigenvalue and error $\|F - F'\|$ of Example A.4 when implementing Algorithm 6.



(a) State trajectories



(b) Input trajectories

Figure A.25: State, input, and cost trajectories for 10 random trials of Example A.4 using the controller from Algorithm 6. All states and inputs stabilize to zero. (a) is the same as Figure 5.3.

Part II

Optimal Control Synthesis for Hybrid Systems with Qualitative and Quantitative Objectives

Chapter 6

INTRODUCTION TO PART II

The previous part of this thesis solves optimal control problems without any strict system behavior specifications. However, in many applications there exist requirements, such as a mobile robot's need to avoid certain regions at all times and visit other regions in a particular sequence. These requirements are incorporated into the controller synthesis procedure using formal methods.

Formal synthesis is a paradigm for designing controllers automatically, which are correct-by-construction, and thus reduces the verification overhead. In this paradigm, a mathematical model of a system to be controlled and formal specifications of properties that are expected of the controlled system are given as inputs to compute a controller that ensures the controlled system satisfies the properties. For example, given a model for the behavior of a robot, synthesize a plan that reaches a given part of the workspace while avoiding certain obstacles. The high level specifications, for instance, could also capture mission level objectives such as surveillance that require autonomous systems to operate over an unbounded amount of time. Such specifications are captured in logics such as Linear Temporal Logic (LTL) or using automata such as Büchi automata [30], [31], [123]–[125].

Since the work of [126] on automated synthesis began, multiple directions have been pursued, including synthesizing finite state systems with respect to temporal logic objectives [127], [128], and controlling discrete event systems [129]. Early works in hybrid control systems focused on identifying subclasses of systems for which controller synthesis is decidable including timed automata [130], rectangular hybrid automata [131], and o-minimal automata [132], [133]. However, these classes of systems have limited continuous and discrete dynamics, and the synthesis problem becomes undecidable for a relatively simple class of hybrid systems [134].

For systems with complex dynamics, [135] introduced an abstraction based controller synthesis. Abstraction based formal controller synthesis consists of the following steps:

1. A finite state abstraction of the system is constructed that "under-approximates" the control actions and "over-approximates" the environment actions;

2. An abstract controller for the abstract system is constructed using results from automata and game theory;
3. A concrete controller for the original system is extracted from the abstract controller, and is then implemented in the given system.

This method has been successfully applied in controller synthesis of switched systems [31], [125], and robotic path planning [136], [137].

Formal synthesis techniques that automatically generate controllers from high level specifications have gained momentum in the recent past [30], [123], [124], [138]. While algorithmic results based on automata and game theory exist for controller synthesis of finite state systems [123], abstraction based methods have gained prominence in the case of dynamical systems with a potentially infinite state space [28], [30], [139]–[141].

Often, in addition to designing a correct controller that satisfies qualitative objectives such as avoiding obstacles, an application may require an optimality condition. For instance, a robot should reach a desired state with minimum battery drain while avoiding all obstacles. This thesis investigates an abstraction-refinement approach to synthesize an optimal controller with regular properties that allow for bounded time specifications such as reaching a target region or traversing a sequence of regions, and ω-regular properties that allow for long term specifications such as performing surveillance using minimum fuel in the long run.

This part of the thesis starts by defining pre-orders on systems, which respect optimality and existence of controllers. That is, if a system $\mathcal{H}_2$ is higher up in the ordering than a system $\mathcal{H}_1$, then the existence of controller for $\mathcal{H}_2$ with respect to a qualitative property implies the existence of the same for $\mathcal{H}_1$, and the cost of the optimal controller for $\mathcal{H}_1$ is at most that of $\mathcal{H}_2$. In this thesis, $\mathcal{H}_2$ is a finite state weighted transition system on which the optimal controller synthesis problem is solved with respect to given objectives. For regular objectives, a finite length two-player game on a finite graph is solved. For ω-regular objectives, a two-player quantitative game, namely, mean payoff parity game is solved. Here, mean payoff refers to the "average" cost, and parity games encode the Büchi condition. For both, convergence to a controller with costs arbitrarily close to the optimal cost is achieved through iterative refinement, when certain kinds of robust controllers exist.

More precisely, first, an "abstraction" — a simplified finite state system — is constructed from partitions of the state and input spaces, and the edges of the system are annotated with weights, which over-approximate the costs in the original system. Then, a two-player game on the finite state system is solved to obtain a controller for the abstraction, and subsequently, a controller for the original dynamical system. This approach iteratively consider finer partitions of the state and input spaces, corresponding to grids of size $C/2^i$ for some constant C and $i = 0, 1, 2, \ldots$. For discrete-time hybrid systems, if the dynamics and the cost function are Lipschitz continuous and the optimal control for the original system satisfies certain kinds of robustness property, the cost of the sequence of controllers constructed by the synthesis procedure converges to the optimal cost.

6.1 Related Work and Contributions

The main contribution of this thesis is the optimality guarantee on the controllers synthesized — an important missing piece in most previous works that study optimal controller synthesis using formal approaches [28], [29]. A hierarchical optimal controller synthesis problem was studied [28], yet, no formal guarantees on the optimal cost are provided. Similarly, [29] considered optimal control synthesis by combining linear temporal logic, potential functions, and model predictive control without formal guarantees on the optimal cost. On the other hand, the sequence of controllers constructed by the approach presented in this thesis converges to the optimal cost for discrete time hybrid systems. Furthermore, for each suboptimal controller, the resulting trajectories have cost no greater than the optimal cost of the corresponding abstract system. Thus, when computational resources is limited, the best suboptimal controller found is guaranteed to generate trajectories with known bounded costs.

In addition, the bounded length regular objective is more general than classical finite horizon optimal control problems [142], [143] because the time horizon is not fixed a priori. One of the approaches described in this thesis focuses on finite horizon optimal control problems, but the input sequence length is not a priori fixed because the regular property consists of finite traces, whereby the length is variable.

Apart from that, the method presented allows for a larger class of cost functions in comparison to previous works [123], [139], [140], [144], [145]. These works [139], [140] used abstraction-based methods to find an optimal time controller that gives the shortest path, which satisfies certain reachability conditions. Similarly, [144] considers a more general problem, but the cost is restricted to only include state cost without allowing for transition cost. In contrast, the method in this thesis encodes transition cost in the abstraction scheme, thus allowing for picking a path that is "shortest" with respect to a more general class of optimality conditions. In term of ω-regular objectives, [123], [145] solve for optimal control under LTL specifications. However, [123] minimizes a specific cost function — the maximum time between satisfying instances of the optimizing proposition, and [145] considers the weighted average cost. A larger class of cost functions is considered in [146], but it restricts the specifications to a fragment of LTL specifications for computational efficiency. The method proposed by this thesis allows for a larger class of cost functions in comparison to previous works, and it does not place any restriction on the possible specifications.

Another approach that is relevant to this thesis is to solve the optimal control problem by abstracting the dynamical systems such that the value function of the abstracted system is an upper bound to the value function associated with the original optimal control problem [141], [147]. This technique allows for uncertainties in the systems. However, it only applies to systems with bounded time behaviors. On the other hand, [148] studies infinite behaviors, mainly the "reach and stay while avoid" specifications, with average costs. Instead, the framework presented in this thesis allows for both regular and ω-regular objectives.

Furthermore, the mixed-integer linear program based techniques in References [124], [149] synthesize optimal controllers for a large class of systems and cost functions with LTL specifications. However, unlike the approach presented, which returns a feedback controller, these techniques return an open loop controller that requires another layer of feedback controller to handle disturbances.

Lastly, while formal controller synthesis with respect to qualitative objectives over infinite horizon such as those expressed in LTL or quantitative objectives such as mean payoff have independently received attention in the context of dynamical systems, a framework that considers both aspects has received relatively less attention. This thesis combines ideas from abstraction based controller synthesis for dynamical systems and finite quantitative games [150], [151] to obtain an abstraction refinement scheme for optimal controller synthesis of ω-regular objectives. The technique in this thesis is complimentary to [150]–[152] that study finite quantitative games. The approach proposed constructs a two-player weighted graph from the abstraction of a given dynamical system and formulates a two-player mean payoff parity game based on ω-regular properties. The algorithms from [150], [151] solve the mean payoff parity game, and return the optimal strategy that are projected back to the original dynamical system using the approach in this thesis. A mean payoff parity game may not return a finite strategy in general [151]. However, a finite memory strategy is achievable for an ε-optimal cost through solving an energy parity game [150]. Therefore, this thesis brings together techniques in finite quantitative games and abstraction based controller synthesis.

The generality of the approach proposed enables control engineers to synthesize controllers with more flexible structure and cost considerations. The method introduced in this thesis applies to the general class of discrete-time hybrid systems. However, due to its generality, the computation burden could be high because the optimizations that compute the weights depend on the cost function and the dynamics. A prototype tool OptCAR that implements the abstraction refinement algorithm is developed for synthesizing a (finite) sequence of controllers for discrete-time linear hybrid systems.

6.2 Outline

This part of the thesis includes the following chapters:

Chapter 7 introduces the mathematical notations, presents the semantic model for discrete time hybrid systems with cost (*i.e.,* weighted transition systems), and formalizes the optimal control problem.

Chapter 8 defines the preorder for optimal control that preserves the cost and presents the abstraction refinement procedure for constructing finite state systems, which simulate a given transition system, termed the abstract system. The abstract system satisfies the condition that the cost of the optimal control on the abstract system provides an upper bound on the cost of the optimal control for the original system. Furthermore, each suboptimal controller yields trajectories that have the cost upper bounded by the cost of the optimal control on the corresponding abstract system.

Chapter 9 presents the abstraction-refinement method to synthesize control inputs for a discrete-time hybrid system. The controlled system behavior satisfies a finite-word linear-time temporal objective while incurring minimal cost. An abstract finite state weighted transition system is constructed from finite partitions of the state and input spaces by solving optimization problems. A sequence of suboptimal controllers is obtained by considering a sequence of uniformly refined partitions. In fact, the costs achieved by the sequence of suboptimal controllers converge to the optimal cost for a class of hybrid systems that has robust optimal input trajectory. Examples illustrate the feasibility of this approach to synthesize automatically suboptimal controllers with improving optimal costs.

Chapter 10 presents the abstraction-refinement based framework for optimal controller synthesis of discrete-time hybrid systems with respect to ω-regular objectives. Similar to Chapter 9, it consists of first abstracting the discrete-time "concrete" system into a finite weighted transition system using a finite partition of the state-space. Then, a two-player mean payoff parity game is solved on the product of the abstract system and the Büchi automaton corresponding to the ω-regular objective, to obtain an optimal "abstract" controller that satisfies the ω-regular objective. The abstract controller is guaranteed to be implementable in the concrete discrete-time system, with a sub-optimal cost. The abstraction is refined with finer partitions to reduce the sub-optimality. Under the assumption on the existence of certain robust controllers, the refinement procedure is guaranteed to find controllers whose costs are arbitrarily close to the optimal cost. An example is presented to illustrate the feasibility of the approach.

Appendices B–D contain the full proofs for results in Chapters 8–10, respectively.

Part of the contents in this part appeared in publication [153]. However, the convergence results are generalized to discrete time hybrid nonlinear systems instead of just piecewise linear systems.

Chapter 7

WEIGHTED TRANSITION SYSTEMS

This chapter describes the semantic model for discrete time hybrid systems with cost (*i.e.,* weighted transition systems), and formalizes the optimal control problem. We begin with defining useful mathematical notations that are used throughout this part of the thesis.

7.1 Notations and Definitions

An ε-ball around a point x is denoted as $\mathcal{B}_\varepsilon(x) = \{y \in \mathbb{R}^n \mid \|y - x\|_\infty \leq \varepsilon\}$. Let $S \subseteq \mathbb{R}^k$ be a k-dimensional subset. The function *Grid* splits S into rectangular sets with ϵ width. That is, $Grid(S, \epsilon) =$

$$\left\{ S' \;\middle|\; \exists d_1, \ldots, d_k \in \mathbb{Z}, S' = S \bigcap \prod_{i=1}^{k} (d_i\epsilon, (d_i + 1)\epsilon) \right\}.$$

Given a function, $f : \mathcal{A} \to \mathcal{B}$, for any $A \subseteq \mathcal{A}$, $f(A) = \{f(a) | a \in A\}$. The domain of a function f is denoted as $dom(f)$. Given an equivalence relation $R \subseteq A \times A$ and an element $a \in A$, $[a]_R = \{b \,|\, (a, b) \in R\}$ denotes the equivalence class of R containing a.

7.2 Weighted Transition Systems

This section defines the weighted transition systems and related concepts.

Definition 7.1. A weighted transition system is defined as $\mathcal{T} = (\mathcal{S}, \mathcal{S}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$, where

- $\mathcal{S}$ is a set of states;
- $\mathcal{S}^{init} \subseteq \mathcal{S}$ is a set of initial states;
- $\mathcal{U}$ is a set of control inputs;
- $\mathcal{P}$ is a set of propositions;
- $\Delta \subseteq \mathcal{S} \times \mathcal{U} \times \mathcal{S}$ is a transition relation;
- $\mathcal{L} : \mathcal{S} \to \mathcal{P}$ is a state labeling function, and
- $\mathcal{W} : \mathcal{S} \times \mathcal{U} \times \mathcal{S} \to \mathbb{R}_+$ is the transition cost function.

An equivalent notation for the set of propositions is to define $\mathcal{P}'$ as the set of propositions and let the labeling function map from states to the power set of $\mathcal{P}'$.

Here, $\mathcal{P}$ is related to $\mathcal{P}'$, whereby $\mathcal{P} = 2^{\mathcal{P}'}$. Henceforth, a weighted transition system is referred as a transition system. For any $s \in \mathcal{S}$, define the set *Enabled*$(s) = \{u \in \mathcal{U} \mid \exists s' \in \mathcal{S}\ s.t.\ (s, u, s') \in \Delta\}$ to represent all inputs for which there is a transition from the state s. A transition system is *complete* if for all $s \in \mathcal{S}$, $Enabled(s) = \mathcal{U}$. A transition system is *finite* if $\mathcal{S}$ and $\mathcal{U}$ are finite. A finite state *automaton* (denoted $(\mathcal{T}, P_f)$) is a finite transition system $\mathcal{T}$ along with a proposition $P_f \in \mathcal{P}$, which represents the final states. For the rest of the section, fix the transition system $\mathcal{T} = (\mathcal{S}, \mathcal{S}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$.

Definition 7.2. A transition system $\mathcal{T}$ is a complete transition system if for all $s \in \mathcal{S}$, $\mathcal{U} = Enabled(s)$.

7.3 Paths and Traces

A *path* of the transition system $\mathcal{T}$ is a finite or infinite sequence of states and inputs, $\zeta = s_0 u_0 s_1 u_1 s_2 \ldots$, where $s_0 \in \mathcal{S}^{init}$, $s_i \in \mathcal{S}$, $u_i \in \mathcal{U}$, and $(s_i, u_i, s_{i+1}) \in \Delta$. The set of all paths of $\mathcal{T}$ is denoted *Paths*$(\mathcal{T})$, and the set of all finite paths of $\mathcal{T}$ that ends at a state (*i.e.,* the last item in a path is a state instead of an input) is denoted *Paths*$_f(\mathcal{T})$. Given a finite path $\zeta = s_0 u_0 s_1 u_1 s_2 \ldots u_{k-1} s_k$ of $\mathcal{T}$, the length of ζ is $k + 1$.

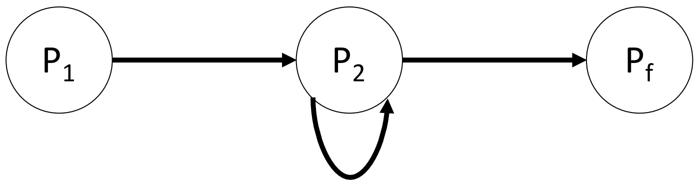
A *trace* of a transition system is the sequence of state labels of a path. The trace of ζ, denoted *Tr*(ζ), is the sequence $\mathcal{L}(s_0)\mathcal{L}(s_1)\ldots$.

7.4 Properties

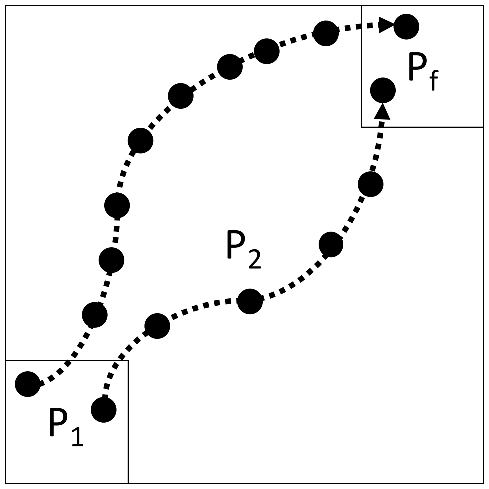
A *property* Π over a set of propositions $\mathcal{P}$ is a set of finite or infinite sequences $\pi = p_0 p_1 \ldots$, where each $p_i \in \mathcal{P}$. A property describes the desired behaviors of the system. For algorithmic purposes, we need to restrict ourselves to properties that can be specified using some finite data structure. Two classes of properties are considered in this thesis: regular properties and ω-regular properties.

7.4.1 Regular Properties

Regular property describes finite behaviors of systems. A property is *regular* if it consists of the traces of a finite state automaton $(\mathcal{T}, P_f)$, that is, it is the set of all traces of paths of $\mathcal{T}$, which start in an initial state and end in a state labelled by P_f. Thus, a regular property is a set of only finite sequences. This thesis considers regular property that is specified by a finite state automaton $(\mathcal{T}, P_f)$. Figure 7.1 shows an illustration. The properties expressed by popular logics such as finite words linear-time temporal logic (LTL) [154] are regular, but their translation into



(a) Automaton representing a regular property Π of a finite behavior.



(b) Example paths given by a winning strategy σ with respect to Π.

Figure 7.1: An example of a regular property and corresponding paths given by a winning strategy. This property is more general than a typical finite horizon control problem because the length of the sequence/path is not set a priori.

the finite transition system representation can lead to an exponential blow up in the number of states with respect to the size of the formula [155].

Remark. Regular specifications already capture properties that are more general than finite horizon control, since a regular property can characterize behaviors involving unbounded length. For instance, consider reaching a target region without a priori bound on the number of steps required to reach the region.

7.4.2 ω-regular Properties

ω-regular properties are an expressive class of properties involving infinite behaviors, which can be represented by Büchi automata.

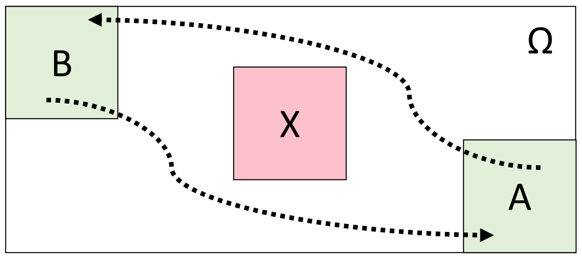
Definition 7.3. A Büchi automaton is defined as $\mathcal{B} = (Q, Q^{init}, \mathcal{P}, \mathcal{E}, \mathcal{F})$, where

- Q is a finite set of states;

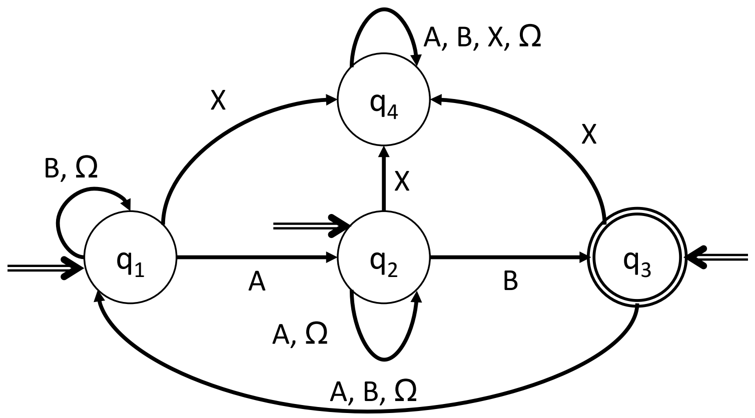
- $\mathcal{Q}^{init} \subseteq \mathcal{Q}$ is a set of initial states;
- $\mathcal{P}$ is a finite set of propositions;
- $\mathcal{E} \subseteq \mathcal{Q} \times \mathcal{P} \times \mathcal{Q}$ is a transition relation, and
- $\mathcal{F} \subseteq \mathcal{Q}$ is a set of accepting states.

A sequence π over Π is accepted by the Büchi automaton if there is a "path" in the automaton whose labels correspond to π and the path visits some state from $\mathcal{F}$ infinitely often. A run of a Büchi automaton $\mathcal{B}$ over a sequence $\pi = p_0 p_1 \ldots$ is a sequence $r = q_0 q_1 \ldots$ such that $q_0 \in \mathcal{Q}^{init}$ and $(q_i, p_i, q_{i+1}) \in \mathcal{E}$ for all i. Given a sequence of states $r = q_0 q_1 \ldots$, $\mathit{Inf}(r)$ denotes the states that occur infinitely often in r, that is, $\mathit{Inf}(r) = \{q \mid \forall i, \exists j > i, q_j = q\}$, A sequence π is *accepted* by $\mathcal{B}$ if there exists a run r over π such that $\mathit{Inf}(r) \cap \mathcal{F} \neq \emptyset$. The *language* of a Büchi automaton $\mathcal{B}$, denoted $\mathcal{L}(\mathcal{B})$, is the set of all sequences π that are accepted by it. A property Π is *ω-regular* if it is the language of some Büchi automaton $\mathcal{B}$, that is, $\Pi = \mathcal{L}(\mathcal{B})$.

Figure 7.2a shows a navigation scenario, where we need to perform surveillance of



(a)



(b)

Figure 7.2: The system is required to visit regions A and B infinitely often and avoid region X. This property is ω-regular and it is represented as a Büchi automaton in (b). The accepting state is q_3, and the initial states are q_1, q_2, and q_3.

the regions A and B (visit them infinitely often), while avoiding the region X. Let Ω represent the rest of the region (shown in white). Figure 7.2b shows a Büchi automaton corresponding to the surveillance objective. Each region is captured using a proposition. We start in state q_1, q_2, or q_3; we move to state q_4 if we encounter X, and never return to $\{q_1, q_2, q_3\}$. We move from q_1 to q_2 through the edge A and from q_2 to q_3 through the edge B. So, every time we visit q_3, we have paid at least one visit to both A and B. From state q_3 we return to q_1 to again visit each of A and B.

7.5 Strategies

A strategy specifies the control inputs to a transition system at different time points. More specifically, a *partial strategy* σ for the transition system $\mathcal{T}$ is a partial function $\sigma : Paths_f(\mathcal{T}) \to \mathcal{U}$ such that for a finite path $\zeta \in Paths_f(\mathcal{T})$ and $\zeta = s_0u_0s_1 \ldots u_{i-1}s_i, \sigma(\zeta) \in Enabled(s_i)$. A finite or infinite path $\zeta = s_0u_0s_1u_1s_2 \ldots$ is said to *conform* to a partial strategy σ, if for all i, $\sigma(s_0u_0 \ldots s_i) = u_i$. A *strategy* σ is a partial strategy for which $\sigma(\zeta)$ is defined for all finite paths ζ that conform with σ.

Let $Paths_\sigma(\mathcal{T}, s_0)$ denote the set of all infinite paths that conform to σ and start at state s_0. Let $Str(\mathcal{T})$ denote the set of all strategies for $\mathcal{T}$.

Definition 7.4. A strategy σ for the transition system $\mathcal{T}$ is *winning* for a state $s_0 \in \mathcal{S}$ with respect to a ω-regular property Π over the propositions $\mathcal{P}$, if $Tr(Paths_\sigma(\mathcal{T}, s_0)) \subseteq \Pi$.

Definition 7.4 is defined for strategy that allows for infinite paths. When the property is regular, only strategies that have no infinite paths conforming to it are of interest. Let $Str_f(\mathcal{T})$ denote the set of all strategies, which have no infinite paths conforming to them. Note that the lengths of the paths which conform to a strategy in $Str_f(\mathcal{T})$ could still be variable.

A finite path $\zeta = s_0u_0 \ldots s_k$ maximally conforms to a strategy $\sigma \in Str_f(\mathcal{T})$, if ζ conforms to σ and there is no extension, $\zeta' = s_0u_0 \ldots s_ku_ks_{k+1}$ of ζ, which conforms to σ. Let $Paths^m_\sigma(\mathcal{T}, s_0)$ denotes the maximally conforming finite paths of $\mathcal{T}$ with respect to σ starting at a state s_0.

To synthesize a strategy for $\mathcal{T}$ from a state $s_0 \in \mathcal{S}^{init}$ such that all maximal executions conforming to it reach a state in a set $\mathcal{S}_f \subseteq \mathcal{S}$, label the states in $\mathcal{S}_f$ with a unique proposition. Then, let the property Π be the set of all traces corresponding to paths, which start in $\mathcal{S}^{init}$ and end in $\mathcal{S}_f$, and do not visit $\mathcal{S}_f$ in the middle.

Therefore, for a regular property, a slightly different form of winning is considered.

Definition 7.5. A strategy σ for the transition system $\mathcal{T}$ is *winning* for a state $s_0 \in \mathcal{S}$ with respect to a regular property Π over the propositions $\mathcal{P}$, if $\sigma \in Str_f(\mathcal{T})$ and $Tr(Paths^m_\sigma(\mathcal{T}, s_0)) \subseteq \Pi$.

7.5.1 Cost of strategies

Two type of costs are considered for finite and infinite paths respectively.

The cost of a finite path is the sum of the weights on the individual edges. Given a finite path $\zeta = s_0 u_0 s_1 \dots s_k$, define

$$\mathcal{W}(\zeta) = \sum_{j=0}^{k-1} \mathcal{W}(s_j, u_j, s_{j+1}).$$

Consequently, the following proposition holds.

Proposition 7.1. *Given $\zeta = s_0 u_0 s_1 \dots s_k$ and $\zeta' = s'_0 u'_0 s'_1 \dots s'_k$, if $\mathcal{W}(s_j, u_j, s_{j+1}) \leq \mathcal{W}(s'_j, u'_j, s'_{j+1})$ for all j, then $\mathcal{W}(\zeta) \leq \mathcal{W}(\zeta')$.*

For infinite paths, we consider mean payoff costs as a generalization of average costs. Given an infinite path $\zeta = s_0 u_0 s_1 \dots$ of the transition system $\mathcal{T}$, define

$$\mathcal{W}(\zeta) = \lim_{N \to \infty} \frac{1}{N} \sum_{j=0}^{N-1} \mathcal{W}(s_j, u_j, s_{j+1}).$$

Similarly, the following proposition holds.

Proposition 7.2. *Given $\zeta = s_0 u_0 s_1 \dots$ and $\zeta' = s'_0 u'_0 s'_1 \dots$, if $\mathcal{W}(s_j, u_j, s_{j+1}) \leq \mathcal{W}(s'_j, u'_j, s'_{j+1})$ for all j, then $\mathcal{W}(\zeta) \leq \mathcal{W}(\zeta')$.*

This monotonicity property seems trivial, but plays an important role in the subsequent analysis. In fact, results in this thesis carry over for several other cost functions for paths such as average cost, maximum cost, or discounted sum. The analysis only relies on the fact that the cost of a path is monotonic with respect to the cost on the transitions. As such, for simplicity, we fix one of the definitions.

For a regular property, the cost of a strategy σ of the transition system $\mathcal{T}$ with respect to an initial state s_0 is defined as

$$\mathcal{W}(\mathcal{T},\sigma,s_0) = \sup\{\mathcal{W}(\zeta)\,|\,\zeta \in \mathit{Paths}^m_\sigma(\mathcal{T},s_0)\}.$$

Note that only the maximally conforming paths are considered. Accordingly, given a regular property Π over $\mathcal{P}$, the optimal cost of winning $\mathcal{T}$ from an initial state s_0 with respect to a property Π is defined as

$$\mathcal{W}(\mathcal{T},s_0,\Pi) = \inf\{\mathcal{W}(\mathcal{T},\sigma,s_0)\,|\,\sigma \in \mathit{Str}_f(\mathcal{T}), \mathit{Tr}(\mathit{Paths}^m_\sigma(\mathcal{T},s_0)) \subseteq \Pi\}.$$

Similarly, for a ω-regular property, the cost of a strategy σ of the transition system $\mathcal{T}$ with respect to an initial state s_0 is defined as

$$\mathcal{W}(\mathcal{T},\sigma,s_0) = \sup\{\mathcal{W}(\zeta)\,|\,\zeta \in \mathit{Paths}_\sigma(\mathcal{T},s_0)\}.$$

Thus, given a ω-regular property Π over $\mathcal{P}$, the optimal cost of winning $\mathcal{T}$ from an initial state s_0 with respect to a property Π is defined as

$$\mathcal{W}(\mathcal{T},s_0,\Pi) = \inf\{\mathcal{W}(\mathcal{T},\sigma,s_0)\,|\,\sigma \in \mathit{Str}(\mathcal{T}), \mathit{Tr}(\mathit{Paths}_\sigma(\mathcal{T},s_0)) \subseteq \Pi\}.$$

For both cases, the cost is taken to be infinity if the minimization is over an empty set. Denote an optimal strategy that achieves the optimal cost as $\sigma(\mathcal{T},s_0,\Pi)$. Note that the optimal strategy may not be unique. Furthermore, the optimal strategy does not exist if a given property is not satisfiable by the system.

7.6 Optimal Control Problem

Given the transition system $\mathcal{T}$, an initial state s_0, and a property Π, the optimal control problem is to compute an optimal winning strategy from s_0 with respect to Π, if it exists, and the optimal cost of winning.

Chapter 8

ABSTRACTION AND REFINEMENT

This chapter defines the preorder for optimal control that preserves the cost and presents the abstraction refinement procedure for constructing finite state systems, which simulate a given transition system.

8.1 Preorders for Optimal Control

In this section, a preorder on the class of transition systems is defined such that it preserves the optimal cost of winning. In other words, the optimal cost of winning in a system higher up in the ordering is an upper bound on the optimal cost of winning in a system below it. For this, the definition of alternating simulations [156] is extended to include costs.

Definition 8.1. Given two transition systems $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{S}_i^{init}, \mathcal{U}_i, \mathcal{P}, \Delta_i, \mathcal{L}_i, \mathcal{W}_i)$, for $i = 1, 2$, a simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$ is a pair of relations (α, β), where $\alpha \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ and $\beta \subseteq \mathcal{S}_1 \times \mathcal{U}_1 \times \mathcal{S}_2 \times \mathcal{U}_2$, such that:

1. $\forall\, (s_1, s_2) \in \alpha$, $\mathcal{L}_1(s_1) = \mathcal{L}_2(s_2)$.
2. $\forall\, s_1 \in \mathcal{S}_1^{init}$, $\exists\, s_2 \in \mathcal{S}_2^{init}$ such that $(s_1, s_2) \in \alpha$;
3. $\forall\, (s_1, s_2) \in \alpha$ and $u_2 \in Enabled(s_2)$, $\exists\, u_1 \in Enabled(s_1)$ such that:
    a) $(s_1, u_1, s_2, u_2) \in \beta$
    b) $\forall\, (s_1, u_1, s_1') \in \Delta_1, \exists\, (s_2, u_2, s_2') \in \Delta_2$ such that $(s_1', s_2') \in \alpha$ and $\mathcal{W}_1(s_1, u_1, s_1') \leq \mathcal{W}_2(s_2, u_2, s_2')$.

Let $\mathcal{T}_1 \preceq_{(\alpha,\beta)} \mathcal{T}_2$ denote that (α, β) is a simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$. If there exists some (α, β) such that $\mathcal{T}_1 \preceq_{(\alpha,\beta)} \mathcal{T}_2$, then $\mathcal{T}_2$ simulates $\mathcal{T}_1$, and it is denoted as $\mathcal{T}_1 \preceq \mathcal{T}_2$.

Theorem 8.1. *$\preceq$ is a preorder on the class of transition systems over a set of propositions $\mathcal{P}$.*

The next result shows that $\preceq$ is an ordering on the transition systems, which "preserves" optimal control with respect to a regular property.

Theorem 8.2. *Given two transition systems $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{S}_i^{init}, \mathcal{U}_i, \mathcal{P}, \Delta_i, \mathcal{L}_i, \mathcal{W}_i)$ for $i = 1, 2$, let Π be a regular property over a set of propositions $\mathcal{P}$, $\mathcal{T}_1 \preceq_{(\alpha,\beta)} \mathcal{T}_2$*

and $(s_0, s'_0) \in \alpha$ *for* $s_0 \in \mathcal{S}_1^{init}$ *and* $s'_0 \in \mathcal{S}_2^{init}$. *If there exists a winning strategy* σ_2 *for* $\mathcal{T}_2$ *from* s'_0 *with respect to* Π, *then there exists a winning strategy* σ_1 *for* $\mathcal{T}_1$ *from* s_0 *with respect to* Π *such that* $\mathcal{W}_1(\mathcal{T}_1, \sigma_1, s_0) \leq \mathcal{W}_2(\mathcal{T}_2, \sigma_2, s'_0)$. *Therefore,* $\mathcal{W}_1(\mathcal{T}_1, s_0, \Pi) \leq \mathcal{W}_2(\mathcal{T}_2, s'_0, \Pi)$.

Similarly, $\preceq$ is also an ordering on the transition systems, which "preserves" optimal control with respect to an ω-regular property.

Theorem 8.3. *Given two transition systems* $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{S}_i^{init}, \mathcal{U}_i, \mathcal{P}, \Delta_i, \mathcal{L}_i, \mathcal{W}_i)$ *for* $i = 1, 2$, *let* Π *be an* ω*-regular property over a set of propositions* $\mathcal{P}$, $\mathcal{T}_1 \preceq_{(\alpha,\beta)} \mathcal{T}_2$ *and* $(s_0, s'_0) \in \alpha$ *for* $s_0 \in \mathcal{S}_1^{init}$ *and* $s'_0 \in \mathcal{S}_2^{init}$. *If there exists a winning strategy* σ_2 *for* $\mathcal{T}_2$ *from* s'_0 *with respect to* Π, *then there exists a winning strategy* σ_1 *for* $\mathcal{T}_1$ *from* s_0 *with respect to* Π *such that* $\mathcal{W}_1(\mathcal{T}_1, \sigma_1, s_0) \leq \mathcal{W}_2(\mathcal{T}_2, \sigma_2, s'_0)$. *Therefore,* $\mathcal{W}_1(\mathcal{T}_1, s_0, \Pi) \leq \mathcal{W}_2(\mathcal{T}_2, s'_0, \Pi)$.

Theorems 8.2 and 8.3 allows for the construction of a strategy for the simulated system (*i.e.,* $\mathcal{T}_1$) given a strategy for the simulation (*i.e.,* $\mathcal{T}_2$). This theorem is therefore important for the algorithms presented in later chapters. The abstraction refinement procedure for constructing finite state systems, which simulate a given transition system is presented next.

8.2 Abstraction

During an abstraction, the state and input spaces are divided into finite number of parts, and they are used as symbolic states and inputs, respectively, in the abstract transition system. Henceforth, fix a transition system $\mathcal{T} = (\mathcal{S}, \mathcal{S}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$.

An abstraction function constructs an abstract transition system $Abs(\mathcal{T}, \equiv_S, \equiv_U)$ given the transition system $\mathcal{T}$, and two equivalence relations $\equiv_S$ and $\equiv_U$ on the state-space $\mathcal{S}$ and the input-space $\mathcal{U}$, respectively. To ensure a well defined abstract transition system, $\equiv_S$ on $\mathcal{S}$ needs to respect both the set of labels $\mathcal{L}$ and the set of initial states $\mathcal{S}^{init}$. In other words, the labels are the same for all equivalent states, and the initial states in the set $\mathcal{S}^{init}$ are not equivalent to any states outside of the set $\mathcal{S}^{init}$. More formally, an equivalence relation $\equiv_S$ on $\mathcal{S}$ respects $\mathcal{L}$, if for all $(s_1, s_2) \in \equiv_S$, $\mathcal{L}(s_1) = \mathcal{L}(s_2)$. Furthermore, an equivalence relation $\equiv_S$ on $\mathcal{S}$ respects $\mathcal{S}^{init}$, if for all $(s_1, s_2) \in \equiv_S$, where $s_1 \in \mathcal{S}^{init}$, we also have $s_2 \in \mathcal{S}^{init}$.

Definition 8.2. Let $\equiv_S \subseteq \mathcal{S} \times \mathcal{S}$ and $\equiv_U \subseteq \mathcal{U} \times \mathcal{U}$ be two equivalence relations of finite index such that $\equiv_S$ respects the labeling function $\mathcal{L}$ and the initial states $\mathcal{S}^{init}$. $Abs(\mathcal{T}, \equiv_S, \equiv_U) = (\mathcal{S}', \mathcal{S}^{init\prime}, \mathcal{U}', \mathcal{P}, \Delta', \mathcal{L}', \mathcal{W}')$, where

- $\mathcal{S}' = \{[s]_{\equiv_S} \mid s \in \mathcal{S}\}$ is the equivalence classes of $\equiv_S$.
- $\mathcal{S}^{init'} = \{[s]_{\equiv_S} \mid s \in \mathcal{S}^{init}\} \subseteq \mathcal{S}'$.
- $\mathcal{U}' = \{[u]_{\equiv_U} \mid u \in \mathcal{U}\}$ is the equivalence classes of $\equiv_U$.
- $\Delta' = \{(S_1, U, S_2) \mid \exists s \in S_1, s' \in S_2, u \in U,\ s.t.\ (s, u, s') \in \Delta\}$.
- For $S \in \mathcal{S}'$, $\mathcal{L}'(S) = \mathcal{L}(s)$ for any $s \in S$.
- For $(S_1, U, S_2) \in \Delta'$, $\mathcal{W}'(S_1, U, S_2) = \sup\{\mathcal{W}(s_1, u, s_2) \mid s_1 \in S_1, s_2 \in S_2, u \in U, (s_1, u, s_2) \in \Delta\}$.

Call $\mathcal{T}$ the concrete system and $Abs(\mathcal{T}, \equiv_S, \equiv_U)$ the abstract system. The next proposition states that the abstract system simulates the concrete system.

Proposition 8.1. *If $\mathcal{T}$ is a complete transition system, $\mathcal{T} \preceq Abs(\mathcal{T}, \equiv_S, \equiv_U)$.*

8.3 Refinement

A sequence of abstract systems, which are closer to the original system than their predecessors in the sequence can be constructed by choosing finer equivalence relations on the state and input spaces.

Definition 8.3. Let $\mathcal{T}_1$ and $\mathcal{T}_3$ be transition systems such that $\mathcal{T}_1 \preceq \mathcal{T}_3$. A transition system $\mathcal{T}_2$ is said to be a refinement of $\mathcal{T}_3$ with respect to $\mathcal{T}_1$, if $\mathcal{T}_1 \preceq \mathcal{T}_2 \preceq \mathcal{T}_3$.

The next proposition states that the finer equivalence relations on the state and input spaces will refine the abstract system.

Proposition 8.2. *Let $\equiv_S, \equiv'_S \subseteq \mathcal{S} \times \mathcal{S}$ and $\equiv_U, \equiv'_U \subseteq \mathcal{U} \times \mathcal{U}$ be equivalence relations of finite index such that $\equiv'_S \subseteq \equiv_S$ and $\equiv'_U \subseteq \equiv_U$. Then, $Abs(\mathcal{T}, \equiv'_S, \equiv'_U)$ is a refinement of $Abs(\mathcal{T}, \equiv_S, \equiv_U)$ with respect to $\mathcal{T}$.*

Given the abstraction refinement procedure, a dynamical system can be abstracted into a finite state transition system with a finer and finer grid until an optimal strategy for the finite state transition system can be computed. Once an optimal strategy is obtained, the optimal controller for the dynamical system can be extracted from the optimal strategy by Theorems 8.2 and 8.3. The next chapter discusses optimal controller synthesis techniques for regular objectives in more detail.

Chapter 9

OPTIMAL CONTROLLER SYNTHESIS FOR REGULAR OBJECTIVES

This chapter considers the optimal controller synthesis procedure for discrete time hybrid systems with regular objectives. Given a hybrid system described as a state transition system, an initial state, and a regular property, the optimal control problem is to compute an optimal winning strategy from the initial state with respect to the regular property, if it exists, and the optimal cost of winning. The optimal winning strategy is the optimal controller that can be applied on a system to ensure that the system's specifications are achieved with minimum cost.

This chapter begins with an overview of the overall abstraction refinement technique for synthesizing optimal controllers, and the main components of the framework. Then, the application of the technique on a hybrid system is presented with an analysis of the algorithm's convergence to the optimal cost. Lastly, two example systems are used to illustrate the implementation of the technique.

9.1 Controller Synthesis for Regular Objectives

In this section, given a weighted transition system $\mathcal{T}$, an initial state s_0, and a regular property Π, we seek to synthesize an optimal controller σ for $\mathcal{T}$. The first subsection discusses the overall abstraction refinement procedure and the main components of the framework. This framework is summarized in Algorithm 8. The next subsection discusses the main subroutine of Algorithm 1, namely, the optimal strategy synthesis procedure given a finite state transition system, an initial state, and a regular property.

9.1.1 Abstraction-Refinement Procedure (**OptCAR**)

For the rest of the section, fix the transition system $\mathcal{T} = (\mathcal{S}, \mathcal{S}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$, where $\mathcal{T}$ is an infinite state system. Note that since $\mathcal{T}$ is input deterministic, $\sigma(\mathcal{T}, s_0, \Pi)$ will correspond to a unique path starting from s_0. In general, solving for an optimal strategy is difficult because $\mathcal{T}$ is a infinite state system; therefore, we focus on synthesizing suboptimal strategies using Algorithm 8. As an overview, Algorithm 8 first partitions the state space into grids of a particular size, and constructs an abstract system $\hat{\mathcal{T}}$ using *ConsAbs* for the system $\mathcal{T}$. Then, it computes

Algorithm 8 OptCAR (Abstraction Refinement Procedure)

```
Input: System $\mathcal{T}$, regular property $\Pi$ as a finite state automaton, initial state $s_0$,
  and rational number $\varepsilon_0 > 0$, and maximum iterations $i_m$
  Set $\varepsilon := \varepsilon_0, i := 0$
  while $i < i_m$ do
      $\hat{\mathcal{T}}, \hat{x}_0 := ConsAbs(\mathcal{T}, s_0, \varepsilon)$
      $J, \hat{\sigma} := SolveFiniteGame(\hat{\mathcal{T}}, \hat{x}_0, \Pi)$
      $\sigma := ExtractController(\hat{\sigma}, \hat{\mathcal{T}}, \mathcal{T})$
      Output $\sigma$ and $J_\varepsilon$
      $\varepsilon := \frac{\varepsilon}{2}$
      $i := i + 1$
  end while
```

the optimal cost J and strategy of the abstract system through a two-player game *SolveFiniteGame*. A suboptimal strategy for $\mathcal{T}$ can then be extracted from the strategy of the abstract system $\hat{\mathcal{T}}$ with the cost upper bounded by J using the procedure *ExtractController*. If the upper bound J is not zero, refine the state space partitions using smaller grids, and repeat the whole process to reduce the cost J. As a result, this algorithm outputs a sequence of suboptimal strategies, whose costs converge to that of the optimal cost.

More precisely, in each iteration, Algorithm 8 performs the following sequence of steps. First, it constructs a finite state abstraction $\hat{\mathcal{T}}$ of $\mathcal{T}$ using the function $ConsAbs(\mathcal{T}, \varepsilon)$. The function $ConsAbs(\mathcal{T}, \varepsilon)$ outputs $Abs(\mathcal{T}, \equiv^\varepsilon_S, \equiv^\varepsilon_U)$, where $\equiv^\varepsilon_S$ and $\equiv^\varepsilon_U$ are equivalence relations whose equivalences classes are the elements of $Grid(\mathcal{S}, \varepsilon)$ and $Grid(\mathcal{U}, \varepsilon)$, respectively. Define the initial abstract state as $\hat{s}_0 := [s_0]_{\equiv^\varepsilon_S}$. This step solves $|\mathcal{S}|^2|\mathcal{U}|$ optimizations, where $|\mathcal{S}|$ is the number of states in $\hat{\mathcal{T}}$ and $|\mathcal{U}|$ is the number of control inputs in $\hat{\mathcal{T}}$. These optimizations can be computed in parallel. Next, $SolveFiniteGame(\hat{\mathcal{T}}, \hat{x}_0, \Pi)$ computes the optimal cost of winning $J = \mathcal{W}(\hat{\mathcal{T}}, \hat{x}_0, \Pi)$ with respect to Π in the finite state transition system $\hat{\mathcal{T}}$ and the corresponding strategy $\hat{\sigma} = \sigma(\hat{\mathcal{T}}, \hat{x}_0, \Pi)$ for $\hat{\mathcal{T}}$ through a two-player game, discussed more in the next section (see Algorithm 9).

Finally, $ExtractController(\hat{\sigma}, \hat{\mathcal{T}}, \mathcal{D})$ outputs a suboptimal strategy/controller σ whose cost is bounded by the optimal cost J for the abstract system. The existence of σ given $\hat{\sigma}$ is guaranteed by Theorem 8.2. Essentially, σ provides the sequence of inputs u for the system as required by the regular property Π. To illustrate the relationship between σ and $\hat{\sigma}$, let $u_0, u_1, \ldots, u_{t-1}$ be the inputs, which have been computed, and let $s_0, s_1, \ldots, s_t$ be the sequence of state generated by the inputs. The

t-th control input u_t is obtained by finding the minimum cost transition (s_t, u_t, s_{t+1}), where $u_t \in U$ and $s_{t+1} \in \mathcal{S}$. The set $\mathcal{U}$ is defined as $\mathcal{U} = \hat{\sigma}([s_0]_{\equiv^{\varepsilon}_{\mathcal{S}}}[u_0]_{\equiv^{\varepsilon}_{U}} \dots [s_t]_{\equiv^{\varepsilon}_{\mathcal{S}}})$, and $\mathcal{S}'$ is the union of all $\mathcal{S}''$ such that $([s_t]_{\equiv^{\varepsilon}_{\mathcal{S}}}, \mathcal{U}, \mathcal{S}'')$ is a transition of $\hat{\mathcal{T}}$. The inputs u_t can be computed by solving a linear program when the cost function and the transitions are linear and the equivalence classes are polyhedral sets.

This algorithm outputs a sequence of suboptimal strategies, whose costs are non-increasing over the iterations. In the beginning of the procedure, when the partitioning is coarse, a winning strategy $\hat{\sigma}$ might not exist even if the underlying system $\mathcal{T}$ has an optimal solution. However, if one continues to refine the grid, a winning strategy will exist if $\mathcal{T}$ has an optimal solution. In fact, a more refined partitioning will return a controller with trajectory cost that is no worse than the trajectory cost resulting from the controller corresponding to the winning strategy for the current partitioning. Thus, the algorithm can be terminated at a specific iteration based on the application and the computational resources. Furthermore, for a special class of systems that has a robust optimal path, we show that the suboptimal cost converges to the optimal cost as iteration index i goes to infinity. See Section 9.2.3 for details.

Algorithm 8 can be instantiated to any class of hybrid systems. However, the computational complexity of the optimization problems that will need to be solved in the construction of the abstract system and the extraction of a winning strategy will depend on the class of dynamics and the type of the cost function. For a piecewise linear system with linear cost function, the maximization during the abstraction procedure is a linear program, because the partitions of $\equiv_{\mathcal{S}}$ and $\equiv_U$ are polyhedral sets (grid elements). If computation resources are limited, the best suboptimal controller found with respect to the cost upper bound J is guaranteed to generate a trajectory that satisfies the properties Π and has cost no greater than J.

9.1.2 Synthesizing Optimal Strategy

This section presents a value iteration scheme for computing the optimal cost and optimal strategy for a finite transition system $\mathcal{T}$ with respect to a regular property Π performed by the subroutine *SolveFiniteGame* in Algorithm 8. We will call a strategy that has a linear structure a *layered strategy*. In other words, there are no paths in the abstract system of length greater than the number of states in the system that conform with the strategy. This section presents the algorithm for computing an optimal strategy for a finite state transition system that is layered. The algorithm is given in Algorithm 9, which is a modified Bellman-Ford algorithm [157].

Algorithm 9 *SolveFiniteGame* (Two-Player Games)

```
Input: Finite state transition system $\mathcal{T}_S$, regular property $\Pi$ specified as $(\mathcal{T}_P, P_f)$
  $\mathcal{T}, S_f := ReduceReach(\mathcal{T}_S, \mathcal{T}_P, P_f)$
  Set for every $s \in \mathcal{S} - S_f$, $C(s) := 0$ if $s \in S_f$ and $\infty$ otherwise
  for $i = 1, \ldots, |\mathcal{S}|$ do
      for $s \in \mathcal{S}$ do
          $C^i(s) := \min_{u \in \mathcal{U}} \max_{(s,u,s') \in \Delta} (\mathcal{W}(s,u,s') + C^{i-1}(s'))$
          $\sigma^i(s) := \arg\min_{u \in \mathcal{U}} \max_{(s,u,s') \in \Delta} (\mathcal{W}(s,u,s') + C^{i-1}(s'))$
      end for
  end for
  if $C^{|S|}(s_0) < \infty$ then
      Output the strategy $\sigma^{|S|}$ and the cost $C^{|S|}(s_0)$
  end if
```

The function *ReduceReach* reduces the problem of computing the layered strategy for a regular property Π to that of reachability. It consists of taking a product of the input transition system $\mathcal{T}_S$ and the transition system $\mathcal{T}_P$ of the property. More formally, given the input transition system $\mathcal{T}_S$ and the transition system $\mathcal{T}_P$ of the property, the product transition system returned by *ReduceReach* is defined as follows.

Definition 9.1. Let $\mathcal{T}_S = (\mathcal{S}_S, \mathcal{S}_S^{init}, \mathcal{U}_S, \mathcal{P}, \Delta_S, \mathcal{L}_S, \mathcal{W}_S)$ be a state transition system, and $\mathcal{T}_P = (\mathcal{S}_P, \mathcal{S}_P^{init}, \mathcal{U}_P, \mathcal{P}, \Delta_P, \mathcal{L}_P, \mathcal{W}_P)$ be the automation that represents the regular property. Then, the product transition system is $\mathcal{T} = (\mathcal{S}, \mathcal{S}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$, where

- $\mathcal{S} = \{(s_1, s_2) \in \mathcal{S}_S \times \mathcal{S}_P \mid \mathcal{L}_S(s_1) = \mathcal{L}_P(s_2)\} \cup \{s_d\}$, where s_d is a dead state;
- $\mathcal{S}^{init} = \mathcal{S}_S^{init} \times \mathcal{S}_P^{init}$;
- $\mathcal{U} = \mathcal{U}_S$;
- $\mathcal{P}$ is the same for both $\mathcal{T}_S$ and $\mathcal{T}_P$. The final states of $\mathcal{T}_P$ is denoted by a proposition $P_f \in \mathcal{P}$;
- $\Delta = \Delta_1 \cup \Delta_2$, where $\Delta_1 = \{((s_1, s_2), u, (s_1', s_2')) \in \mathcal{S} \times \mathcal{U} \times (\mathcal{S} \setminus \{s_d\}) \mid (s_1, u, s_1') \in \Delta_S, (s_2, a, s_2') \in \Delta_P$ for some $a\}$ and $\Delta_2 = \{((s_1, s_2), u, s_d) \in \mathcal{S} \times U \times \{s_d\}\}$ such that there exists $(s_1, u, s_1') \in \Delta_S$ for some s_1' and there does not exist a and s_2' such that $(s_2, a, s_2') \in \Delta_P$ and $\mathcal{L}_S(s_1') = \mathcal{L}_P(s_2')$;
- $\mathcal{L}(s) = \mathcal{L}_S(s)$ for $s \in \mathcal{S}_S$;
- $\mathcal{W}((s_1, s_2), u, (s_1', s_2')) = \mathcal{W}_S(s_1, u, s_2)$.

Furthermore, the set of final states S_f of $\mathcal{T}$ with respect with reachability is solved as $S_f = \{(s_1, s_2) \in (\mathcal{S}\backslash\{s_d\}) \times (\mathcal{S}\backslash\{s_d\}) \mid \mathcal{L}_P(s_2) = P_f\}$.

The algorithm initially assigns a cost of 0 to the states in S_f and ∞ otherwise. The cost C^i in the i-iteration captures the optimal cost of reaching S_f by a strategy in which all paths that conform to it have length at most i, and σ^i stores a corresponding strategy. Therefore, $C^{|\mathcal{S}|}$ provides a layered strategy if $C^{|\mathcal{S}|}(s_0) < \infty$. The algorithm can be improved, wherein it terminates earlier than completing the $|\mathcal{S}|$ iterations, if the costs C do not change between iterations. In the worst case, this algorithm runs in $O(|\Delta||\mathcal{S}|)$ time, where $|\Delta|$ is the number of transitions in $\mathcal{T}$ and $|\mathcal{S}|$ is the number of states in $\mathcal{T}$.

Next, we will describes how to implement Algorithms 8 and 9 to synthesize an optimal controller for a class of hybrid systems, and show that Algorithm 8 will converge to the optimal cost with refinement.

9.2 Controller Synthesis for Hybrid Systems

This section considers an optimal control problem for discrete-time hybrid systems. The abstraction refinement approach is applied to construct a series of controllers with improving suboptimal costs that converge to the optimal cost under the existence of a robust optimal control.

9.2.1 Problem Formulation

A discrete-time hybrid system is a tuple $(\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \{f_i, P_i)\}_{i\in[m]}, \mathcal{L}_d, \mathcal{J})$, where the state-space $\mathcal{X} \subseteq \mathbb{R}^n$ and the input-space $\mathcal{U} \subseteq \mathbb{R}^p$ are compact sets, $\mathcal{X}^{init} \subseteq \mathcal{X}$ is the set of initial states, $\mathcal{P}$ is a finite set of propositions, f_i is a Lipschitz continuous function of $x \in \mathcal{X}$ and $u \in \mathcal{U}$, and P_i is a polyhedral set, such that $\{P_i\}_{i\in[m]}$ is a polyhedral partition of $\mathcal{X}$, $\mathcal{L}_d : [m] \to \mathcal{P}$ is a labeling function and $\mathcal{J} : \mathcal{X}\times\mathcal{U} \to \mathbb{R}_+$ is a Lipschitz continuous cost function. Note that f_i can be the same for different i. We associate a unique label to each region P_i. We could have assigned different labels to different regions in some polyhedral partition of P_i; we do not lose expressiveness here, since the latter can be transformed to the former problem by considering a finer partition whose regions are the regions partitioning each P_i according to the label.

Given an initial state $x_0 \in \mathcal{X}^{init}$ and a sequence of control inputs $\mathbf{u} = \{u_t\}_{t\in[k]}$, where $u_t \in \mathcal{U}$, $\phi(x_0, \mathbf{u}) = \{x_t\}_{t\in[k+1]}$ is the sequence of states visited under the control $\mathbf{u}$, where $x_{t+1} = f_t(x_t, u_t)$, and $f_t = f_i$ if $x_t \in P_i$. The cost of the sequence $\phi(x_0, \mathbf{u})$,

$\mathcal{J}(\phi(x_0, \mathbf{u}))$, is given by $\sum_{t\in[k]} \mathcal{J}(x_{t+1}, u_t)$. We define the partition sequence of $\{x_t\}_{t\in[k+1]}$, denoted $PS(\{x_t\}_{t\in[k+1]})$, to be the sequence of partitions visited by the states, namely, $P_{i_1}, \ldots, P_{i_{k+1}}$ such that $x_t \in P_{i_t}$ for all $t \in [k+1]$.

Problem 9.1 (Optimal control problem)**.** Given an n-dimensional discrete-time hybrid system $\mathcal{D} = (\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \{(f_i, P_i)\}_{i\in[m]}, \mathcal{L}_d, \mathcal{J})$, a state $x_0^* \in \mathcal{X}^{init}$ and a regular property Π over $\mathcal{P}$, find a sequence of control inputs $\mathbf{u}^*$ for which $\mathcal{L}_d(\phi(x_0^*, \mathbf{u}^*)) \in \Pi$ and $\mathcal{J}(\phi(x_0^*, \mathbf{u}^*))$ is minimized.

Remark. Although the regular property Π is specified over finite sequences, it could potentially contain finite sequences of unbounded lengths (*i.e.,* no fixed upper bound on the sequence length). Therefore, the Problem 9.1 is not the same as a classical finite horizon problem, because the optimal control sequence length is not fixed a priori.

9.2.2 Equivalent Optimal Strategy Problem

A discrete-time hybrid system $\mathcal{D} = (\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \{(f_i, P_i)\}_{i\in[m]}, \mathcal{L}_d, \mathcal{J})$ can be represented as a weighted transition system, $\mathcal{T}_{\mathcal{D}} = (\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$, where $\Delta = \{(x, u, x') \in \mathcal{X} \times \mathcal{U} \times \mathcal{X} \mid x' = f(x, u),$ where $f = f_i$ for $x \in P_i\}$, $\mathcal{L}(x) = \mathcal{L}_d(i)$, where $x \in P_i$, and $\mathcal{W}(x, u, x') = \mathcal{J}(x', u)$. Consequently, Problem 9.1 is equivalent to the following problem:

Problem 9.2 (Optimal strategy problem)**.**
Given a weighted transition system $\mathcal{T}_{\mathcal{D}} = (\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$, a state $x_0^* \in \mathcal{X}^{init}$ and a regular property Π over $\mathcal{P}$, find an optimal winning strategy $\sigma(\mathcal{T}_{\mathcal{D}}, x_0^*, \Pi)$ for which the optimal cost of winning $\mathcal{T}_{\mathcal{D}}$ with respect to Π, $\mathcal{W}(\mathcal{T}_{\mathcal{D}}, x_0^*, \Pi)$, is achieved.

9.2.3 Analysis of Algorithm 8 for Problem 9.2

This section analyzes the output of Algorithm 8, and shows that the suboptimal cost converges to the optimal cost if a robust optimal strategy exists. Note that even without the existence of a robust optimal strategy, we can still guarantee that the costs due to refinement are non-increasing.

Definition 9.2. Given the system defined in Problem 9.1, let $\mathbf{u}$ be the input sequence that results in a trajectory $\phi(x_0, \mathbf{u}) = \{x_t\}_{t\in[k+1]}$ for an initial state x_0, where $PS(\phi(x_0, \mathbf{u})) = \{P_{i_t}\}_{t\in[k+1]}$. This input sequence $\mathbf{u}$ is said to be *robust* with respect to the initial state x_0 if there exists constants $\varepsilon_t > 0$ such that $\mathcal{B}_{\varepsilon_t}(x_t) \subseteq P_{i_t}$ for all $t \in [k+1]$.

This definition of robustness is different from the typical robust control's definition. In robust control, the robustness is defined with respect to system uncertainties or external disturbances. Here, the system is deterministic. Instead, this definition concerns with an input sequence that will result in a state trajectory, which does not land on the boundary of polyhedral partitions of the state space $\mathcal{X}$. In other words, the state label will not change if a state is slightly perturbed.

Let us denote the elements in the iteration of Algorithm 8 corresponding to a particular ε as $\hat{\mathcal{T}}_\varepsilon$ for $\hat{\mathcal{T}}$, $\hat{x}_0^\varepsilon$ for $\hat{x}_0$, J_ε for J, $\hat{\sigma}_\varepsilon$ for $\hat{\sigma}$, and σ_ε for $\sigma_{\mathcal{D}}$.

Henceforth, let $\mathbf{u}^* = \{u_t^*\}_{t\in[k]}$ be a robust optimal control input sequence with respect to the initial state x_0^* and $\zeta^* = \phi(x_0^*, \mathbf{u}^*) = \{x_t^*\}_{t\in[k+1]}$ be the corresponding optimal trajectory for Problem 9.1. The proof also requires a special kind of strategy that ensures that there is a unique path conforming to this strategy, by choosing inputs that result in exactly one successor state (see Figure 9.1).

Definition 9.3. A **chain strategy** for a transition system $\mathcal{T}$ and an initial state s_0 is a strategy $\sigma \in Str_f(\mathcal{T})$ such that there is one path in $Paths_\sigma^m(\mathcal{T}, s_0)$.

Next, we present the main theorem of this chapter.

Theorem 9.1. *If there exists a optimal control* $\mathbf{u}^*$ *that is robust with respect to the initial state* x_0 *for Problem 9.1, the sequence of sub-optimal costs* $\{J_{\varepsilon_0/2^i}\}_{i\in\mathbb{Z}_+}$ *output by Algorithm 8 converges to the optimal cost* $J_{opt} = \mathcal{W}(\mathcal{T}_{\mathcal{D}}, x_0, \Pi)$*. Furthermore, for each sub-optimal cost* $J_{\varepsilon_0/2^i}$*, there exists a suboptimal winning strategy* $\sigma_{\varepsilon_0/2^i}$ *with cost no larger than* $J_{\varepsilon_0/2^i}$*.*

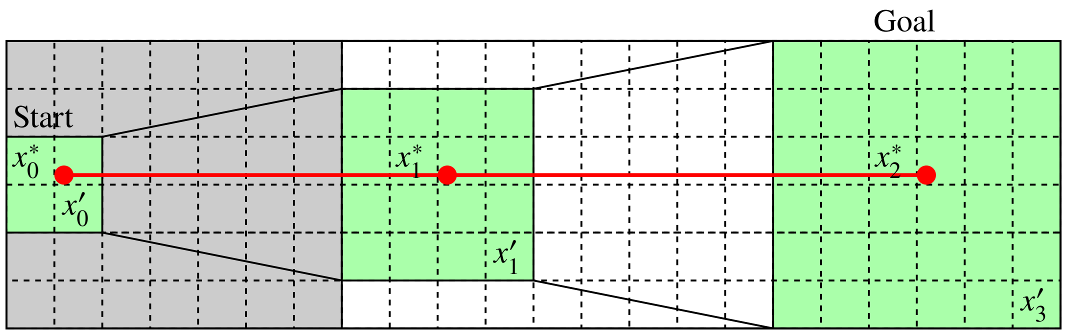


Figure 9.1: An illustration of chain strategy and refinement. The domain is separated into two areas (gray and white), where two different dynamics apply. The red dots are the optimal path.

Proof. A sketch of the proof is provided. The full proof is provided in Appendix C.

First, let $M_x > \varepsilon_x > 0$ and $M_u > \varepsilon_u > 0$. Let $x_0 \in \mathcal{B}_{\varepsilon_x}(x_0^*)$, $u_t \in \mathcal{B}_{\varepsilon_u}(u_t^*)$ $\forall t \in [k]$, where $\mathbf{u} = \{u_t\}_{t\in[k]}$ and $\zeta = \phi(x_0, \mathbf{u}) = \{x_t\}_{t\in[k+1]}$. Then, for any trajectory whose initial state and inputs have a bounded deviation from that of the optimal trajectory, the error between any of those trajectories is bounded due to continuity of the system. As a result, the suboptimal cost is bounded and decreases to zero if ε_x and ε_u decrease to zero because the cost function in Problem 9.2 is also continuous.

Next, given a specific cost sub-optimality, a chain strategy that satisfies a certain cost error bound exists. More specifically, a neighborhood N_t^x is constructed around each x_t^* and N_t^u around u_t^* such that all the transitions from N_t^x on N_t^u will end in N_{t+1}^x. This construction exists by the robustness of the optimal control as defined in Definition 9.2 and the fact that the system is continuous. Under this construction, all executions from N_0^x will land in N_t^x after t steps. This chain of neighborhoods gives a chain strategy. See Figure 9.1 for an illustration of the chain strategy. To ensure that the cost of the strategy is within $\delta > 0$ of the optimal cost, choose the N_t^x and N_t^u to be contained in some $M_x > \varepsilon_x > 0$ and $M_u > \varepsilon_u > 0$ balls. These two observations ensure that $|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq \delta$ for any path ζ starting in an ε_x ball around x_0^*.

In addition, choose the N_j^x and N_j^u in such a way that they correspond to an element of an $\varepsilon_0/2^i$ grid for some i (not necessarily the same i for all neighborhoods). Define $\equiv_X$ and $\equiv_U$ such that the N_j^x and N_j^u are all equivalence classes of $\mathcal{X}$ and $\mathcal{U}$, respectively. Note that we need to ensure that for any i, j, N_j^x is the same as N_i^x or the two are disjoint, and, a similar condition for N_j^u holds. This condition can be easily ensured during the construction by picking small enough ε_x and ε_u.

Now, assume that every N_t^x corresponds to an element of $Grid(\mathcal{X}, \varepsilon_0/2^{i_t})$ for some i_t, and similarly, N_t^u corresponds to an element of $Grid(\mathcal{U}, \varepsilon_0/2^{j_t})$ for some j_t. Let i be the maximum of the i_ts and j_ts. Note that $Grid(\mathcal{X}, \varepsilon_0/2^i)$ refines N_t^x and similarly, $Grid(\mathcal{U}, \varepsilon_0/2^i)$ refines N_t^u. In Figure 9.1, the squares around x_t^* with bold borders are N_t^x, and the dashed squares, which are contained in them correspond to the refined partition. One can define a strategy σ_ε (not necessarily a chain anymore) for $\hat{\mathcal{T}}_\varepsilon$, which correspond to following the neighborhoods N_t^x. Thus, all the paths in $\hat{\mathcal{T}}_\varepsilon$ which conform to σ_ε will be contained in the neighborhoods N_t^x. Therefore, the cost of σ_ε is bounded by that of σ, which is at most δ away from the optimal cost, and the optimal cost of $\hat{\mathcal{T}}_\varepsilon$ is at most δ away from that of $\mathcal{T}_\mathcal{D}$.

Observe that $J_{\varepsilon_0/2^i} \leq J_{\varepsilon_0/2^j}$ for all $i > j$. Furthermore, for any $\delta > 0$, there exists $\varepsilon = \varepsilon_0/2^i$, such that $|\mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi) - \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0^*, \Pi)| \leq \delta$. Note $J_\varepsilon = \mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi)$ and $J_{opt} = \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0^*, \Pi)$ is the optimal cost. As such, $|J_\varepsilon - J_{opt}| \leq \delta$, and $J_{\varepsilon_0/2^i}$ converges to J_{opt} as i goes to infinity. Lastly, for each sub-optimal cost $J_{\varepsilon_0/2^i}$, there exists a suboptimal winning strategy $\sigma_{\varepsilon_0/2^i}$ with cost no larger than $J_{\varepsilon_0/2^i}$. □

At this point, we have shown that the strategy given by OptCAR incurs a suboptimal cost that converges to the optimal cost of $\mathcal{D}$. The strategies used in the proof of Theorem 9.1 have the property that the length of the maximal paths, which conform to the strategy are finite and have a bound (in fact, they are all of the same length). Furthermore, the trace of all the paths is the same. However, during implementation, Algorithm 8 may return a sequence of suboptimal strategies $\sigma_{\varepsilon_0/2^i}$ that results in paths with different lengths because there could be paths with different lengths that have the same cost upper bound. Nonetheless, the cost of each path results from $\sigma_{\varepsilon_0/2^i}$ is bounded by the cost $J_{\varepsilon_0/2^i}$.

In addition, the strategy that is considered in the proof of Theorem 9.1 gives a sequence of inputs which satisfy the property Π from any point in an open neighborhood around the given initial state x_0^*. Furthermore, there is an open neighborhood around each of the control inputs such that the resulting paths satisfy Π. Thus, Algorithm 8 in fact returns a controller that is robust against input uncertainties under the assumption that the original system has such an optimal control.

9.3 Implementation

Algorithm 8 and 9 are implemented in the tool OptCAR in Python 2.7. A Python package, NetworkX, is used to represent the graph structures that arise in solving Algorithm 9, and the Parma Polyhedra Library [158] is used to represent the polyhedral sets that arise in the state space gridding and to solve the linear program problem that arises in the weight computation. OptCAR is tested on a linear dynamical system and a piecewise linear system on a MacBook Pro 8.2, 4 core Intel Core i7 processor with speed 2200 Hz, and 8GB RAM.

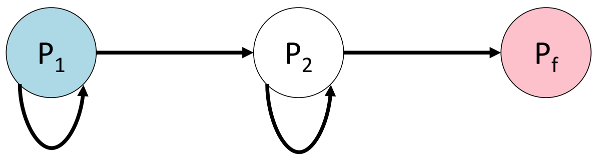


Figure 9.2: Automaton that represents the propositions of the two examples, where P_1 is the pink region, P_2 is the white region, and P_f is the light blue region.

9.3.1 Linear Dynamical System

The following linear dynamical system example is obtained from [159]:

$$x_{t+1} = Ax_t + Bu_t \tag{9.1}$$

$$A = \begin{bmatrix} 0.68 & -0.14 \\ 0.14 & 0.68 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0.1 \end{bmatrix},$$

where $x_t = (x_t^1, x_t^2) \in [-1, 1]^2$, and $u_t \in [-1, 1]$.

The cost function is $\mathcal{J}(\phi(x_0, u)) = \sum_{t\in[k]} \|u_t\|_2^2$. This cost is approximated as $\sum_{t\in[k]} \|u_t\|_1$ during implementation of OptCAR. The goal is to drive the system from an initial point $x_0 = (0.9, 0.9)$ to a final zone defined by a box at the origin, $P_f = \{x \mid \|x\|_\infty \leq \frac{1}{5}\}$. The propositions of this example are represented in Figure 9.2. The algorithm is implemented on two uniform grids on the states: sizes 20×20 and 40×40. The input, u, is partitioned into 5 uniform intervals.

Strategies obtained from OptCAR are compared with the strategy given by linear quadratic regulator (LQR) in Table 9.1. For a linear system, LQR is always a superior technique in comparison to OptCAR because the computation is significantly more efficient. The goal of this example is to illustrate that in an example with known optimal controller, the strategies given by OptCAR approximates the optimal control of LQR very closely, and it improves with refinement. Figure 9.3 shows the state trajectory for the three cases.

Table 9.1: Performance of OptCAR and LQR for System (9.1). [1]

Grid	20×20	40×40	LQR
Computation time (seconds)	355.82	5212.91	0.04
Optimal cost	0.5	0	0
Optimal step	6	6	6
Final point	(-0.0468,0.1499)	(-0.0468,0.1999)	(-0.0468,0.1999)

[1] 20×20 and 40×40 are optimal controller synthesis using OptCAR for two different uniform grids. Computation time is the time a method takes to compute the optimal strategy. Optimal step is the total number of steps that the optimal path takes to reach the goal region. The final point is where the optimal path ends in the goal region.

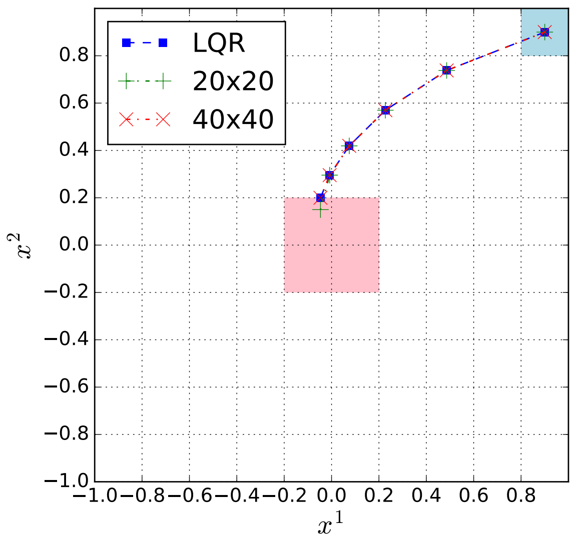


Figure 9.3: Simulated result of OptCAR and LQR for System (9.1).

9.3.2 Two-tank System

A two-tank system from [160] is used as an example of a piecewise linear system (Figure 9.4). The water can flow in between the two tanks through a pipe that connects them. The pipe is located at level 0.2. Tank 1 (left) has an inflow of water that is managed by a controller, and tank 2 (right) has an outflow of water that is fixed. The goal of the controller is to fill up tank 2 to level 0.4 from an initially low water level 0.1 using as small amount of water as possible from the source above tank 1. The goal will be made precise after the system is described formally next.

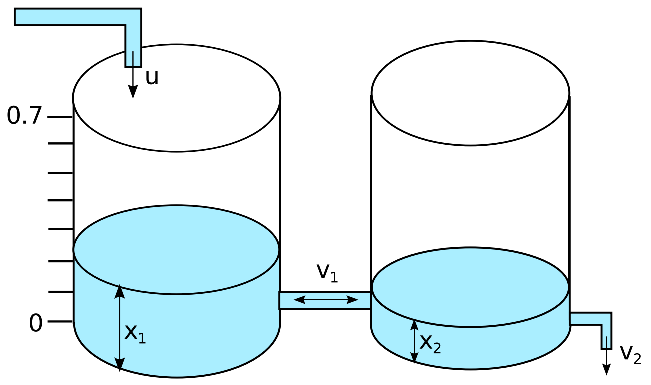
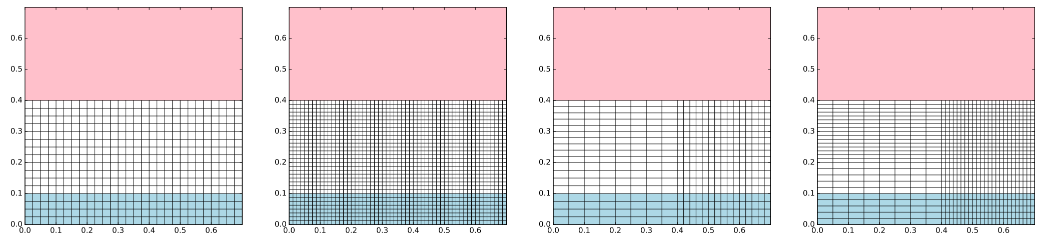


Figure 9.4: A schematic of a two-tank system.



(a) Uniform 28×17 (b) Uniform 56×33 (c) Non-uniform 23×17(d) Non-uniform 38×25

Figure 9.5: Partitions for the two-tank system. The goal region does not need to be partitioned because the transitions within the goal region are irrelevant.

The two-tank system has a linearized dynamics given by

$$x_{t+1} = Ax_t + Bu_t \tag{9.2}$$

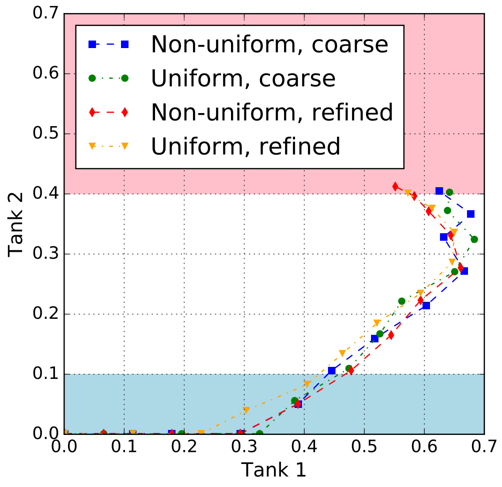
$$A = \begin{cases} A_1 & x \in [0, 0.2]^2 \\ A_2 & \text{otherwise} \end{cases} \quad B = \begin{bmatrix} 342.6753 \\ 0 \end{bmatrix},$$

where

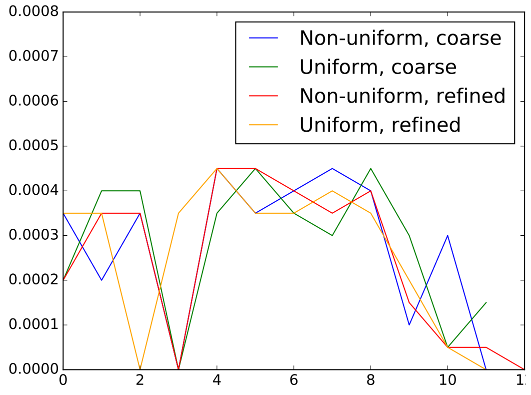
$$A_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.9635 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0.8281 & 0.1719 \\ 0.1719 & 0.7196 \end{bmatrix},$$

$x_t = (x_t^1, x_t^2) \in [0, 0.7]^2$, and $u_t \in [0, 0.0005]$. The water level in tank 1 at time t is x_t^1, and the water level in tank 2 at time t is x_t^2. The cost function is chosen to be $\mathcal{J}(\phi(x_0, u)) = \sum_{t \in [k]} \|u_t\|_1$ to represent minimal water inflow, and the goal is to drive the system from partition, $[0, 0.7] \times [0, 0.1]$, to partition $[0, 0.7] \times [0.4, 0.7]$. The propositions of this example are represented in Figure 9.2.

The algorithm is implemented on two uniform grids on the states — a 28×17 (coarse) grid and a 56×33 (refined) grid, and two non-uniform grids — a 23×17 (coarse) grid and a 32×25 (refined) grid. Figure 9.5 illustrates the grids. The input, u, is partitioned into 10 uniform intervals. The goal region is represented as one partition for all cases. This choice of goal representation speeds up computation time, and does not change the results in Section 9.2.3. Once a path arrives at the goal region, the path ends. Thus, the goal region does not need to be partitioned because the transitions within the goal region are irrelevant. In addition, the partitions' sizes for a non-uniform grid do not necessary have to be the same. Partitions whereby the transitions are more likely to be far can be larger because the states most likely will not end up at the neighboring partitions if the partitions are small. Another example of non-uniform grids with the same principle would be to have finer grids



(a) State trajectory



(b) Input trajectory

Figure 9.6: State trajectory and control input generated by the controller from OptCAR for System (9.2).

near the goal and coarser grids away from the goal. Such modification is feasible if the control engineer has prior information about the system from his/her past experiences. These modifications reduce computation time, and also allow for finer grids at regions that matter to achieve a better result.

Strategies obtained from OptCAR are compared in Table 9.2. Figure 9.6 shows the state trajectory and control input for the four cases, all start from $(0.001, 0.001)$. This example shows that choosing a suitable partition can reduce the computation time dramatically while still achieving comparable performance to the performance of a naive uniform grid. Therefore, future extension of this technique includes designing an intelligent scheme to partition the domain such that computation time is reduced. Lastly, about $60\% - 70\%$ of the computation time is used to construct the abstraction (*i.e.*, *ConsAbs* step in Algorithm 8) in which the computations can be parallelized

Table 9.2: Performance of OptCAR using Different Grids for System (9.2). [2]

Grid	28×17	56×33	23×17	38×25
Computation time (seconds)	1234.53	22119.36	1057.43	4309.04
Optimal cost	0.00340	0.00320	0.00335	0.00320
Optimal step	12	12	12	13
Final point	(0.642,0.402)	(0.573,0.401)	(0.625,0.405)	(0.552,0.412)

[2] The first two columns are results for uniform grids. The last two columns are results for non-uniform grids. The grids are shown in Figure 9.5. Computation time is the time OptCAR takes to compute the optimal strategy. Optimal step is the total number of steps that the optimal path takes to reach the goal region. The final point is where the optimal path ends in the goal region.

easily to decrease computation time.

9.4 Summary

This chapter considered the problem of synthesizing optimal control strategies for discrete-time hybrid system with respect to regular properties. An abstraction-refinement approach is presented for constructing arbitrarily precise approximations of the optimal cost and the corresponding strategies. This approach computes a sequence of suboptimal controller that converges to the optimal controller with refinement. The resulting suboptimal controller generates trajectories that incur cost no greater than the optimal cost of the corresponding abstract system. The abstraction based approach can be applied to the general class of hybrid systems; however, the challenge is in computing edges and weights, especially, for non-linear dynamics and in continuous time.

Future work will include extending the technique to more complex dynamics and continuous-time hybrid systems. To reduce computation time, a more intelligent gridding scheme in the refinement or the abstraction step will be developed, for example, [161], [162]. Lastly, the neighborhoods of states and inputs in the abstract system naturally model measurement errors and input uncertainties of the concrete system. Thus, a potential future application of this technique is in synthesizing robust optimal control for a hybrid system.

The next chapter extends the cost preserving abstraction technique from regular properties to ω-regular properties.

Chapter 10

OPTIMAL CONTROLLER SYNTHESIS FOR ω-REGULAR OBJECTIVES

This chapter considers the optimal controller synthesis procedure for discrete time hybrid systems with ω-regular objectives. Given a hybrid system described as a state transition system, an initial state, and a ω-regular property, the optimal control problem is to compute an optimal winning strategy from the initial state with respect to the ω-regular property, if it exists, and the optimal cost of winning. The optimal winning strategy is the optimal controller that can be applied to a system to ensure that the system's specifications are achieved with minimum cost.

Similar to Chapter 9, this chapter begins with an overview of the overall abstraction refinement technique for synthesizing optimal controllers, but focuses only on the specific components of the framework that are different from the previous chapter. Then, the application of the technique on a hybrid system is presented with an analysis of the algorithm's convergence to the optimal cost. Lastly, an example mobile robot system is used to illustrate the implementation of the technique.

10.1 Controller Synthesis for ω-regular Objectives

In this section, given a weighted transition system $\mathcal{T}$, an initial state s_0, and an ω-regular property Π, we seek to synthesize an optimal controller σ for $\mathcal{T}$. The first part of this section discusses the overall abstraction refinement procedure and the

Algorithm 10 OptCAR (Abstraction Refinement Procedure)

```
Input: System $\mathcal{T}$, property $\Pi$ as a finite state automaton, initial state $s_0$, rational
  number $\varepsilon_0 > 0$, maximum iterations $i_m$, and optimality gap $\varepsilon$
  Set $\varepsilon := \varepsilon_0$, $i := 0$
  while $i < i_m$ do
      $\hat{\mathcal{T}}, \hat{x}_0 := ConsAbs(\mathcal{T}, s_0, \varepsilon)$
      $J_\varepsilon, \hat{\sigma} := SolveTwoPlayerGame(\hat{\mathcal{T}}, \hat{x}_0, \Pi, \varepsilon)$
      $\sigma := ExtractController(\hat{\sigma}, \hat{\mathcal{T}}, \mathcal{T})$
      Output $\sigma$ and $J_\varepsilon$
      $\varepsilon := \frac{\varepsilon}{2}$
      $i := i + 1$
  end while
```

main components of the framework. This framework summarized in Algorithm 10 is similar to Algorithm 8 in Chapter 9. The next part discusses one of the components of Algorithm 10 that is specific to ω-regular objectives, namely, optimal strategy synthesis procedure given a finite state transition system, an initial state, and an ω-regular property Π.

10.1.1 Abstraction-Refinement Procedure

Algorithm 10 presents the abstraction-refinement procedure that is similar to Algorithm 8 in Chapter 9. It first partitions the state and input spaces of the given system $\mathcal{T}$ into grids with particular cell sizes, ϵ, and constructs an abstract system $\hat{\mathcal{T}}$ using *ConsAbs*. Then, it computes the optimal cost, J, and a strategy for the abstract system by solving a two-player game using *SolveTwoPlayerGame*. A suboptimal strategy for $\mathcal{T}$ is extracted from the strategy of the abstract system $\hat{\mathcal{T}}$ with the cost upper bounded by J using the procedure *ExtractController*. To reduce J, refine the state space partitions using smaller grid sizes, $\epsilon/2$, and repeat the whole process. The details of each of the functions can be found in Chapter 9. Here, instead of solving a two-player game over finite sequences on a finite graph, Algorithm 10 solves a two-player game over infinite sequences on a finite graph because the ω-regular objectives consist of infinite traces. This step is explained in detail in the next subsection.

This algorithm outputs a sequence of suboptimal strategies, whose costs are non-increasing over the iterations. Note that the optimal cost for $\mathcal{T}$ might not be achievable due to the optimality gap ε. As such, the algorithm is chosen to terminate after a fixed number of iterations. However, for a special class of systems that has a "lasso-type" robust optimal path (see Definition 10.2 and Figure 10.1), we show that the suboptimal cost converges to the optimal cost as i goes to infinity. See Section 10.2.3 for more details.

A winning strategy $\hat{\sigma}$ might not exist when the partitioning is coarse, even if the underlying system $\mathcal{T}$ has an optimal solution. However, once a winning strategy is found for a particular partitioning, a controller that has bounded trajectory cost is obtained. Furthermore, a more refined partitioning will return a controller with trajectory cost that is no worse than the trajectory cost resulting from the controller corresponding to the winning strategy for the current partitioning. Thus, the algorithm can be terminated at a specific iteration based on the application and the computational resources.

Algorithm 11 *SolveTwoPlayerGame* (Two-Player Games)

```
Input: Finite state transition system $\mathcal{T}$, property $\Pi$ specified as $\mathcal{B}$, initial state $s_0$,
    and optimality gap $\varepsilon$.
    $\mathcal{A}_{\mathcal{T},\mathcal{B}} := ProductSystem(\mathcal{T}, \mathcal{B})$
    $C, \sigma_{\mathcal{A}} := MeanPayoffParityGame(\mathcal{A}_{\mathcal{T},\mathcal{B}}, \varepsilon)$
    $\sigma := ExtractStrategy(\sigma_{\mathcal{A}})$
    if $C(s_0) < \infty$ then
        Output the strategy $\sigma$ and the cost $C(s_0)$
    end if
```

10.1.2 Synthesizing Optimal Strategy

Here, we explain the procedure to compute the optimal cost and optimal strategy for a finite transition system $\mathcal{T}$ with respect to a ω-regular property Π performed by the subroutine *SolveTwoPlayerGame* in Algorithm 10 that is different from Chapter 9. Algorithm 11 reduces the problem of optimal strategy synthesis for general ω-regular objectives to that of a specific ω-regular objective, which is an instance of a mean payoff parity games. Essentially, it constructs a weighted product transition system from the weighted transition system $\mathcal{T}$ and a Büchi automaton that encodes the ω-regular property Π. This product system has an associated ω-regular property Π_p that is the independent of $\mathcal{T}$ and Π. Solving for the optimal winning strategy of the product system with respect to Π_p is an instance of solving a mean payoff parity game [151]. Once the optimal cost and winning strategy $\sigma_{\mathcal{A}}$ of the product system are computed, the winning strategy for $\mathcal{T}$ is extracted from $\sigma_{\mathcal{A}}$ by projecting $\sigma_{\mathcal{A}}$ onto $\mathcal{T}$. Finally, Algorithm 11 returns both the optimal cost and the winning strategy for $\mathcal{T}$.

More precisely, the function *ProductSystem* forms a weighted product transition system that is a product of a weighted transition system $\mathcal{T}$ and the Büchi automaton $\mathcal{B}$ representing the property Π. From here onwards, fix $\mathcal{T} = (\mathcal{S}, \mathcal{S}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$ as the weighted transition system and $\mathcal{B} = (Q, Q^{init}, \mathcal{P}, \mathcal{E}, \mathcal{F})$ as the Büchi automaton representing the property Π, where $\Pi = \mathcal{L}(\mathcal{B})$.

Definition 10.1. A weighted product transition system of $\mathcal{T}$ and $\mathcal{B}$ is defined as $\mathcal{A}_{\mathcal{T},\mathcal{B}} = (Q_{\mathcal{A}}, Q_{\mathcal{A}}^{init}, \mathcal{U}_{\mathcal{A}}, \mathcal{P}_{\mathcal{A}}, \Delta_{\mathcal{A}}, \mathcal{L}_{\mathcal{A}}, \mathcal{W}_{\mathcal{A}})$, where

- $Q_{\mathcal{A}} = \{(s, q) \in \mathcal{S} \times Q\}$;
- $Q_{\mathcal{A}}^{init} = \{(s, q) \in \mathcal{S}^{init} \times Q^{init}\}$;
- $\mathcal{U}_{\mathcal{A}} = \mathcal{U} \times \mathcal{E}$;
- $\mathcal{P}_{\mathcal{A}} = \{0, 1\}$;

- $\Delta_{\mathcal{A}} = \{((s,q),(u,e),(s',q')) \in Q_{\mathcal{A}} \times \mathcal{U}_{\mathcal{A}} \times Q_{\mathcal{A}} \mid (s,u,s') \in \Delta, e = (q,p,q') \in \mathcal{E}, \mathcal{L}(s) = p\}$;
- $\mathcal{L}_{\mathcal{A}} : Q_{\mathcal{A}} \to \{0,1\}$, where $\mathcal{L}_{\mathcal{A}}(s,q) = 0$ if $q \in \mathcal{F}$ and $\mathcal{L}_{\mathcal{A}}(s,q) = 1$ if otherwise, and
- $\mathcal{W}_{\mathcal{A}} : Q_{\mathcal{A}} \times \mathcal{U}_{\mathcal{A}} \times Q_{\mathcal{A}} \to \mathbb{R}_+$ such that $\mathcal{W}_{\mathcal{A}}((s,q),(u,e),(s',q')) = \mathcal{W}(s,u,s')$.

Fix $\mathcal{A}_{\mathcal{T},\mathcal{B}} = (Q_{\mathcal{A}}, Q_{\mathcal{A}}^{init}, \mathcal{U}_{\mathcal{A}}, \mathcal{P}_{\mathcal{A}}, \Delta_{\mathcal{A}}, \mathcal{L}_{\mathcal{A}}, \mathcal{W}_{\mathcal{A}})$. We consider the following winning objective Π_p for $\mathcal{A}_{\mathcal{T},\mathcal{B}}$: Π_p consists of all sequences over $\{0,1\}$ that contain infinitely many 0s. Note that this objective is the same for any $\mathcal{T}$ and $\mathcal{B}$, and its simple form enables the problem to be posed as a mean payoff parity game.

Before moving on to the mean payoff parity game, we will discuss the relationship between the winning strategy $\sigma_{\mathcal{A}}$ of the product system $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ and the winning strategy σ of the corresponding transition system $\mathcal{T}$. To do so, we need to define the projection of a strategy. Let $\zeta_{\mathcal{A}} = (s_0,q_0)(u_0,e_0)(s_1,q_1)(u_1,e_1)(s_2,q_2)\dots$ be a path conforming to $\sigma_{\mathcal{A}}$ from initial state (s_0,q_0) and let $\zeta_{\mathcal{A}}^i = (s_0,q_0)(u_0,e_0)\dots(s_i,q_i)$ be a prefix of $\zeta_{\mathcal{A}}$ up to index i. Furthermore, let $Proj_{\mathcal{T}}(\zeta_{\mathcal{A}})$ be the projection of a path $\zeta_{\mathcal{A}}$ in $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ onto $\mathcal{T}$, where $Proj_{\mathcal{T}}(\zeta_{\mathcal{A}}) = s_0u_0s_1u_1\dots$ and $Proj_{\mathcal{T}}(\zeta_{\mathcal{A}}^i) = s_0u_0\dots s_i$. Then, the projection of a strategy $\sigma_{\mathcal{A}}$ of $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ onto $\mathcal{T}$ is defined as $\sigma = Proj_{\mathcal{T}}^{\sigma}(\sigma_{\mathcal{A}})$, whereby if $\sigma_{\mathcal{A}}(\zeta_{\mathcal{A}}^i) = (u_i,e_i)$, then $\sigma(Proj_{\mathcal{T}}(\zeta_{\mathcal{A}}^i)) = u_i$ for all $i \geq 0$. This projection forms a “one-to-one” cost preserving correspondence between $\sigma_{\mathcal{A}}$ and σ.

Theorem 10.1. *A winning strategy $\sigma_{\mathcal{A}}$ of $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ with respect to Π_p from the state (s_0,q_0) exists if and only if a winning strategy σ for $\mathcal{T}$ with respect to $\mathcal{L}(\mathcal{B})$ from s_0 exists. Furthermore, $\sigma = Proj_{\mathcal{T}}^{\sigma}(\sigma_{\mathcal{A}})$ is a winning strategy for $\mathcal{T}$, and $\mathcal{W}(\mathcal{A}_{\mathcal{T},\mathcal{B}}, \sigma_{\mathcal{A}}, (s_0,q_0)) = \mathcal{W}(\mathcal{T},\sigma,s_0)$.*

The proof is available in Appendix D.1. Theorem 10.1 implies that if a winning strategy for $\mathcal{T}$ exists, there will be a winning strategy $\sigma_{\mathcal{A}}$ for $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ that the algorithm can search for. Furthermore, given $\sigma_{\mathcal{A}}$, we can construct a winning strategy σ for $\mathcal{T}$ by projection, and the cost of σ is the same as the cost of $\sigma_{\mathcal{A}}$. Therefore, *ExtractStrategy* in Algorithm 11 extracts a winning strategy for $\mathcal{T}$ from $\sigma_{\mathcal{A}}$ by projecting $\sigma_{\mathcal{A}}$ onto $\mathcal{T}$. Note that *ExtractStrategy* is different from *ExtractController* in Algorithm 10 because *ExtractController* extracts a suboptimal controller for the concrete system $\mathcal{T}$ from the optimal strategy of the abstract system $\hat{\mathcal{T}}$.

Now, we return to the mean payoff parity game. A mean payoff parity game is a two-player game on a finite graph, whereby the strategy for player 1 is winning if

$\min(Inf(Tr(\zeta)))$ is even for all paths ζ that conform to the strategy, and optimal if the cost of the strategy is minimized [151]. Solving for an optimal winning strategy for $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ with respect to Π_p is an instance of a mean payoff parity game. Thus, given $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ and Π_p, *MeanPayoffParityGame* solves a mean payoff parity game to obtain the winning strategy $\sigma_{\mathcal{T}}$ that is ε-optimal, and the optimal value C for each states. If a state s is not winning, $C(s) = \infty$. The strategy is ε-optimal because a mean payoff parity game may not have a finite memory optimal strategy in general [151]. However, a finite memory strategy is achievable for an ε-optimal cost through solving an energy parity game [150]. The example that follows uses the algorithm in [150] to solve for an ε-optimal strategy. Due to this optimality gap, we do not expect to obtain the optimal control for the concrete system in general. Instead, this technique returns a suboptimal controller that has a finite memory. Nonetheless, if the optimal controller has finite memory, the optimality gap is zero.

10.1.3 Complexity of Algorithm 10 and 11

The computation cost for Algorithm 10 is mostly due to *ConsAbs*. This step involves $|\mathcal{S}|^2|\mathcal{U}|$ parallelizable optimizations, where $|\mathcal{S}|$ is the number of state in $\hat{\mathcal{T}}$, and $|\mathcal{U}|$ is the number of inputs in $\hat{\mathcal{T}}$. All optimizations are linear programs if the system is linear, and the state and input partitions are represented by polyhedra.

In Algorithm 11, *ProductSystem* runs in $O(|\mathcal{S}||Q||\Delta||\mathcal{E}|)$ time, where $|\mathcal{S}|$ and $|\Delta|$ are the number of states and transitions in $\mathcal{T}$, and $|Q|$ and $|\mathcal{E}|$ are the number of states and transitions in $\mathcal{B}$. The mean payoff parity games solves in $O(|\Delta_{\mathcal{A}}||Q_{\mathcal{A}}|\mathcal{W}^m_{\mathcal{A}}(|Q_{\mathcal{A}}|+1))$ time [150], where $|Q_{\mathcal{A}}|$ and $|\Delta_{\mathcal{A}}|$ are the number of states and transitions in $\mathcal{A}$, and $\mathcal{W}^m_{\mathcal{A}}$ is the maximum value of the cost $\mathcal{W}$ over all transitions.

10.2 Controller Synthesis for Hybrid Systems

The abstraction-refinement approach is implemented for discrete-time piecewise linear systems, and the algorithm converges to the optimal controller if the optimal controller is robust and the optimal path is a “lasso”. We first formally define the optimal control problem for this class of system with ω-regular objectives.

10.2.1 Problem Formulation

A discrete-time piecewise linear system is a tuple $(\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \{(f_i, P_i)\}_{i\in[m]}, \mathcal{L}_d, \mathcal{J})$, where the state-space $\mathcal{X} \subseteq \mathbb{R}^n$ and the input-space $\mathcal{U} \subseteq \mathbb{R}^p$ are compact sets, $\mathcal{X}^{init} \subseteq \mathcal{X}$ is the set of initial states, $\mathcal{P}$ is a finite set of propositions, f_i is a Lipschitz continuous function of $x \in \mathcal{X}$ and $u \in \mathcal{U}$, and P_i is a polyhedral set, such that $\{P_i\}_{i\in m}$ is a polyhedral partition of $\mathcal{X}$, $\mathcal{L}_d : [m] \to \mathcal{P}$ is a labeling function, and $\mathcal{J} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_+$ is a Lipschitz continuous cost function. Note that A_i and B_i do not have to be different for different i. Given an initial state $x_0 \in \mathcal{X}^{init}$ and a sequence of control inputs $\mathbf{u} = u_0u_1\ldots$, where $u_t \in \mathcal{U}$, $\phi(x_0, \mathbf{u}) = x_0x_1\ldots$ is the sequence of states visited under the control $\mathbf{u}$, where $x_{t+1} = f_t(x_t, u_t)$. The cost of the sequence $\phi(x_0, \mathbf{u})$, $\mathcal{J}(\phi(x_0, \mathbf{u}))$, is given by $\lim_{k\to\infty} \frac{1}{k} \sum_{t\in[k-1]} \mathcal{J}(x_{t+1}, u_t)$. Define the partition sequence of $z = x_0x_1\ldots$, denoted $PS(z)$, to be the sequence of partitions visited by the states $P_1P_2\ldots$ such that $x_t \in P_t$ for all t.

Problem 10.1 (Optimal control problem)**.** Given an n-dimensional discrete-time piecewise linear system $\mathcal{D} = (\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \{(f_i, P_i)\}_{i\in[m]}, \mathcal{L}_d, \mathcal{J})$, a state $x_0^* \in \mathcal{X}^{init}$, and a ω-regular property Π over $\mathcal{P}$, find a sequence of control inputs $\mathbf{u}^*$ for which $\mathcal{L}_d(\phi(x_0^*, \mathbf{u}^*)) \in \Pi$ and $\mathcal{J}(\phi(x_0^*, \mathbf{u}^*))$ is minimized.

10.2.2 Equivalent Optimal Strategy Problem

A discrete-time piecewise linear system $\mathcal{D} = (\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \{(f_i, P_i)\}_{i\in[m]}, \mathcal{L}_d, \mathcal{J})$ can be represented as a weighted transition system, $\mathcal{T}_{\mathcal{D}} = (\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$, where $\Delta = \{(x, u, x') \in \mathcal{X} \times \mathcal{U} \times \mathcal{X} \mid x' = f_i(x, u)$, $\mathcal{L}(x) = \mathcal{L}_d(i)$, where $x \in P_i$, and $\mathcal{W}(x, u, x') = \mathcal{J}(x', u)$. Consequently, Problem 10.1 is equivalent to the following problem:

Problem 10.2 (Optimal strategy problem)**.** Given a weighted transition system $\mathcal{T}_{\mathcal{D}} = (\mathcal{X}, \mathcal{X}^{init}, \mathcal{U}, \mathcal{P}, \Delta, \mathcal{L}, \mathcal{W})$, a state $x_0^* \in \mathcal{X}^{init}$, and a ω-regular property Π over $\mathcal{P}$, find an optimal winning strategy $\sigma(\mathcal{T}_{\mathcal{D}}, x_0^*, \Pi)$ for which the optimal cost of winning $\mathcal{T}_{\mathcal{D}}$ with respect to Π, $\mathcal{W}(\mathcal{T}_{\mathcal{D}}, x_0^*, \Pi)$, is achieved.

Solving Problem 10.2 in general is difficult because $\mathcal{T}_{\mathcal{D}}$ is a infinite state system; nonetheless, we can synthesize suboptimal strategies using the approach described in Section 10.1 with formal guarantee on cost sub-optimality. Furthermore, we show that if the transition system $\mathcal{T}_{\mathcal{D}}$ has a robust optimal control and the optimal path is a lasso, the suboptimal cost converges to the optimal cost.

10.2.3 Analysis of Algorithm 10 for Problem 10.2

This section analyzes the output of Algorithm 10 for Problem 10.2 for a special class of systems that has a robust optimal control and a lasso optimal path (see Figure 10.1). The suboptimal cost converges to the optimal cost for this system.

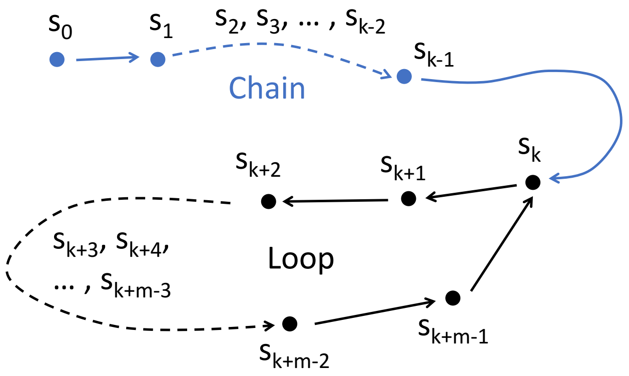


Figure 10.1: A lasso path in which the chain part is in blue, and the loop part is in black.

Definition 10.2. A path is a **lasso** if the path has the form $s_p(s_l)^\omega$, where for some constants $k > 0$ and $m > 0$, $s_p = s_0u_0s_1u_1 \ldots s_{k-1}u_{k-1}$ (referred to as the chain), and $s_l = s_ku_k \ldots s_{k+m-1}u_{k+m-1}$ (referred to as the loop).

Definition 10.3. Let $\mathbf{u}$ be the input sequence that results in a lasso trajectory $\phi(x_0, \mathbf{u}) = x_p(x_l)^\omega$ for an initial state x_0, where $x_p = \{x_i\}_{i\in[k-1]}$, $x_l = \{x_{j+k}\}_{j\in[m-1]}$, and $PS(\phi(x_0, \mathbf{u})) = \{P_{i_t}\}_{t\in\mathbb{N}}$. This input sequence $\mathbf{u}$ is said to be *robust* with respect to the initial state x_0 if

1. there exists $\varepsilon_t > 0$ such that $\mathcal{B}_{\varepsilon_t}(x_t) \subseteq P_{i_t}$ for all $t \in \mathbb{N}$;
2. there exist bounds $M_x > 0$ and $M_u > 0$, and constant $L < 1$ such that for all $\{x'_i\}_{i\in[m]}$ and $\{u'_i\}_{i\in[m-1]}$, with $(x'_i, u'_i, x'_{i+1}) \in \Delta$ for all $i \in [m-1]$, $x'_0 \in \mathcal{B}_{M_x}(x_k)$ and $u'_i \in \mathcal{B}_{M_u}(u_{k+i})$ for all $i \in [m-1]$, we have $\|x'_m - x_k\|_\infty \le L \|x'_0 - x_k\|_\infty$.

Robustness ensures that the sequence of labels of the optimal lasso do not change when the initial state is perturbed slightly, and the perturbed trajectory does not diverge every time it makes a loop. Instead, it becomes strictly closer to the optimal lasso. Note that the second property can be achieved trivially if the system is stable.

Henceforth, let $\mathbf{u}^* = u^*_p(u^*_l)^\omega$ be a robust optimal control input sequence with respect to x^*_0, where $u^*_p = \{u^*_i\}_{i\in[k-1]}$ and $u^*_l = \{u^*_{j+k}\}_{j\in[m-1]}$. In addition, let $\phi(x^*_0, \mathbf{u}^*) = x^*_p(x^*_l)^\omega$ be the corresponding optimal trajectory for Problem 10.1, where $x^*_p = \{x^*_i\}_{i\in[k-1]}$ and $x^*_l = \{x^*_{j+k}\}_{j\in[m-1]}$. Lastly, let $\zeta^* = s^*_p(s^*_l)^\omega$ be the corresponding optimal state and input sequence, where $s^*_p = \{x^*_iu^*_i\}_{i\in[k-1]}$ and $s^*_l = \{x^*_{j+k}u^*_{j+k}\}_{j\in[m-1]}$. The proof requires a special kind of strategy, which has a unique path conforming to it by choosing inputs that result in exactly one successor state.

Definition 10.4. A **chain lasso strategy** for a transition system $\mathcal{T}$ and an initial state s_0 is a strategy $\sigma \in Str(\mathcal{T})$ such that there is one path in $Paths^m_\sigma(\mathcal{T}, s_0)$ and the path is a lasso.

Next, we will show that the strategy given by Algorithm 10 produces a suboptimal cost that converges to the optimal cost of $\mathcal{D}$. Denote the elements in the iteration of Algorithm 10 corresponding to a particular ε as $\hat{\mathcal{T}}_\varepsilon$ for $\hat{\mathcal{T}}$, $\hat{x}^\varepsilon_0$ for $\hat{x}_0$, J_ε for J, $\hat{\sigma}_\varepsilon$ for $\hat{\sigma}$ and σ_ε for $\sigma_\mathcal{D}$.

Theorem 10.2. *If there exists a robust optimal control* $\mathbf{u}^*$ *with respect to* x^*_0 *for Problem 10.1 and the optimal path is a lasso, the sequence of sub-optimal costs* $\{J_{\varepsilon_0/2^i}\}_{i\in\mathbb{N}_+}$ *output by Algorithm 10 converges to the optimal cost* $J_{opt} = \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0, \Pi)$. *Furthermore, for each sub-optimal cost* $J_{\varepsilon_0/2^i}$, *there exists a suboptimal winning strategy* $\sigma_{\varepsilon_0/2^i}$ *with cost no larger than* $J_{\varepsilon_0/2^i}$.

Proof. A sketch of the proof is provided. The full proof is available in Appendix D.2. First, note that given Definition 10.3, if the initial states and the inputs are slightly perturbed from the optimal inputs, the perturbed trajectory has error that is bounded relative to the optimal trajectory. In addition, given the cost function in Problem 10.2, the cost of this suboptimal trajectory is bounded due to continuity of the state transitions. Because the error and the cost are bounded, we can construct an abstraction to give a chain lasso winning strategy σ for a given concrete system $\mathcal{T}_\mathcal{D}$ such that the cost of σ is bounded by any given cost sub-optimality $\delta > 0$ with respect to the optimal cost.

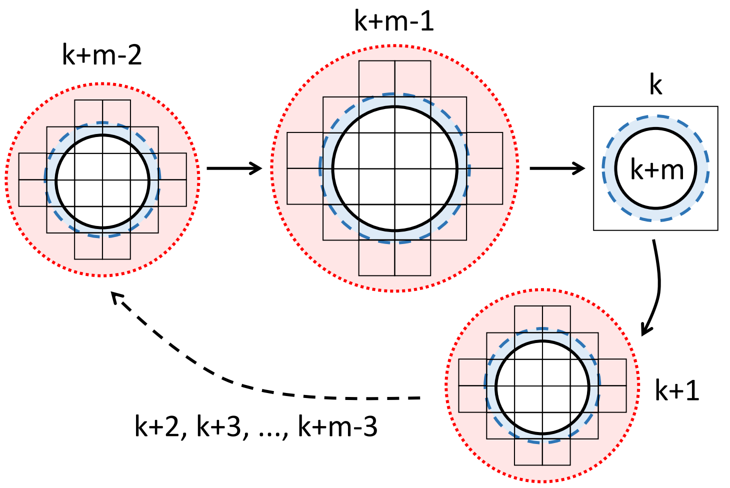


Figure 10.2: Construction of neighborhoods for the loop in the optimal path.

To obtain this chain lasso strategy, we will construct a sequence of disjoint neighborhoods $\{N_t^x\}_{t\in[k+m-1]}$ and $\{N_t^u\}_{t\in[k+m-1]}$ around the optimal states and inputs such that every transition from N_t^x on N_t^u will end in N_{t+1}^x, and each neighborhood is an element or a collection of elements in $\varepsilon_0/2^i$ grid for $i \in \mathbb{N}$. The grid size can be different for different neighborhoods, and the neighborhoods is disjoint by picking a small enough ε_0. The same approach as Theorem 9.1 constructs the chain of the chain lasso strategy in a backward induction from the beginning of the loop. The loop of the chain lasso strategy is constructed backward inductively as illustrated by Figure 10.2. The white square is N_k^x. The white circles are neighborhoods that are reachable from N_k^x on $N_t^u \subseteq \mathcal{B}_{M_u}(u_t^*)$ for $t \geq k$. At $k+m$, define a blue circle such that it is contained within the white square and the white circle is contained within the blue circle. Then, define the red circle at $k+m-1$ such that all states within the red circle will reach a state within the subsequent blue circle on N_{k+m-1}^u. Now, construct the collection of squares in $k+m-1$ to form N_{k+m-1}^x, whereby the squares are elements in $\varepsilon_0/2^i$ grid for $i \in \mathbb{N}$. This sequence of construction can be performed backward from $k+m$ to $k+1$, and it ensures that every transition from N_t^x on N_t^u will end in N_{t+1}^x. Furthermore, $|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq \delta$ for any path ζ starting in an ε_x ball around x_0^*. Thus, we have a chain lasso strategy σ with bounded cost.

Next, refine σ to construct a uniform grid by choosing the size of the uniform grid ε to be the smallest grid of all neighborhoods N_t^u and N_t^x. Then, define a strategy σ_ε (not necessarily a chain lasso anymore) for $\hat{\mathcal{T}}_\varepsilon$, which follows the neighborhoods N_t^x. All paths in $\hat{\mathcal{T}}_\varepsilon$ conforming to σ_ε are contained in the neighborhoods N_t^x. Thus, the cost of σ_ε is bounded by that of σ, and consequently, the optimal cost of $\hat{\mathcal{T}}_\varepsilon$ is at most δ larger than that of $\mathcal{T}_\mathcal{D}$.

Lastly, because the optimal path is a lasso, the optimal strategy has a finite memory, and thus the optimality gap ε is zero. As a result, $J_\varepsilon = \mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi)$, $J_{opt} = \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0^*, \Pi)$ is the optimal cost, and $|J_\varepsilon - J_{opt}| \leq \delta$ for a given $\delta > 0$. In fact, for each J_ε, there exists a suboptimal winning strategy σ_ε with cost no larger than J_ε. In addition, $J_{\varepsilon_0/2^i}$ converges to J_{opt} as $i \longrightarrow \infty$ because $J_{\varepsilon_0/2^i} \leq J_{\varepsilon_0/2^j}$ for all $i > j$. □

The trace of all the paths given by the strategies in the proof of Theorem 10.2 is the same. During implementation, Algorithm 10 may return a sequence of suboptimal strategies $\sigma_{\varepsilon_0/2^i}$ that results in paths with different lengths for the chain part and/or the loop part. Nonetheless, the cost of each path results from $\sigma_{\varepsilon_0/2^i}$ is bounded by the cost $J_{\varepsilon_0/2^i}$.

10.3 Implementation

Consider a surveillance example that is represented in Figure 7.2a. A surveillance robot is required to visit region A and B repeatedly while avoiding obstacle X. In addition, the robot has to move using a minimal amount of energy. Mathematically, the robot dynamics can be represented with the following linear dynamical system:

$$x_{t+1} = x_t + u_t$$
$$y_{t+1} = y_t + v_t,$$

where $[x_t, y_t] \in [-4,4]^2$, and controls $[u_t, v_t] \in [-2,2]^2$. The cost function is $\mathcal{J}(\phi(x_0,u)) = \lim_{k\to\infty} 1/k \sum_{t\in[k-1]} \|u_t\|_1$. The goal is to drive the robot from an initial point $x_0 = (-2.5,-2.5)$ to region $A = \{(x,y) \mid -4 \le x \le -3, 3 \le y \le 4\}$ and $B = \{(x,y) \mid 3 \le x \le 4, -4 \le y \le -3\}$. The robot is required to visit region A and B infinitely often, and always avoid region $X = \{(x,y) \mid -1 \le x \le 1, -1 \le y \le 1\}$. The property of this example is represented in Figure 7.2a (not to scale) and as a Büchi automaton in Figure 7.2b. The algorithm is implemented on a 8×8 uniform grid on the states. The input, u, is partitioned into 4×4 uniform intervals.

Algorithm 10 and 11 are implemented in Python 2.7. A Python package, NetworkX, is used to represent the graph structures that arise in solving Algorithm 11, and the Parma Polyhedra Library [158] is used to represent the polyhedral sets that arise in the gridding and to solve the linear program problem that arises in the weight computation. The algorithms are implemented on a MacBook Pro with 2.9GHz Intel Core i5 processor, and 16GB RAM.

Figure 10.3 shows the state trajectory of the robot under the strategy given by Algorithm 10 and 11. The strategy is able to direct the system to move from the initial position at $(-2.5,-2.5)$ to region A and then region B. The cost at the end of time $N = 31$ is 1.33. Since the strategy forms a cycle, the cost when $N \to \infty$ will not be larger than 1.33, which is lower than the cost of the strategy at 1.5 given by the algorithms.

10.4 Summary

This chapter presents a technique to synthesize optimal controllers for discrete-time hybrid systems with ω-regular objectives based on an abstraction-refinement procedure, extending the results in Chapter 9. This method provides a guarantee on the upper bound of the trajectory cost when implementing the suboptimal controller. Furthermore, for systems with a robust optimal controller and a lasso optimal path, the abstraction-refinement procedure converges to the optimal solution.

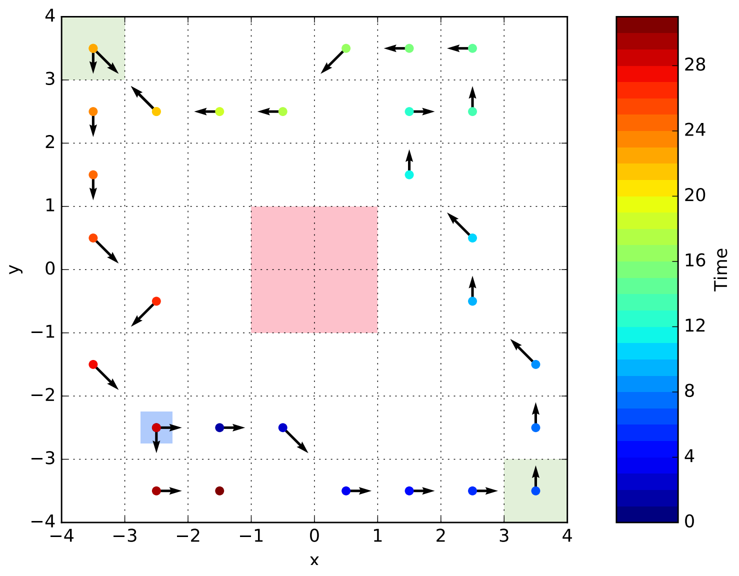


Figure 10.3: Simulated results on the linear dynamical system. The blue box indicates the starting position. The arrows represent the control input directions, and the round markers indicates the position from time 0 to time 31 (blue to red).

Future works include extending the results to more complex systems such as continuous-time systems, and implementing the technique on more sophisticated examples. To reduce computation time, a more intelligent gridding scheme in the refinement or abstraction step [161], [162] will be developed.

Appendix B

PROOFS FOR CHAPTER 8

B.1 Proof of Theorem 8.1

Define (α, β) to be identity relations on the state and input spaces, then $\mathcal{T} \preceq_{(\alpha,\beta)} \mathcal{T}$, and hence $\preceq$ is reflexive. To show that $\preceq$ is transitive, suppose $\mathcal{T}_1 \preceq_{(\alpha_1,\beta_1)} \mathcal{T}_2$ and $\mathcal{T}_2 \preceq_{(\alpha_2,\beta_2)} \mathcal{T}_3$. Define α such that $(s_1, s_3) \in \alpha$ if $(s_1, s_2) \in \alpha_1$ and $(s_2, s_3) \in \alpha_2$ for some s_2, and define β such that $(s_1, u_1, s_3, u_3) \in \beta$ if $(s_1, u_1, s_2, u_2) \in \beta_1$ and $(s_2, u_2, s_3, u_3) \in \beta_2$ for some (s_2, u_2). Then, $\mathcal{T}_1 \preceq_{(\alpha,\beta)} \mathcal{T}_3$.

B.2 Proof of Theorem 8.2

Let σ_2 be a strategy for $\mathcal{T}_2$ and s'_0. In addition, define a partial mapping $G : \mathit{Paths}_f(\mathcal{T}_1) \rightarrow \mathit{Paths}_f(\mathcal{T}_2)$ such that the domain of G is the set of all paths from s_0 that conform to σ_1, and for any path ζ_1 in the domain of G, $\mathcal{L}_1(\zeta_1) = \mathcal{L}_2(G(\zeta_1))$, and $\mathcal{W}_1(\zeta_1) \leq \mathcal{W}_2(G(\zeta_1))$. This construction ensures that if σ_2 is winning from s'_0 with respect to Π, then so is σ_1 from s_0 and $\mathcal{W}_1(\mathcal{T}_1, \sigma_1, s_0) \leq \mathcal{W}_2(\mathcal{T}_2, \sigma_2, s'_0)$. We also ensure that if $G(\zeta_1) = \zeta_2$, then $(s_k, s'_k) \in \alpha$, where s_k and s'_k are the end states of ζ_1 and ζ_2, respectively. Further, for any ζ_1 in the domain of G, ζ_1 is a maximal path conforming to σ_1 if and only if ζ_2 is a maximal path conforming to σ_2.

Next, define σ_1 and G by induction on the length of words in their domain. Set $G(s_0) = s'_0$. Suppose σ_1 for paths of length $k-1$ and G for paths of length k, are defined such that the invariant holds. Let $\zeta_1 = s_0 u_0 s_1 \ldots s_k$ conform to σ_1. Then, $G(\zeta_1)$ is defined. Let $G(\zeta_1) = s'_0 u'_0 s'_1 \ldots s'_k$ and $(s_k, s'_k) \in \alpha$. If $G(\zeta_1)$ is a maximal path conforming to σ_2, then $\sigma_1(\zeta_1)$ is not defined (*i.e.,* ζ_1 is not in the domain of σ_1). Otherwise $\sigma_2(G(\zeta_1)) = u'_k$. Then, from the second condition of simulation, there exists u_k such that $(s_k, u_k, s'_k, u'_k) \in \beta$. Choose $\sigma_1(\zeta_1) = u_k$. For any $\zeta_2 = s_0 u_0 s_1 \ldots s_{k+1}$, define $G(\zeta_2) = s'_0 u'_0 s'_1 \ldots s'_{s+1}$ such that $(s_{k+1}, s'_{k+1}) \in \alpha$ and $\mathcal{W}_1(s_k, u_k, s_{k+1}) \leq \mathcal{W}_2(s'_k, u'_k, s'_{k+1})$. It can be verified that the construction satisfies the inductive invariant.

B.3 Proof of Theorem 8.3

Let σ_2 be a strategy for $\mathcal{T}_2$ and s'_0. In addition, define a partial mapping $G : \mathit{Paths}_f(\mathcal{T}_1) \rightarrow \mathit{Paths}_f(\mathcal{T}_2)$ such that the domain of G is the set of all paths from s_0 that conform to σ_1, and for any path ζ_1 in the domain of G, $\mathcal{L}_1(\zeta_1) = \mathcal{L}_2(G(\zeta_1))$, and $\mathcal{W}_1(\zeta_1) \leq \mathcal{W}_2(G(\zeta_1))$. This construction ensures that if σ_2 is winning from s'_0 with respect to Π, then so is σ_1 from s_0 and $\mathcal{W}_1(\mathcal{T}_1, \sigma_1, s_0) \leq \mathcal{W}_2(\mathcal{T}_2, \sigma_2, s'_0)$. We also ensure that if $G(\zeta_1) = \zeta_2$, then $(s_k, s'_k) \in \alpha$, where s_k and s'_k are the states of ζ_1 and ζ_2, respectively.

Next, define σ_1 and G by induction. Set $G(s_0) = s'_0$. Suppose σ_1 for paths of length $k-1$ and G for paths of length k, are defined such that the invariant holds. Let $\zeta_1 = s_0 u_0 s_1 \dots s_k$ conform to σ_1. Then, $G(\zeta_1)$ is defined. Let $G(\zeta_1) = s'_0 u'_0 s'_1 \dots s'_k$ and $(s_k, s'_k) \in \alpha$. Then, $\sigma_2(G(\zeta_1)) = u'_k$. From the second condition of simulation, there exists u_k such that $(s_k, u_k, s'_k, u'_k) \in \beta$. Choose $\sigma_1(\zeta_1) = u_k$. For any $\zeta_2 = s_0 u_0 s_1 \dots s_{k+1}$, define $G(\zeta_2) = s'_0 u'_0 s'_1 \dots s'_{s+1}$ such that $(s_{k+1}, s'_{k+1}) \in \alpha$ and $\mathcal{W}_1(s_k, u_k, s_{k+1}) \leq \mathcal{W}_2(s'_k, u'_k, s'_{k+1})$. This construction satisfies the inductive invariant.

B.4 Proof of Proposition 8.1

Consider $Abs(\mathcal{T}, \equiv_S, \equiv_U) = (\mathcal{S}', \mathcal{S}^{init'}, \mathcal{U}', \mathcal{P}, \Delta', \mathcal{L}', \mathcal{W}')$ as in Definition 8.2. Define $(s, [s]_{\equiv_S}) \in \alpha$ for $s \in \mathcal{S}$, and $(s, u, [s]_{\equiv_S}, [u]_{\equiv_U}) \in \beta$ for $s \in \mathcal{S}$ and $u \in \mathcal{U}$.

The first property in Definition 8.1 is satisfied by construction because for all $S \in \mathcal{S}'$, $\mathcal{L}'(S) = \mathcal{L}(s)$ for any $s \in S$. The second property also holds by construction of $\mathcal{S}^{init'}$ where $\forall s \in \mathcal{S}^{init}$, there exists a $[s]_{\equiv_S} \in \mathcal{S}^{init'}$ such that $(s, [s]_{\equiv_S}) \in \alpha$.

To verify the third property, consider any $(s_1, S_1) \in \alpha$ and $U \in Enabled(S_1)$. Because $U \in \mathcal{U}'$, there exists a $u \in \mathcal{U}$, where $[u]_{\equiv_U} = U$. Given that $Enabled(s_1) = \mathcal{U}$ for a complete transition system, $u \in Enabled(s_1)$. By definition of β, $(s_1, u, S_1, U) \in \beta$. Furthermore, by construction, for $(s_1, u, s_2) \in \Delta$, there exists $S_2 \in \mathcal{S}'$ such that $(s_2, S_2) \in \equiv_S$, and $(S_1, U, S_2) \in \Delta'$ if $(s_1, u, s_2) \in \Delta$. Thus, there exists a $(S_1, U, S_2) \in \Delta'$, where $(s_2, S_2) \in \alpha$. Lastly, $\mathcal{W}'(S_1, U, S_2) \geq \mathcal{W}(s_1, u, s_2)$ because $\mathcal{W}'$ is the maximum over all $s_1 \in S_1, u \in U, s_2 \in S_2$ of $\mathcal{W}(s_1, u, s_2)$.

B.5 Proof of Proposition 8.2

First, $\mathcal{T} \preceq Abs(\mathcal{T}, \equiv'_S, \equiv'_U)$ follows from Proposition 8.1. Define $\alpha = \{([s]_{\equiv_{S'}}, [s]_{\equiv_S}) \mid s \in \mathcal{S}\}$ and $\beta = \{([s]_{\equiv_{S'}}, [u]_{\equiv'_U}, [s]_{\equiv_S}, [u]_{\equiv_U}) \mid s \in \mathcal{S} \text{ and } u \in \mathcal{U}\}$. Then, properties in Definition 8.1 are satisfied for $Abs(\mathcal{T}, \equiv'_S, \equiv'_U) \preceq_{(\alpha,\beta)} Abs(\mathcal{T}, \equiv_S, \equiv_U)$, and thus $\mathcal{T} \preceq Abs(\mathcal{T}, \equiv'_S, \equiv'_U) \preceq Abs(\mathcal{T}, \equiv_S, \equiv_U)$.

Appendix C

PROOFS FOR CHAPTER 9

C.1 Proof of Theorem 9.1

For this section, we assume the system is given as in Problem 9.1 in all lemmas.

To prove Theorem 9.1, first, we show that for any trajectory whose initial state and inputs have a bounded deviation from that of the optimal trajectory, the trajectory itself will have a bounded deviation from the optimal trajectory.

Lemma C.1. *There exist bounds $M_x > 0$ and $M_u > 0$ and constants $c_1, c_2 \geq 0$ that depend on $PS(\phi(x_0^*, \mathbf{u}^*))$, such that for all $\varepsilon_x \in [0, M_x]$ and $\varepsilon_u \in [0, M_u]$, if $x_0 \in \mathcal{B}_{\varepsilon_x}(x_0^*)$ and $u_t \in \mathcal{B}_{\varepsilon_u}(u_t^*)$ $\forall t \in [k]$, where $k < \infty$, $\mathbf{u} = \{u_t\}_{t\in[k]}$ and $\phi(x_0, \mathbf{u}) = \{x_t\}_{t\in[k+1]}$, then for all $t \in [k]$,*

$$\left\|x_{t+1} - x_{t+1}^*\right\|_\infty \leq c_1(t)\varepsilon_x + c_2(t)\varepsilon_u,$$
$$PS(\phi(x_0, \mathbf{u})) = PS(\phi(x_0^*, \mathbf{u}^*)).$$

Proof. First, we claim that

$$\left\|x_{t+1} - x_{t+1}^*\right\|_\infty \leq \left(\prod_{i=0}^{t} C_i\right)\varepsilon_x + \left(\sum_{k=0}^{t}\prod_{i=k}^{t} C_i\right)\varepsilon_u.$$

The claim is proved by induction over t. Let $\bar{x}_t = [x_t, u_t] \in \mathbb{R}^{n+p}$. Note that because f_t is Lipschitz continuous for all t, for each f_t and i, there exists a constant $C_t > 0$ such that $|f_t(x_i, u_i) - f_t(x_i^*, u_i^*)| \leq C_t \left\|\bar{x}_i - \bar{x}_i^*\right\|_\infty$. Consider the base case of the induction when $t = 0$

$$\begin{aligned}
\left\|x_1 - x_1^*\right\|_\infty &= \left\|f_0(x_0, u_0) - f_0(x_0^*, u_0^*)\right\|_\infty \\
&\leq C_0\left\|\bar{x}_0 - \bar{x}_0^*\right\|_\infty \\
&\leq C_0(\left\|x_0 - x_0^*\right\|_\infty + \left\|u_0 - u_0^*\right\|_\infty) \\
&= C_0(\varepsilon_x + \varepsilon_u).
\end{aligned}$$

where the last equality is given by the problem statement. The base case satisfies the claim.

Now, assume that the claim is true up for all $t \leq \tau$. Then, consider

$$\begin{aligned}
\left\|x_{\tau+1} - x^*_{\tau+1}\right\|_\infty &= \left\|f_\tau(x_\tau, u_\tau) - f_\tau(x^*_\tau, u^*_\tau)\right\|_\infty \\
&\leq C_\tau \left\|\bar{x}_\tau - \bar{x}^*_\tau\right\|_\infty \\
&\leq C_\tau(\left\|x_\tau - x^*_\tau\right\|_\infty + \left\|u_\tau - u^*_\tau\right\|_\infty) \\
&\leq C_\tau \left(\prod_{i=0}^{\tau-1} C_i\right) \varepsilon_x + C_\tau \left(\sum_{k=0}^{\tau-1} \prod_{i=k}^{\tau-1} C_i\right) \varepsilon_u + C_\tau \varepsilon_u \\
&= \left(\prod_{i=0}^{\tau} C_i\right) \varepsilon_x + \left(\sum_{k=0}^{\tau-1} \prod_{i=k}^{\tau} C_i\right) \varepsilon_u + C_\tau \varepsilon_u \\
&= \left(\prod_{i=0}^{\tau} C_i\right) \varepsilon_x + \left(\sum_{k=0}^{\tau} \prod_{i=k}^{\tau} C_i\right) \varepsilon_u.
\end{aligned}$$

The result satisfies the claim. Thus, the claim is true by induction. Thus, $c_1(t) = \prod_{i=0}^{t} C_i$ and $c_2(t) = \sum_{k=0}^{t} \prod_{i=k}^{t} C_i$.

Now, note that $\mathbf{u}^*$ is robust with respect to the polyhedral partitioning of $\mathcal{X}$. Thus, for all $t \in [k]$, there exists a constant $\varepsilon_t > 0$ such that $\mathcal{B}_{\varepsilon_t}(x_t) \subseteq P_{i_t}$. Set M_x and M_u such that

$$(M_x, M_u) = \min_{t \in [k]} \max\{(\varepsilon_x, \varepsilon_u) \mid c_1(t)\varepsilon_x + c_2(t)\varepsilon_u \leq \varepsilon_t\}.$$

Then, $PS(\phi(x_0, \mathbf{u})) = PS(\phi(x^*_0, \mathbf{u}^*))$. □

This lemma implies that the error from the optimal state at any time is bounded linearly by the error from the initial state and the error of control inputs from the optimal ones. As ε_x and ε_u decrease to zero, the state error decreases to zero. Although the constants c_1 and c_2 depend on t, t would not make the constants unbounded because t is finite. Next, we show that the suboptimal cost of this trajectory is bounded.

Lemma C.2. *Given the cost function in Problem 9.2, there exist bounds $M_x > 0$ and $M_u > 0$ and constants $c_3, c_4 \geq 0$ that depend on $PS(\phi(x^*_0, \mathbf{u}^*))$ such that for all $\varepsilon_x \in [0, M_x]$ and $\varepsilon_u \in [0, M_u]$, if $x_0 \in \mathcal{B}_{\varepsilon_x}(x^*_0)$ and $u_t \in \mathcal{B}_{\varepsilon_u}(u^*_t)$ $\forall t \in [k]$, where $k < \infty$, $\mathbf{u} = \{u_t\}_{t \in [k]}$ and $\zeta = \phi(x_0, \mathbf{u})$, then*

$$\begin{gathered}
|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq c_3 \varepsilon_x + c_4 \varepsilon_u, \\
PS(\phi(x_0, \mathbf{u})) = PS(\phi(x^*_0, \mathbf{u}^*)).
\end{gathered}$$

Proof. First, compute

$$\begin{aligned}|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| &= \left|\sum_{t=0}^{k} \mathcal{J}(x_{t+1}, u_t) - \mathcal{J}(x^*_{t+1}, u^*_t)\right| \\ &\leq \sum_{t=0}^{k} |\mathcal{J}(x_{t+1}, u_t) - \mathcal{J}(x^*_{t+1}, u^*_t)|.\end{aligned}$$

Since $\mathcal{J}(x, u)$ is a Lipschitz continuous function, there exists a constant $C_t > 0$ such that

$$|\mathcal{J}(x_{t+1}, u_t) - \mathcal{J}(x^*_{t+1}, u^*_t)| \leq C_t \left\|\bar{x}_t - \bar{x}^*_t\right\|_\infty .$$

Therefore,

$$|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq \sum_{t=0}^{k} C \left\|\bar{x}_t - \bar{x}^*_t\right\|_\infty ,$$

where $C = \max_{t\in[k]} C_t$ and $\bar{x}_t = [x_{t+1}, u_t] \in \mathbb{R}^{n+p}$ is a joined vector of x and u. By Lemma C.1,

$$\begin{aligned}|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| &\leq \sum_{t=0}^{k} C \max\{c_1(t+1)\varepsilon_x + c_2(t+1)\varepsilon_u, \varepsilon_u\} \\ &\leq \max\{c'_3\varepsilon_x + c'_4\varepsilon_u, kC\varepsilon_u\},\end{aligned}$$

where $c'_3 = \max_{t\in[k]} kCc_1(t+1)$, and $c'_4 = \max_{t\in[k]} kCc_2(t+1)$. Then,

$$\begin{aligned}&|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq c_3\varepsilon_x + c_4\varepsilon_u \\ &c_3 = \begin{cases} c'_3, & c'_3\varepsilon_x + c'_4\varepsilon_u \geq kC\varepsilon_u \\ 0, & c'_3\varepsilon_x + c'_4\varepsilon_u < kC\varepsilon_u \end{cases} \\ &c_4 = \begin{cases} c'_4, & c'_3\varepsilon_x + c'_4\varepsilon_u \geq kC\varepsilon_u \\ kC, & c'_3\varepsilon_x + c'_4\varepsilon_u < kC\varepsilon_u. \end{cases}\end{aligned}$$

The constants M_x and M_u are set to be the same as the ones in Lemma C.1. Then, $PS(\phi(x_0, \mathbf{u})) = PS(\phi(x^*_0, \mathbf{u}^*))$. □

This lemma states that given a continuous cost function, there will be a small neighborhood of the optimal trajectory in which the trajectories will go through the same partition sequence and difference in the cost is bounded and decreases to zero if ε_x and ε_u decrease to zero.

At this point, we have shown that the suboptimal cost is bounded by terms that depends on the input error and initial state error. Next, we show that given a specific cost sub-optimality, there exists a strategy that satisfies this cost error. In other words, we can construct an abstraction to give a chain strategy that satisfies a certain cost error bound.

Lemma C.3. *Given any $\delta > 0$, there exists a chain winning strategy σ for some $\hat{\mathcal{T}} = Abs(\mathcal{T}_{\mathcal{D}}, \equiv_X, \equiv_U)$ such that*

$$|\mathcal{W}(\hat{\mathcal{T}}, \sigma, \hat{x}_0) - \mathcal{W}(\mathcal{T}_{\mathcal{D}}, x_0^*, \Pi)| \leq \delta,$$

where $\hat{x}_0 = [x_0^*]_{\equiv_X}$.

Proof. The broad idea will be to identify neighborhoods N_t^x around x_t^* and N_t^u around u_t^* such that N_t^x is contained in the region of the partition containing x_t^* and all transitions from N_t^x on N_t^u lead to N_{t+1}^x. Further, we will ensure that the maximum cost of any transition from N_t^x to N_{t+1}^x using an input from N_t^u is bounded. Then, by choosing N_t^x and N_t^u to be regions of $\equiv_X$ and $\equiv_Y$, we obtain a chain strategy in $\hat{\mathcal{T}} = |(|\,\mathcal{T}_{\mathcal{D}}, \equiv_X, \equiv_U)$, where the only region of $\hat{\mathcal{T}}$ reachable from the abstract state N_t^x on input N_t^u is N_{t+1}^x. Refer to Figure 9.1 for an illustration of the chain strategy.

Let $PS(\{x_t^*\}_t) = \{P_{i_t}\}_t$. We construct the sequence inductively, starting from $t = k+1$ and moving backwards. Let N_{k+1}^x be a grid cell of size $\varepsilon_0/2^i$ that contains an open ball around x_{t+1}^*, which is contained in P_{i_t}. We can find such N_{k+1}^x because of the robustness of the optimal control as defined in Definition 9.2. Assume we have computed $N_{t+1}^x, N_{t+1}^u, \ldots N_{k+1}^x$. We show how to compute N_t^x and N_t^u. Let $N_{t+1}^x = \mathcal{B}_{\varepsilon'_{t+1}}(x_{t+1}^*)$. Because the dynamic f_{i_t} is Lipschitz continuous, we have $\left\|x_{t+1}^* - x_{t+1}\right\| \leq C_1 \left\|x_t^* - x_t\right\| + C_2 \left\|u_t^* - u_t\right\|$. Set $N_t^u \subseteq \mathcal{B}_{\varepsilon_u}(x_t^*)$, where $\varepsilon_u \in [0, M_u]$. Choose $N_t^x \subseteq \mathcal{B}_{\varepsilon'_t}(x_t^*)$ such that $\varepsilon'_t \in [0, M_x]$, $C_1\varepsilon'_t + C_2\varepsilon_u \leq \varepsilon'_{t+1}$ and $\mathcal{B}_{\varepsilon'_t}(x_t^*) \subseteq P_{i_t}$. Then, every trajectory from N_t^x on N_t^u will end in N_{t+1}^x and the labels for each states within N_t^x is the same. By induction, under this construction, all executions from N_0^x will be in N_t^x after t steps. This chain of neighborhoods gives us a chain strategy.

Further, by Lemma C.2, we can choose ε_x and ε_u such that $c_3\varepsilon_x + c_4\varepsilon_u \leq \delta$ for some constants c_3 and c_4 in order to ensure that the cost of the strategy is within δ of the optimal cost. Thus, $|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq \delta$ for any path ζ starting in an ε_x ball around x_0^*. In addition, choose the N_j^x and N_j^u such that they correspond to an element of an $\varepsilon_0/2^i$ grid for some i (not necessarily the same i for all neighborhoods). Finally, define $\equiv_X$ and $\equiv_U$ such that the N_j^x and N_j^u are all equivalence classes of $\mathcal{X}$ and $\mathcal{U}$, respectively. Note that we need to ensure that for any i, j, N_j^x is the same as N_i^x or the two are disjoint, and a similar condition for N_j^u holds. This condition can be easily ensured during the construction by picking small enough ε_x and ε_u. □

Lemma C.3 guarantees a chain strategy. However, the partitions corresponding to the neighborhoods of N^x and N^u may not correspond to an uniform grid for any ε. Enumeration in Algorithm 8 only contains uniform grids with grid size $\varepsilon_0/2^i$. Thus, the next lemma constructs a uniform grid by refining the chain strategy obtained from Lemma C.3.

Lemma C.4. *For a given $\delta > 0$, there exists a constant $\varepsilon = \varepsilon_0/2^i > 0$, such that $|\mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi) - \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0^*, \Pi)| \leq \delta$. Furthermore, there exists a winning strategy σ_ε with cost no larger than $\mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi)$.*

Proof. From the proof of Lemma C.3, we obtain a sequence of neighborhoods N_t^x and N_t^u, which correspond to a chain strategy, say σ starting from N_0^x. Further, as observed in the proof, we can assume that every N_t^x corresponds to an element of $Grid(\mathcal{X}, \varepsilon_0/2^{i_t})$ for some i_t, and similarly, N_t^u corresponds to an element of $Grid(\mathcal{U}, \varepsilon_0/2^{j_t})$ for some j_t. Let i be the maximum of the i_ts and j_ts. Note that $Grid(\mathcal{X}, \varepsilon_0/2^i)$ refines N_t^x and similarly, $Grid(\mathcal{U}, \varepsilon_0/2^i)$ refines N_t^u. In Figure 9.1, the squares around x_t^* with bold borders are N_t^x, and the dashed squares, which are contained in them correspond to the refined partition. One can define a strategy σ_ε (not necessarily a chain anymore) for $\hat{\mathcal{T}}_\varepsilon$, which correspond to following the neighborhoods N_t^x. Thus, all the paths in $\hat{\mathcal{T}}_\varepsilon$, which conform to σ_ε will be contained in the neighborhoods N_t^x. Therefore, the cost of σ_ε is bounded by that of σ, which is at most δ away from the optimal cost, and the optimal cost of $\hat{\mathcal{T}}_\varepsilon$ is at most δ away from that of $\mathcal{T}_\mathcal{D}$. □

Proof of Theorem 9.1. First, observe that $J_{\varepsilon_0/2^i} \leq J_{\varepsilon_0/2^j}$ for all $i > j$. Further, from Lemma C.4, for any $\delta > 0$, there exists $\varepsilon = \varepsilon_0/2^i$, such that $|\mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi) - \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0^*, \Pi)| \leq \delta$. Note $J_\varepsilon = \mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi)$ and $J_{opt} = \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0^*, \Pi)$ is the optimal cost. Thus, $|J_\varepsilon - J_{opt}| \leq \delta$. Therefore, $J_{\varepsilon_0/2^i}$ converges to J_{opt} as i goes to infinity. In addition, from Lemma C.4, for each sub-optimal cost $J_{\varepsilon_0/2^i}$, there exists a suboptimal winning strategy $\sigma_{\varepsilon_0/2^i}$ with cost no larger than $J_{\varepsilon_0/2^i}$.

Appendix D

PROOFS FOR CHAPTER 10

D.1 Proof of Theorem 10.1

This section proves Theorem 10.1. First, we show that the projection of the winning strategy $\sigma_{\mathcal{A}}$ of the product system $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ onto $\mathcal{T}$ is a winning strategy for $\mathcal{T}$.

Lemma D.1. *Let $\sigma_{\mathcal{A}}$ be a winning strategy of $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ for the state (s_0, q_0) with respect to $\Pi_{\mathcal{T},\mathcal{B}}$. Then, $\sigma = Proj^{\sigma}_{\mathcal{T}}(\sigma_{\mathcal{A}})$ is a winning strategy of $\mathcal{T}$ for the state s_0 with respect to $\mathcal{L}(\mathcal{B})$.*

Proof. By definition of $Proj^{\sigma}_{\mathcal{T}}(\sigma_{\mathcal{A}})$, for any $\zeta \in Paths_{\sigma}(\mathcal{T}, s_0)$, there exists a $\zeta_{\mathcal{A}}$ such that $\zeta = Proj_{\mathcal{T}}(\zeta_{\mathcal{A}})$ and $\zeta_{\mathcal{A}} \in Paths_{\sigma_{\mathcal{A}}}(\mathcal{A}_{\mathcal{T},\mathcal{B}}, (s_0, q_0))$. Let $\zeta \in Paths_{\sigma}(\mathcal{T}, s_0)$, and consider the $\zeta_{\mathcal{A}}$ that satisfies $\zeta = Proj_{\mathcal{T}}(\zeta_{\mathcal{A}})$. Build $r = q_0q_1\ldots$ from $\zeta_{\mathcal{A}}$. By definition of $\mathcal{A}_{\mathcal{T},\mathcal{B}}$, $(q_i, \mathcal{L}(s_i), q_{i+1}) \in E$ and $Tr(\zeta) = \mathcal{L}(s_0)\mathcal{L}(s_1)\ldots$. Thus, r is a run of $\mathcal{B}$ over the sequence $Tr(\zeta)$. Because $\sigma_{\mathcal{A}}$ is winning, $Inf(Tr(\zeta_{\mathcal{A}})) \cap \{0\} \neq \emptyset$, and there are infinitely many (s_i, q_i) in $\zeta_{\mathcal{A}}$ such that $q_i \in F$. Consequently, there are infinitely many q_i in r such that $q_i \in F$, and $Inf(r) \cap F \neq \emptyset$. Thus, $Tr(\zeta) \in \mathcal{L}(\mathcal{B})$, and σ is a winning strategy for $\mathcal{T}$ with respect to $\mathcal{B}$. □

Proof of Theorem 10.1. First, Lemma D.1 implies that if a winning strategy of $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ with respect to $\Pi_{\mathcal{T},\mathcal{B}}$ exists, a winning strategy for $\mathcal{T}$ with respect to $\mathcal{L}(\mathcal{B})$ exists.

To prove the converse, we first construct a strategy $\sigma_{\mathcal{A}}$ for $\mathcal{A}_{\mathcal{T},\mathcal{B}}$. Let σ be the winning strategy for $\mathcal{T}$ with respect to $\mathcal{B}$. Consider any path $\zeta \in Paths_{\sigma}(\mathcal{T}, s_0)$ and let $\zeta = s_0u_0s_1u_1\ldots$. Because σ is winning, there exists a run $r = q_0q_1\ldots$ of $\mathcal{B}$ over $Tr(\zeta)$ such that $(q_i, \mathcal{L}(s_i), q_{i+1}) \in E$ and $Inf(Tr(r)) \cap F \neq \emptyset$. Construct a path $\zeta_{\mathcal{A}}$ in $\mathcal{A}_{\mathcal{T},\mathcal{B}}$ from r and ζ such that $\zeta_{\mathcal{A}} = (s_0, q_0)(u_0, e_0)(s_1, q_1)\ldots$, where $e_i = (q_i, \mathcal{L}(s_i), q_{i+1})$. Let $\zeta^i_{\mathcal{A}} = (s_0, q_0)(u_0, e_0)\ldots(s_i, q_i)$ be a prefix of $\zeta_{\mathcal{A}}$, and $\zeta^i = s_0u_0\ldots s_iu_i$ be a prefix of ζ. Set $\sigma_{\mathcal{A}}(\zeta^i_{\mathcal{A}}) = (\sigma(\zeta^i), e_i)$.

Next, we show that $\sigma_{\mathcal{A}}$ is winning. For any $\zeta_{\mathcal{A}} \in Paths_{\sigma_{\mathcal{A}}}(\mathcal{A}_{\mathcal{T},\mathcal{B}}, (s_0, q_0))$, there exists a ζ such that $\zeta = Proj_{\mathcal{T}}(\zeta_{\mathcal{A}})$ and $\zeta \in Paths_{\sigma}(\mathcal{T}, s_0)$ by the construction of $\sigma_{\mathcal{A}}$. Therefore, $Inf(Tr(r)) \cap F \neq \emptyset$, where $r = q_0q_1\ldots$, and there exists some $q \in F$ such that (s, q), where $s \in \mathcal{S}$ occurs infinitely often in $\zeta_{\mathcal{A}}$. So, $Inf(Tr(\zeta_{\mathcal{A}})) \cap \{0\} \neq \emptyset$, and thus $\sigma_{\mathcal{A}}$ is a winning strategy of $\mathcal{A}_{\mathcal{T},\mathcal{B}}$.

Lastly, we show that the cost is preserved between $\sigma_{\mathcal{A}}$ and σ that is $\mathcal{W}(\mathcal{A}_{\mathcal{T},\mathcal{B}}, \sigma_{\mathcal{A}}, (s_0, q_0)) = \mathcal{W}(\mathcal{T}, \sigma, s_0)$, where $\sigma = Proj^{\sigma}_{\mathcal{T}}(\sigma_{\mathcal{A}})$. For all $\zeta_{\mathcal{A}} \in Paths_{\sigma_{\mathcal{A}}}(\mathcal{A}_{\mathcal{T},\mathcal{B}}, (s_0, q_0))$, there exists a ζ such that $\zeta = Proj_{\mathcal{T}}(\zeta_{\mathcal{A}})$ and $\zeta \in Paths_{\sigma}(\mathcal{T}, s_0)$. Note that the transition cost is preserved such that $\mathcal{W}_{\mathcal{A}}((s_i, q_i), (u_i, e_i), (s_{i+1}, q_{i+1})) = \mathcal{W}(s_i, u_i, s_{i+1})$. Thus, $\mathcal{W}_{\mathcal{A}}(\zeta_{\mathcal{A}}) = \mathcal{W}(\zeta)$, and $\mathcal{W}(\mathcal{A}_{\mathcal{T},\mathcal{B}}, \sigma_{\mathcal{A}}, (s_0, q_0)) \leq \mathcal{W}(\mathcal{T}, \sigma, s_0)$. Similarly, for any $\zeta \in Paths_{\sigma}(\mathcal{T}, s_0)$, there exists a $\zeta_{\mathcal{A}}$ such that $\zeta = Proj_{\mathcal{T}}(\zeta_{\mathcal{A}})$ and $\zeta \in Paths_{\sigma_{\mathcal{A}}}(\mathcal{A}_{\mathcal{T},\mathcal{B}}, (s_0, q_0))$. Thus, $\mathcal{W}(\mathcal{A}_{\mathcal{T},\mathcal{B}}, \sigma_{\mathcal{A}}, (s_0, q_0)) \geq \mathcal{W}(\mathcal{T}, \sigma, s_0)$. As a result, $\mathcal{W}(\mathcal{A}_{\mathcal{T},\mathcal{B}}, \sigma_{\mathcal{A}}, (s_0, q_0)) = \mathcal{W}(\mathcal{T}, \sigma, s_0)$.

D.2 Proof of Theorem 10.2

This section proves Theorem 10.2. For this section, we assume the system is given as in Problem 10.1 in all lemmas. First, we show that when initial state and inputs have a bounded deviation from that of the optimal trajectory, the trajectory will have a bounded deviation from the optimal trajectory.

Lemma D.2. *There exist bounds $M_x > 0$ and $M_u > 0$ and constants $c_1, c_2 \geq 0$ that depend on $PS(\phi(x_0^*, \mathbf{u}^*))$, such that for all $\varepsilon_x \in [0, M_x]$ and $\varepsilon_u \in [0, M_u]$, if $x_0 \in \mathcal{B}_{\varepsilon_x}(x_0^*)$ and $u_t \in \mathcal{B}_{\varepsilon_u}(u_t^*)$ $\forall t \in \mathbb{N}$, where $\mathbf{u} = \{u_t\}_{t \in \mathbb{N}}$ and $\phi(x_0, \mathbf{u}) = \{x_t\}_{t \in \mathbb{N}}$, then for all $t \in \mathbb{N}$,*

$$\left\|x_{t+1} - x_{t+1}^*\right\|_\infty \leq c_1 \varepsilon_x + c_2 \varepsilon_u,$$
$$PS(\phi(x_0, \mathbf{u})) = PS(\phi(x_0^*, \mathbf{u}^*)).$$

Proof. For $x_p^* x_k^*$ (*i.e.*, $t \in [k]$), the lemma is true by Lemma C.1 because it is a finite length trajectory. Let c_1', c_2', M_x' and M_u' be the associated bounds for $x_p^* x_k^*$. When $t = k + m$, by robustness property, there exists a constant $L < 1$, and bounds $M_x'' \leq c_1' \varepsilon_x + c_2' \varepsilon_u$, and $M_u'' > 0$ such that $\left\|x_{k+m} - x_k^*\right\|_\infty \leq L \left\|x_k - x_k^*\right\|_\infty$. Let $R_k = \{x \mid x \in \mathcal{B}_{M_x''}(x_k^*)\}$, and $R_{k+j+1} = \{x \mid x = f_{k+j}(x', u), x' \in R_{k+j}, u \in \mathcal{B}_{M''_u}(u_{k+j}^*), x \in P_{i_{k+j+1}}\}$ for all $j \in \mathbb{N}$, where P_i is the label for the state x_i^*. The neighborhoods $\{R_{k+j+1}\}_{j \in \mathbb{N}}$ are compact because the transitions are continuous. Furthermore, by robustness property, $R_{k+m} \subset R_k$. Thus, for $j \in [m-1]$ and $i \in \mathbb{N}$, $R_{j+im+k} \subset R_{j+k}$. As a result, x_{j+im+k} is an element in R_{k+j} for all $j \in [m-1]$ and $i \in \mathbb{N}$. Let $d_t = \max\{\left\|x - x_t^*\right\|_\infty \mid x \in R_t\}$ be the maximum distance between a state in the neighborhood R_t and the optimal state x_t^* for all $t \geq k$. This maximum is well defined for all $t \geq k$ because $x_t^* \in R_t$. Now, let $\bar{d}$ be the maximum among all d_{k+j} for $j \in [m-1]$. Then, $\left\|x_t - x_t^*\right\| \leq \bar{d}$ for all $t \geq k$. Finally, set $M_x = \min\{M_x', M_x''\}$, $M_u = \min\{M_u', M_u''\}$, and c_1 and c_2 such that $c_1\varepsilon_x + c_2\varepsilon_u \geq \max\{\bar{d}, c_1'\varepsilon_x + c_2'\varepsilon_u\}$. Note that by construction, the sequence $\{R_{k+j+1}\}_{j\in\mathbb{N}}$ gives us $PS(\phi(x_0, \mathbf{u})) = PS(\phi(x_0^*, \mathbf{u}^*))$. □

Lemma D.2 ensures that the error from the optimal state at any time is bounded linearly by the error from the initial state and the largest error of control inputs from the optimal ones. The state error decreases to zero when ε_x and ε_u decrease to zero. The constants c_1 and c_2, which depend on t would not be unbounded by the robustness property. The main consequence of a bounded deviation is that the suboptimal cost of this trajectory is also bounded, showed next.

Lemma D.3. *Given the cost function in Problem 10.2, there exist bounds $M_x > 0$ and $M_u > 0$, and constants $c_3, c_4 \geq 0$ such that for all $\varepsilon_x \in [0, M_x]$ and $\varepsilon_u \in [0, M_u]$, if $x_0 \in \mathcal{B}_{\varepsilon_x}(x_0^*)$ and $u_t \in \mathcal{B}_{\varepsilon_u}(u_t^*)$ $\forall t \in \mathbb{N}$,*

$$|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq c_3\varepsilon_x + c_4\varepsilon_u,$$
$$PS(\phi(x_0, \mathbf{u})) = PS(\phi(x_0^*, \mathbf{u}^*)),$$

where $\phi(x_0, \mathbf{u}) = \{x_t\}_{t\in\mathbb{N}}$, and $\zeta = \{x_t u_t\}_{t\in\mathbb{N}}$.

Proof. First, compute

$$\begin{aligned}
&|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \\
&\quad = \left|\lim_{k\to\infty} \frac{1}{k}\sum_{t=0}^{k-1} \mathcal{J}(x_{t+1}, u_t) - \lim_{k\to\infty} \frac{1}{k}\sum_{t=0}^{k-1} \mathcal{J}(x_{t+1}^*, u_t^*)\right| \\
&\quad = \left|\lim_{k\to\infty} \frac{1}{k}\sum_{t=0}^{k-1} \mathcal{J}(x_{t+1}, u_t) - \mathcal{J}(x_{t+1}^*, u_t^*)\right| \\
&\quad \leq \lim_{k\to\infty} \frac{1}{k}\sum_{t=0}^{k-1} \left|\mathcal{J}(x_{t+1}, u_t) - \mathcal{J}(x_{t+1}^*, u_t^*)\right|.
\end{aligned}$$

The second equality holds because $\mathcal{J}(x_{t+1}, u_t)/k$ forms a convergent series for a lasso path. Since $\mathcal{J}(x, u)$ is a Lipschitz continuous function, there exists a constant $C_t > 0$ such that

$$|\mathcal{J}(x_{t+1}, u_t) - \mathcal{J}(x_{t+1}^*, u_t^*)| \leq C_t \left\|\bar{x}_t - \bar{x}_t^*\right\|_\infty.$$

Therefore,

$$|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq \lim_{k\to\infty} \frac{1}{k}\sum_{t=0}^{k-1} C \left\|\bar{x}_t - \bar{x}_t^*\right\|_\infty,$$

where $C = \sup_{t \in \mathbb{N}_+} C_t$ and $\bar{x}_t = [x_{t+1}, u_t] \in \mathbb{R}^{n+p}$ is a joined vector of x and u. Given that the optimal input is robust,

$$\begin{aligned} |\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| &\leq \lim_{k \to \infty} \frac{1}{k} \sum_{t=0}^{k-1} C \max\{c_1 \epsilon_x + c_2 \epsilon_u, \epsilon_u\} \\ &= C \max\{c_1 \epsilon_x + c_2 \epsilon_u, \epsilon_u\}. \end{aligned}$$

Then,

$$\begin{aligned} &|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq c_3 \epsilon_x + c_4 \epsilon_u \\ &c_3 = \begin{cases} c_1 C, & c_1 \epsilon_x + c_2 \epsilon_u \geq \epsilon_u \\ 0, & c_1 \epsilon_x + c_2 \epsilon_u < \epsilon_u \end{cases} \\ &c_4 = \begin{cases} c_2 C, & c_1 \epsilon_x + c_2 \epsilon_u \geq \epsilon_u \\ C, & c_1 \epsilon_x + c_2 \epsilon_u < \epsilon_u. \end{cases} \end{aligned}$$

The constants M_x and M_u are set to be the same as the ones in Lemma D.2. Then, $PS(\phi(x_0, \mathbf{u})) = PS(\phi(x_0^*, \mathbf{u}^*))$. □

Lemma D.3 implies that there exists a small neighborhood of the optimal trajectory in which the trajectories will go through the same partition sequence and difference in the cost is bounded and decreases to zero if ε_x and ε_u decrease to zero. Given a specific cost sub-optimality, we will now show that there exists a strategy that satisfies this cost error. In other words, we can construct an abstraction to give a chain lasso strategy that satisfies a certain cost error bound.

Lemma D.4. *Given any $\delta > 0$, there exists a chain lasso winning strategy σ for some $\hat{\mathcal{T}} = Abs(\mathcal{T}_{\mathcal{D}}, \equiv_X, \equiv_U)$ such that*

$$|\mathcal{W}(\hat{\mathcal{T}}, \sigma, \hat{x}_0) - \mathcal{W}(\mathcal{T}_{\mathcal{D}}, x_0^*, \Pi)| \leq \delta,$$

where $\hat{x}_0 = [x_0^*]_{\equiv_X}$.

Proof. The proof constructs the chain part of the chain lasso strategy followed by the loop part. For each part, we find neighborhoods N_t^x around x_t^* and N_t^u around u_t^* such that N_t^x is contained in the region of the partition containing x_t^*, all transitions from N_t^x and N_t^u lead to N_{t+1}^x. We also bound the maximum cost of any transition from N_t^x to N_{t+1}^x using an input from N_t^u. Lastly, we obtain a chain lasso strategy in $\hat{\mathcal{T}} = Abs(\mathcal{T}_{\mathcal{D}}, \equiv_X, \equiv_U)$ by choosing N_t^x and N_t^u to be regions of $\equiv_X$ and $\equiv_Y$.

More precisely, let $PS(\phi(x_0^*, \mathbf{u}^*)) = \{P_{i_t}\}_{t\in\mathbb{N}}$. By robustness of the optimal control (Definition 10.3), we first choose N_k^x to be a grid cell of size $\varepsilon_0/2^j \leq M_x$ for $j \in \mathbb{N}$ that contains an open ball around x_k^*, which is contained in P_{i_k}. Next, construct the chain part of the chain lasso strategy inductively, starting from $t = k$ and moving backwards similar to Lemma C.3. Under this construction, all executions from N_0^x will be in N_t^x after t steps. For all $t \in [k-1]$, choose the N_t^x and N_t^u such that they correspond to an element of a $\varepsilon_0/2^j$ grid for $j \in \mathbb{N}$.

Now, construct the loop part of the chain lasso strategy inductively. Let $R_k^x = N_k^x$ (white square), and for $t = k+1, \ldots, k+m$, let R_t^x be the region where all transitions from R_{t-1}^x on $R_{t-1}^u \subseteq \mathcal{B}_{M_u}(u_{t-1}^*)$ end in (white circles in Figure 10.2). Define the ε expansion of a set A as $\mathcal{E}_\varepsilon(A) = A \cup \{y \mid \min_{x\in A} \|y - x\|_\infty \leq \varepsilon\}$. The induction begins from $t = k + m$ and moves backwards until $t = k + 1$. At $t = k + m$, ε_{k+m} is chosen such that $E_{k+m} = \mathcal{E}_{\varepsilon_{k+m}}(R_{k+m}^x)$ and $E_{k+m} \subset R_k^x$. The latter is possible because of the second robustness property in Definition 10.3 for the optimal path. Assume that $E_j = \mathcal{E}_{\varepsilon_j}(R_j^x)$ (blue circles in Figure 10.2) for $j = t+1, \ldots, k+m$, and $F_j = \mathcal{E}_{\varepsilon_j'}(R_j^x)$ (red circles in Figure 10.2) for $j = t+1, \ldots, k+m-1$ are computed. Construct E_t and F_t by choosing ε_t' such that all trajectories from F_t on $N_t^u \subseteq R_t^u$ will end in E_{t+1}, and ε_t to be smaller than ε_t'. This construction is possible because of the continuity of the transition Δ. At $t = k$, let $N_k^u \subseteq R_k^u$. For all $t = k, \ldots, k+m-1$, N_t^u is chosen to correspond to an element of a $\varepsilon_0/2^j$ grid for $j \in \mathbb{N}$. Given $\{E_{k+j+1}\}_{j\in[m-2]}$ and $\{F_{k+j+1}\}_{j\in[m-2]}$, construct $\{N_{k+j+1}^x\}_{j\in[m-2]}$, whereby each N_{k+j+1}^x is a set of elements of a $\varepsilon_0/2^i$ grid for $i \in \mathbb{N}$ such that $E_t \in N_t^x$ and $N_t^x \in F_t$ (group of squares in Figure 10.2). Note that i may be different for different j. Then, every transition from N_t^x on N_t^u will end in N_{t+1}^x. Thus, the chain of neighborhoods $\{N_j^x\}_{j\in[k+m-1]}$ and $\{N_j^u\}_{j\in[k+m-1]}$ gives us a chain lasso strategy.

Further, by Lemma D.3, we can choose ε_x and ε_u such that $c_3\varepsilon_x + c_4\varepsilon_u \leq \delta$ for some constants c_3 and c_4 in order to ensure that the cost of the strategy is within δ of the optimal cost. Thus, $|\mathcal{W}(\zeta) - \mathcal{W}(\zeta^*)| \leq \delta$ for any path ζ starting in an ε_x ball around x_0^*.

Finally, define $\equiv_X$ and $\equiv_U$ such that the N_j^x and N_j^u are all equivalence classes of $\mathcal{X}$ and $\mathcal{U}$, respectively. Note that we need to ensure that for any $i, j \in [k+m-1]$, N_j^x is the same as N_i^x or the two are disjoint. Similar conditions hold for N_i^u. These conditions can be easily ensured during the construction by picking small enough $\varepsilon_0/2^j$ for $j \in \mathbb{N}$. □

Algorithm 10 contains only uniform grids with grid size $\varepsilon_0/2^i$. But, the partitions corresponding to the neighborhoods of N^x and N^u given by Lemma D.4 may not correspond to an uniform grid. Thus, we next construct a uniform grid by refining the chain lasso strategy obtained from Lemma D.4.

Lemma D.5. *For a given $\delta > 0$, there exists an $\varepsilon = \varepsilon_0/2^i > 0$, such that $|\mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi) - \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0^*, \Pi)| \leq \delta$, where $x_0^\varepsilon = [x_0^*]_{\equiv_X^\varepsilon}$. Furthermore, there exists a winning strategy σ_ε with cost no larger than $\mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi)$.*

Proof. From the proof of Lemma D.4, we obtain a sequence of neighborhoods N_t^x and N_t^u, which corresponds to a chain lasso strategy, say σ starting from N_0^x. Furthermore, every N_t^x corresponds to an element of $Grid(\mathcal{X}, \varepsilon_0/2^{i_t})$ or a collection of elements of $Grid(\mathcal{X}, \varepsilon_0/2^{i_t})$ for some i_t. Similarly, every N_t^u corresponds to an element of $Grid(\mathcal{U}, \varepsilon_0/2^{j_t})$ for some j_t. Let i be the maximum of the i_ts and j_ts. Then, $Grid(\mathcal{X}, \varepsilon_0/2^i)$ refines N_t^x, and $Grid(\mathcal{U}, \varepsilon_0/2^i)$ refines N_t^u. Define a strategy σ_ε (not necessarily a chain lasso anymore) for $\hat{\mathcal{T}}_\varepsilon$, which corresponds to following the neighborhoods N_t^x. All the paths in $\hat{\mathcal{T}}_\varepsilon$ conforming to σ_ε are contained in the neighborhoods N_t^x. As a result, the cost of σ_ε is bounded by that of σ, which is at most δ larger than the optimal cost. Therefore, the optimal cost of $\hat{\mathcal{T}}_\varepsilon$ is at most δ larger than that of $\mathcal{T}_\mathcal{D}$. □

Proof of Theorem 10.2. Note that $J_{\varepsilon_0/2^i} \leq J_{\varepsilon_0/2^j}$ for all $i > j$, and for any $\delta > 0$, there exists $\varepsilon = \varepsilon_0/2^i$, such that $|\mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi) - \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0^*, \Pi)| \leq \delta$ by Lemma D.5. Because the optimal path is a lasso, the optimal strategy has a finite memory, and thus the optimality gap ε is zero. As a result, $J_\varepsilon = \mathcal{W}(\hat{\mathcal{T}}_\varepsilon, \hat{x}_0^\varepsilon, \Pi)$ and $J_{opt} = \mathcal{W}(\mathcal{T}_\mathcal{D}, x_0^*, \Pi)$ is the optimal cost. Thus, $|J_\varepsilon - J_{opt}| \leq \delta$. Therefore, $J_{\varepsilon_0/2^i}$ converges to J_{opt} as i goes to infinity. Furthermore, for each sub-optimal cost $J_{\varepsilon_0/2^i}$, there exists a suboptimal winning strategy $\sigma_{\varepsilon_0/2^i}$ with cost of winning $J_{\varepsilon_0/2^i}$ by Lemma D.5.

Chapter 11

CONCLUSION

Optimal controller synthesis for nonlinear systems is a challenging problem. Due to the nonlinearity, many efficient techniques that are developed for linear systems are no longer applicable for the nonlinear counterparts. As a result, most techniques for nonlinear systems are extremely ad-hoc to specific systems and rarely scalable to real world engineering systems such as robots. This thesis aims to push the boundary further by developing optimal controller synthesis techniques that are more scalable and guarantee correctness of the specifications. This thesis focuses on two general classes of nonlinear systems: linearly solvable nonlinear systems and hybrid nonlinear systems.

Part I Optimal Control Synthesis for Linearly Solvable Nonlinear Systems

The first part of this thesis discusses the techniques for linearly solvable systems. Suboptimal controller is synthesized using SOS programming with performance guarantees for first exit and finite horizon problems. This technique is the first to combine the optimality condition given by the HJB equation with stability condition given by the Lyapunov analysis. However, this method suffers from the curse of dimensionality. Thus, this thesis proposes a low rank tensor decomposition based technique that scales linearly with dimensions to synthesize controllers for first exit, finite horizon and infinite horizon problems. The existing ALS algorithm is improved to avoid ill-conditioning issues, and A MATLAB tool that implements the algorithms is developed [32]. The ability to compute the solution to the linear HJB equation for a quadcopter with twelve dimensions using a personal laptop and produce a controller that achieves the objective is a strong indicator of this technique's great potential for implementation in other robotics and engineering systems.

Apart from the potential improvements and future works discussed in the individual chapters, the techniques presented in this thesis are general and applicable for solving any linear PDE, including the Fokker-Planck equation [114] used for uncertainty propagation and state estimation. In fact, any equation that is naturally a linear PDE, including those found in fluid mechanics, quantum mechanics, and solid mechanics could benefit from the technique presented in this thesis.

Furthermore, the techniques presented so far are restricted to linearly solvable systems that have linear HJB equation. One major improvement for this line of research is to consider the nonlinear HJB equation directly. Solving the nonlinear HJB equation directly is an attractive approach for multiple reasons: the desirability function can have really poor numerics despite that the value function has a relatively "nice" shape and magnitude, and the classes of systems and cost functions that can benefit from this approach become larger. To solve the nonlinear HJB equation, a creative combination of numerical techniques in nonlinear PDE with the low rank tensor representations or the SOS programs is necessary.

Part II Optimal Control Synthesis for Hybrid Systems with Qualitative and Quantitative Objectives

Apart from optimality, in many applications, correctness of the system behavior is also necessary. In other words, qualitative specifications such as obstacle avoidance and surveillance are essential in addition to quantitative specifications such as minimizing energy used. The second part of this thesis focuses on synthesizing optimal controller for hybrid nonlinear systems with regular and $\omega-$regular objectives, where regular objectives describe finite time behavior, and $\omega-$regular objectives describe long term behavior. An abstraction-refinement based approach that preserves the cost is developed for computing the optimal controller. The algorithms converge to the optimal cost if the systems are robust with respect to the initial states and the optimal inputs. The resulting controller is guaranteed to be correct and the system performance is guaranteed to satisfy a given cost bound. A Python tool that implements the algorithms is developed [33].

This technique consider a specific type of hybrid system, where the states and control inputs are continuous variables. But, the fundamental structure that allows for the construction of cost preserving abstraction is that the system can be represented as a weighted transition system. Thus, this technique can potentially be applied in more general setting, where some of the states and control inputs are discrete variables. Furthermore, the cost preserving abstraction may be combined with other abstraction approaches that preserve other system properties, for example stability [163], in order to encode optimality in the framework.

Another potential future direction of this work is extending the technique to robust optimal control synthesis for nondeterministic hybrid systems. The neighborhoods of states and inputs in the abstract system naturally model measurement errors and input uncertainties of the concrete system. Therefore, this technique may apply for synthesizing robust optimal control for a stochastic hybrid system.

Another interesting future extension of this technique is to consider continuous time systems. Currently, a continuous time system can be discretized in time and the approach described in this thesis will apply. However, a more careful examination of the theoretical results is necessary as the time discretization introduces another layer of abstraction. An interesting approach is to develop the cost preserving abstraction-refinement procedure for the continuous time system directly without discretizing the system a priori.

In addition, the methods presented in this thesis rely on space discretization that scales exponentially with dimension. This curse of dimensionality prevents the application of these algorithms to large scale systems. In order to alleviate this curse, a more intelligent gridding scheme in the refinement step could be developed, for example, counter-example guided refinement [161]. Alternatively, a more efficient abstraction procedure could be developed, for example, optimizing the grid size to minimize the number of transitions in the abstraction subject to the appropriate dynamic constraints [162]. Alternatively, a distributed scheme could be developed such that the computation can be performed in a more efficient manner such as parallelization.

Lastly, more generally, the approaches in Part I could potentially be combined with the abstraction based approach introduced in Part II for studying the properties of the new alternating simulation relation defined in Chapter 8. Lyapunov-type approaches have been used in the past to construct and analyze approximate bisimulations that reduces complexity of safety verification for both deterministic and nondeterministic hybrid systems [164], [165].

BIBLIOGRAPHY

[1] D. S. Bernstein, "Feedback control: An invisible thread in the history of technology," *IEEE Control Systems*, vol. 22, no. 2, pp. 53–68, Apr. 2002. DOI: `10.1109/37.993315`.

[2] O. Mayr, *The Origins of Feedback Control*. Cambridge, MA, USA: MIT Press, 1970.

[3] W. L. Luyben, *Process modeling, simulation and control for chemical engineers*. McGraw-Hill Higher Education, 1989.

[4] A. Abdessameud and A. Tayebi, "Global trajectory tracking control of vtol-uavs without linear velocity measurements," *Automatica*, vol. 46, no. 6, pp. 1053–1059, 2010.

[5] J. R. Wertz, *Spacecraft attitude determination and control*. Springer Science & Business Media, 2012, vol. 73.

[6] J. W. Grizzle, C. Chevallereau, R. W. Sinnet, and A. D. Ames, "Models, feedback control, and open problems of 3d bipedal robotic walking," *Automatica*, vol. 50, no. 8, pp. 1955–1988, 2014.

[7] L. Lessard, B. Recht, and A. Packard, "Analysis and design of optimization algorithms via integral quadratic constraints," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 57–95, 2016.

[8] F. A. Chandra, G. Buzi, and J. C. Doyle, "Glycolytic oscillations and limits on robust efficiency," *Science*, vol. 333, no. 6039, pp. 187–192, 2011.

[9] N. Li, J. Cruz, C. S. Chien, S. Sojoudi, B. Recht, D. Stone, M. Csete, D. Bahmiller, and J. C. Doyle, "Robust efficiency and actuator saturation explain healthy heart rate control and variability," *Proceedings of the National Academy of Sciences*, vol. 111, no. 33, E3476–E3485, 2014.

[10] E. Todorov, "Stochastic optimal control and estimation methods adapted to the noise characteristics of the sensorimotor system," *Neural Computation*, vol. 17, no. 5, pp. 1084–1108, 2005.

[11] D. W. Franklin and D. M. Wolpert, "Computational mechanisms of sensorimotor control," *Neuron*, vol. 72, no. 3, pp. 425–442, 2011.

[12] R. J. Peterka, "Sensorimotor integration in human postural control," en, *Journal of Neurophysiology*, vol. 88, no. 3, pp. 1097–1118, Sep. 2002.

[13] R. Kalman, "Contributions to the theory of optimal control," *Bol. de Soc. Math. Mexicana*, 1960.

[14] J. Doyle, "Guaranteed margins for lqg regulators," *IEEE Transactions on Automatic Control*, vol. 23, no. 4, pp. 756–757, 1978.

[15] G. Zames, “Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms, and approximate inverses,” *IEEE Transactions on Automatic Control*, vol. 26, no. 2, pp. 301–320, 1981.

[16] J. C. Doyle, K. Glover, P. P. Khargonekar, and B. A. Francis, “State-space solutions to standard h/sub 2/and h/sub infinity/control problems,” *IEEE Transactions on Automatic control*, vol. 34, no. 8, pp. 831–847, 1989.

[17] Y.-S. Wang, “A system level approach to optimal controller design for large-scale distributed systems,” PhD thesis, California Institute of Technology, 2017.

[18] Y.-S. Wang and N. Matni, “Localized lqg optimal control for large-scale systems,” in *American Controls Conf. (ACC)*, 2016, pp. 1954–1961.

[19] R. Bellman, “On the theory of dynamic programming,” *Proceedings of the National Academy of Sciences*, vol. 38, no. 8, pp. 716–719, 1952.

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

[21] H. J. Kushner and F. C. Schweppe, “A maximum principle for stochastic control systems,” *Journal of Mathematical Analysis and Applications*, vol. 8, no. 2, pp. 287–302, 1964.

[22] R. E. Bellman and S. E. Dreyfus, *Applied Dynamic Programming*. Princeton University Press, 1962.

[23] L. R. G. Carrillo, A. E. D. López, R. Lozano, and C. Pégard, “Modeling the quad-rotor mini-rotorcraft,” in *Quad Rotorcraft Control*, Springer, 2013, pp. 23–34.

[24] H. K. Khalil, “Nonlinear systems,” *Prentice-Hall, New Jersey*, vol. 2, no. 5, pp. 5–1, 1996.

[25] A. Bloch, J. Baillieul, P. Crouch, J. E. Marsden, D. Zenkov, P. S. Krishnaprasad, and R. M. Murray, *Nonholonomic mechanics and control*. Springer, 2003, vol. 24.

[26] S. Prajna, A. Papachristodoulou, and F. Wu, “Nonlinear control synthesis by sum of squares optimization: A lyapunov-based approach,” in *Asian Control Conference*, vol. 1, 2004, pp. 157–165.

[27] P. A. Parrilo, “Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization,” PhD thesis, California Institute of Technology, 2000.

[28] C. Seatzu, D. Gromov, J. Raisch, D. Corona, and A. Giua, “Optimal control of discrete-time hybrid automata under safety and liveness constraints,” *Nonlinear Analysis: Theory, Methods & Applications*, vol. 65, no. 6, pp. 1188–1210, 2006.

[29] E. A. Gol, M. Lazar, and C. Belta, "Temporal logic model predictive control," *Automatica*, vol. 56, pp. 78–85, 2015.

[30] M. Svoreňová, J. Křetínský, M. Chmelík, K. Chatterjee, I. Černá, and C. Belta, "Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games," *Nonlinear Analysis: Hybrid Systems*, vol. 23, pp. 230–253, 2017.

[31] J. Liu, N. Ozay, U. Topcu, and R. M. Murray, "Synthesis of reactive switching protocols from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 58, no. 7, pp. 1771–1785, 2013. DOI: `10.1109/TAC.2013.2246095`.

[32] *Sequential Alternating Least Squares (SeALS) in MATLAB*, `https://github.com/ypleong/SeALS`.

[33] *OptCAR*, `https://github.com/ypleong/OPTCAR`.

[34] Y. P. Leong, M. B. Horowitz, and J. W. Burdick, "Linearly solvable stochastic control lyapunov functions," *SIAM Journal on Control and Optimization*, vol. 54, no. 6, pp. 3106–3125, 2016. DOI: `10.1137/16M105767X`,

[35] E. Stefansson and Y. P. Leong, "Sequential alternating least squares for solving high dimensional linear Hamilton-Jacobi-Bellman equation," in *IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2016, pp. 3757–3764. DOI: `10.1109/IROS.2016.7759553`,

[36] Y. P. Leong, M. B. Horowitz, and J. W. Burdick, "Suboptimal stabilizing controllers for linearly solvable system," in *IEEE Int. Conf. on Decision and Control (CDC)*, 2015, pp. 7157–7164. DOI: `10.1109/CDC.2015.7403348`,

[37] P. A. Parrilo, "Semidefinite programming relaxations for semialgebraic problems," *Mathematical Programming*, vol. 96, no. 2, pp. 293–320, 2003.

[38] A. Papachristodoulou and S. Prajna, "Analysis of non-polynomial systems using the sum of squares decomposition," in *Positive Polynomials in Control*, Springer, 2005, pp. 23–43.

[39] M. Krstic, I. Kanellakopoulos, and P. V. Kokotovic, *Nonlinear and adaptive control design*. Wiley, 1995.

[40] R. A. Freeman and J. A. Primbs, "Control Lyapunov functions: New ideas from an old source," in *IEEE Int. Conf. on Decision and Control (CDC)*, vol. 4, 1996, pp. 3926–3931.

[41] E. D. Sontag, "A lyapunov-like characterization of asymptotic controllability," *SIAM Journal on Control and Optimization*, vol. 21, no. 3, pp. 462–471, 1983.

[42] J. A. Primbs, V. Nevistić, and J. C. Doyle, "Nonlinear optimal control: A control Lyapunov function and receding horizon perspective," *Asian Journal of Control*, vol. 1, no. 1, pp. 14–24, 1999.

[43] S. Kolathaya and A. D. Ames, "Exponential convergence of a unified CLF controller for robotic systems under parameter uncertainty," in *American Controls Conf. (ACC)*, 2014, pp. 3710–3715.

[44] P. Ogren, M. Egerstedt, and X. Hu, "A control Lyapunov function approach to multi-agent coordination," in *Proceedings of the 40th IEEE Conference on Decision and Control*, vol. 2, 2001, pp. 1150–1155.

[45] H. Deng and M. Krstić, "Stochastic nonlinear stabilization—i: A backstepping design," *Systems & Control Letters*, vol. 32, no. 3, pp. 143–150, 1997.

[46] P. Florchinger, "Feedback stabilization of affine in the control stochastic differential systems by the control lyapunov function method," *SIAM Journal on Control and optimization*, vol. 35, no. 2, pp. 500–511, 1997.

[47] R. Khasminskii, *Stochastic Stability of Differential Equations*. Springer Science & Business Media, 2011, vol. 66.

[48] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 2. Belmont, MA: Athena Scientific, 1995, vol. 1.

[49] J. B. Lasserre, D. Henrion, C. Prieur, and E. Trélat, "Nonlinear optimal control via occupation measures and LMI-relaxations," *SIAM Journal on Control and Optimization*, vol. 47, no. 4, pp. 1643–1666, 2008.

[50] W. H. Fleming, *Logarithmic transformations and stochastic control*. Springer, 1982.

[51] C. J. Holland, "A new energy characterization of the smallest eigenvalue of the schrödinger equation," *Communications on Pure and Applied Mathematics*, vol. 30, no. 6, pp. 755–765, 1977.

[52] P. Dai Pra, "A stochastic control approach to reciprocal diffusion processes," *Applied mathematics and Optimization*, vol. 23, no. 1, pp. 313–329, 1991.

[53] R. Filliger and M.-O. Hongler, "Relative entropy and efficiency measure for diffusion-mediated transport processes," *Journal of Physics A: Mathematical and General*, vol. 38, no. 6, p. 1247, 2005.

[54] H. J. Kappen, "Path integrals and symmetry breaking for optimal control theory," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 11, 2005.

[55] E. Todorov, "Efficient computation of optimal actions," *Proceedings of the National Academy of Sciences (PNAS)*, vol. 106, no. 28, pp. 11 478–11 483, 2009.

[56] K. Dvijotham and E. Todorov, "Linearly-solvable optimal control," *Reinforcement learning and approximate dynamic programming for feedback control*, pp. 119–141, 2012.

[57] E. Theodorou, “Iterative path integral stochastic optimal control: Theory and applications to motor control,” PhD thesis, University of Southern California, 2011.

[58] F. Stulp, E. A. 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.

[59] W. Wiegerinck and B. Broek, “Stochastic optimal control in continuous spacetime multi-agent systems,” in *In Proceedings UAI*, 2006.

[60] P. Rutquist, T. Wik, and C. Breitholtz, “Solving the Hamilton-Jacobi-Bellman equation for a stochastic system with state constraints,” in *IEEE Int. Conf. on Decision and Control (CDC)*, Dec. 2014.

[61] W. H. Fleming and H. M. Soner, *Controlled Markov processes and viscosity solutions*. New York: Springer, 2006, vol. 25.

[62] E. Theodorou, F. Stulp, J. Buchli, and S. Schaal, “An iterative path integral stochastic optimal control approach for learning robotic tasks,” in *World Congress*, vol. 18, 2011, pp. 11 594–11 601.

[63] H. Kappen, “Linear theory for control of nonlinear stochastic systems,” *Physical Review Letters*, vol. 95, no. 20, 2005.

[64] E. Lavretsky and K. Wise, *Robust and Adaptive Control: With Aerospace Applications*. Springer Science & Business Media, 2012.

[65] W. A. Strauss, *Partial Differential Equations: An Introduction*, 2nd ed. John Wiley & Sons, Inc., 2008.

[66] M. G. Crandall, H. Ishii, and P.-L. Lions, “User’s guide to viscosity solutions of second order partial differential equations,” *Bulletin of the American Mathematical Society*, vol. 27, no. 1, pp. 1–67, 1992.

[67] A. Papachristodoulou and S. Prajna, “A tutorial on sum of squares techniques for systems analysis,” in *American Controls Conf. (ACC)*, 2005, pp. 2686–2700.

[68] J. B. Lasserre, “Global optimization with polynomials and the problem of moments,” *SIAM Journal on Optimization*, vol. 11, no. 3, pp. 796–817, 2001.

[69] G. D. Smith, *Numerical solution of partial differential equations: finite difference methods*. Oxford university press, 1985.

[70] J. C. Strikwerda, *Finite difference schemes and partial differential equations*. SIAM, 2004.

[71] B. Fornberg, “Generation of finite difference formulas on arbitrarily spaced grids,” *Mathematics of computation*, vol. 51, no. 184, pp. 699–706, 1988.

[72] J. P. Boyd, *Chebyshev and Fourier Spectral Methods*. Courier Corporation, 2001.

[73] R. Baltensperger and M. R. Trummer, "Spectral differencing with a twist," *SIAM Journal on Scientific Computing*, vol. 24, no. 5, pp. 1465–1487, 2003. DOI: 10.1137/S1064827501388182.

[74] K. W. Morton and D. F. Mayers, *Numerical Solution of Partial Differential Equations: An Introduction*, 2nd ed. New York: Cambridge University Press, 2005.

[75] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[76] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis," *University of California at Los Angeles*, 1970.

[77] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[78] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, Sep. 1966. DOI: 10.1007/bf02289464.

[79] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011. DOI: 10.1137/090752286.

[80] A. A. Gorodetsky, S. Karaman, and Y. M. Marzouk, "Function-train: A continuous analogue of the tensor-train decomposition," *arXiv:1510.09088*, 2016.

[81] G. Beylkin and M. Mohlenkamp, "Algorithms for numerical analysis in high dimensions," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 2133–2159, 2005.

[82] B. W. Bader, T. G. Kolda, *et al.*, *Matlab tensor toolbox version 2.6*, http://www.sandia.gov/~tgkolda/TensorToolbox/, Feb. 2015.

[83] B. W. Bader and T. G. Kolda, "Algorithm 862: MATLAB tensor classes for fast algorithm prototyping," *ACM Transactions on Mathematical Software*, vol. 32, no. 4, pp. 635–653, Dec. 2006. DOI: 10.1145/1186785.1186794.

[84] M. J. Reynolds, A. Doostan, and G. Beylkin, "Randomized alternating least squares for canonical tensor decompositions: Application to a pde with random data," *SIAM Journal on Scientific Computing*, vol. 38, no. 5, A2634–A2664, 2016.

[85] M. Horowitz and J. Burdick, "Semidefinite relaxations for stochastic optimal control policies," in *American Controls Conf. (ACC)*, Jun. 2014, pp. 3006–3012. DOI: 10.1109/ACC.2014.6859382.

[86] M. B. Horowitz, I. Papusha, and J. W. Burdick, “Domain decomposition for stochastic optimal control,” in *IEEE Int. Conf. on Decision and Control (CDC)*, 2014.

[87] W. Rudin, *Principles of Mathematical Analysis*. McGraw-Hill, New York, 1964, vol. 3.

[88] A. Papachristodoulou, “Scalable analysis of nonlinear systems using convex optimization,” PhD thesis, California Institute of Technology, 2005.

[89] J. Lofberg, “YALMIP : a toolbox for modeling and optimization in MATLAB,” in *IEEE International Symposium on Computer Aided Control Systems Design*, 2004, pp. 284–289.

[90] E. D. Andersen and K. D. Andersen, “The MOSEK interior point optimizer for linear programming: An implementation of the homogeneous algorithm,” in *High performance optimization*, Springer, 2000, pp. 197–232.

[91] Wolfram Research, Inc., *Mathematica 10.0*. Champaign, Illinois, 2014.

[92] A. A. Ahmadi and A. Majumdar, “Dsos and sdsos optimization: Lp and socp-based alternatives to sum of squares optimization,” in *2014 48th Annual Conference on Information Sciences and Systems (CISS)*, Mar. 2014, pp. 1–5. DOI: 10.1109/CISS.2014.6814141.

[93] A. A. Ahmadi and A. Majumdar, “Dsos and sdsos optimization: More tractable alternatives to sum of squares and semidefinite optimization,” *arXiv:1706.02586*, 2017.

[94] V. Chandrasekaran and P. Shah, “Relative Entropy Relaxations for Signomial Optimization,” *arXiv:1409.7640*, 2014.

[95] M. B. Horowitz, A. Damle, and J. W. Burdick, “Linear Hamilton Jacobi Bellman Equations in High Dimensions,” in *IEEE Int. Conf. on Decision and Control (CDC)*, 2014.

[96] F. Permenter, P. Parrilo, *et al.*, “Basis selection for sos programs via facial reduction and polyhedral approximations,” in *IEEE Int. Conf. on Decision and Control (CDC)*, 2014, pp. 6615–6620.

[97] A. A. Ahmadi, P. Parrilo, *et al.*, “Towards scalable algorithms with formal guarantees for lyapunov analysis of control systems via algebraic optimization,” in *IEEE Int. Conf. on Decision and Control (CDC)*, 2014, pp. 2272–2281.

[98] D. Papp and S. Yıldız, “Sum-of-squares optimization without semidefinite programming,” *arXiv:1712.01792*, Dec. 2017.

[99] J. Garcke and A. Kröner, “Suboptimal feedback control of PDEs by solving HJB equations on adaptive sparse grids,” INRIA Saclay, Research Report, 2015.

[100] C. O. Aguilar and A. J. Krener, “Numerical solutions to the Bellman equation of optimal control,” *Journal of optimization theory and applications*, vol. 160, no. 2, pp. 527–552, 2014.

[101] W. M. McEneaney, “Convergence rate for a curse-of-dimensionality-free method for Hamilton-Jacobi-Bellman PDEs represented as maxima of quadratic forms,” *SIAM Journal on Control and Optimization*, vol. 48, no. 4, pp. 2651–2685, 2009.

[102] S. Gombao, “Approximation of optimal controls for semilinear parabolic PDE by solving Hamilton-Jacobi-Bellman equations,” in *Proc. of the 15th International Symposium on the Mathematical Theory of Networks and Systems, University of Notre Dame, South Bend, Indiana, USA*, Citeseer, 2002.

[103] A. Gorodetsky, S. Karaman, and Y. Marzouk, “Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition,” in *Proceedings of Robotics: Science and Systems*, Rome, Italy, Jul. 2015.

[104] J. Han, A. Jentzen, and W. E, “Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning,” *arXiv:1707.02568*, 2017.

[105] Y. T. Chow, J. Darbon, S. Osher, and W. Yin, “Algorithm for overcoming the curse of dimensionality for state-dependent hamilton-jacobi equations,” *arXiv:1704.02524*, 2017.

[106] D. J. Biagioni, D. Beylkin, and G. Beylkin, “Randomized interpolative decomposition of separated representations,” *Journal of Computational Physics*, vol. 281, pp. 116–134, 2015.

[107] A. Jadbabaie, “Receding horizon control of nonlinear systems: A control lyapunov function approach.,” PhD thesis, California Institute of Technology, 2001.

[108] G. Peters and J. H. Wilkinson, “Inverse iteration, ill-conditioned equations and newton’s method,” *SIAM review*, vol. 21, no. 3, pp. 339–360, 1979.

[109] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[110] E. Stefansson and Y. P. Leong, *Sequential Alternating Least Squares (SeALS) MATLAB User’s Guide*, `https://github.com/ypleong/SeALS`, Online, 2016.

[111] A. Jameson, “Analysis and design of numerical schemes for gas dynamics, 1: Artificial diffusion, upwind biasing, limiters and their effect on accuracy and multigrid convergence,” *International Journal of Computational Fluid Dynamics*, vol. 4, no. 3-4, pp. 171–218, 1995.

[112] B. Sun and B.-Z. Guo, "Convergence of an upwind finite-difference scheme for Hamilton-Jacobi-Bellman equation in optimal control," *IEEE Transactions on Automatic Control*, vol. 60, no. 11, pp. 3012–3017, 2015.

[113] R. Yu and Y. Liu, "Learning from multiway data: Simple and efficient tensor regression," in *International Conference on Machine Learning*, 2016, pp. 373–381.

[114] H. Risken, "Fokker-planck equation," in *The Fokker-Planck Equation*, Springer, 1996, pp. 63–95.

[115] Y. Sun and M. Kumar, "Numerical solution of high dimensional stationary fokker–planck equations via tensor decomposition and chebyshev spectral differentiation," *Computers & Mathematics with Applications*, vol. 67, no. 10, pp. 1960–1977, 2014.

[116] ——, "A tensor decomposition approach to high dimensional stationary fokker-planck equations," in *acc*, 2014, pp. 4500–4505.

[117] ——, "Nonlinear bayesian filtering based on fokker-planck equation and tensor decomposition," in *18th International Conference on Information Fusion (Fusion)*, 2015, pp. 1483–1488.

[118] ——, "Uncertainty propagation in orbital mechanics via tensor decomposition," *Celestial Mechanics and Dynamical Astronomy*, vol. 124, no. 3, pp. 269–294, 2016.

[119] M. Chen and C. J. Tomlin, "Exact and efficient hamilton-jacobi reachability for decoupled systems," in *IEEE Int. Conf. on Decision and Control (CDC)*, 2015, pp. 1297–1303.

[120] H. M. Osinga and J. Hauser, "The geometry of the solution set of nonlinear optimal control problems," *Journal of Dynamics and Differential Equations*, vol. 18, no. 4, pp. 881–900, 2006.

[121] J. Hauser, S. Sastry, and G. Meyer, "Nonlinear control design for slightly non-minimum phase systems: Application to V/STOL aircraft," *Automatica*, vol. 28, no. 4, pp. 665–679, 1992.

[122] K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[123] S. L. Smith, J. Tumová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[124] E. M. Wolff and R. M. Murray, "Optimal control of nonlinear systems with temporal logic specifications," in *Robotics Research*, Springer, 2016, pp. 21–37.

[125] T. Wongpiromsarn, U. Topcu, and R. M. Murray, “Receding horizon temporal logic planning,” *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012. DOI: 10.1109/TAC.2012.2195811.

[126] A. Church, “Logic, arithmetic, and automata,” in *Proc. Internat. Congr. Mathematicians (Stockholm)*, Djursholm: Inst. Mittag-Leffler, 1962, pp. 23–35.

[127] Z. Manna and P. Wolper, “Synthesis of communicating processes from temporal logic specifications,” in *Logics of Programs, Workshop, Yorktown Heights, New York*, May 1981, pp. 253–281. DOI: 10.1007/BFb0025786.

[128] E. M. Clarke and E. A. Emerson, “Design and synthesis of synchronization skeletons using branching-time temporal logic,” in *Logics of Programs, Workshop, Yorktown Heights, New York*, May 1981, pp. 52–71. DOI: 10.1007/BFb0025774.

[129] P. J. Ramadge and W. M. Wonham, “Supervisory control of a class of discrete event processes,” *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, Jan. 1987. DOI: 10.1137/0325013.

[130] E. Asarin, O. Maler, and A. Pnueli, “Symbolic controller synthesis for discrete and timed systems,” in *Hybrid Systems II*, 1994, pp. 1–20.

[131] T. A. Henzinger, B. Horowitz, and R. Majumdar, “Rectangular hybrid games,” in *Proceedings of CONCUR ’99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands*, Aug. 1999, pp. 320–335.

[132] P. Bouyer, T. Brihaye, and F. Chevalier, “O-minimal hybrid reachability games,” *Logical Methods in Computer Science*, vol. 6, no. 1, 2010.

[133] V. Vladimerou, P. Prabhakar, M. Viswanathan, and G. E. Dullerud, “Specifications for decidable hybrid games,” *Theor. Comput. Sci.*, vol. 412, no. 48, pp. 6770–6785, 2011. DOI: 10.1016/j.tcs.2011.08.036.

[134] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, “What’s decidable about hybrid automata?” In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, ser. STOC ’95, Las Vegas, Nevada, USA: ACM, 1995, pp. 373–382. DOI: 10.1145/225058.225162.

[135] J. Raisch and S. O’Young, “Discrete approximation and supervisory control of continuous systems,” *IEEE Transactions on Automatic Control, Special Issue on Hybrid Systems*, vol. 43, no. 4, pp. 569–573, 1998.

[136] J. A. DeCastro and H. Kress-Gazit, “Guaranteeing reactive high-level behaviors for robots with complex dynamics,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 749–756. DOI: 10.1109/IROS.2013.6696435.

[137] M. Kloetzer and C. Belta, "Temporal logic planning and control of robotic swarms by hierarchical abstractions," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 320–330, 2007. DOI: 10.1109/TRO.2006.889492.

[138] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications," in *American Controls Conf. (ACC)*, 2012, pp. 735–742.

[139] A. Girard, "Controller synthesis for safety and reachability via approximate bisimulation," *Automatica*, vol. 48, no. 5, pp. 947–953, 2012.

[140] M. Mazo and P. Tabuada, "Symbolic approximate time-optimal control," *Systems & Control Letters*, vol. 60, no. 4, pp. 256–263, 2011.

[141] G. Reissig and M. Rungger, "Abstraction-based solution of optimal stopping problems under uncertainty," in *IEEE Int. Conf. on Decision and Control (CDC)*, 2013, pp. 3190–3196.

[142] A. Bemporad, F. Borrelli, and M. Morari, "Piecewise linear optimal controllers for hybrid systems," in *American Controls Conf. (ACC)*, vol. 2, 2000, 1190–1194 vol.2. DOI: 10.1109/ACC.2000.876688.

[143] B. Lincoln and B. Bernhardsson, "LQR optimization of linear system switching," *IEEE Trans. Automat. Contr.*, vol. 47, no. 10, pp. 1701–1705, 2002. DOI: 10.1109/TAC.2002.803539.

[144] F. de Roo and M. Mazo, "On symbolic optimal control via approximate simulation relations," in *IEEE Int. Conf. on Decision and Control (CDC)*, 2013, pp. 3205–3210.

[145] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimal control of non-deterministic systems for a computationally efficient fragment of temporal logic," in *IEEE Int. Conf. on Decision and Control (CDC)*, 2013, pp. 3197–3204.

[146] E. M. Wolff, "Optimal control with weighted average costs and temporal logic specifications," *Proc. of Robotics: Science and Systems VIII*, 2012.

[147] G. Reissig and M. Rungger, "Symbolic optimal control," *arXiv:1709.07333*, 2017.

[148] M. Rungger, G. Reissig, and M. Zamani, "Symbolic synthesis with average performance guarantees," in *IEEE Int. Conf. on Decision and Control (CDC)*, 2016, pp. 7404–7410.

[149] S. Karaman, R. G. Sanfelice, and E. Frazzoli, "Optimal control of mixed logical dynamical systems with linear temporal logic specifications," in *IEEE Int. Conf. on Decision and Control (CDC)*, 2008, pp. 2117–2122.

[150] K. Chatterjee and L. Doyen, "Energy parity games," *Theoretical Computer Science*, vol. 458, pp. 49–60, 2012.

[151] K. Chatterjee, T. A. Henzinger, and M. Jurdzinski, "Mean-payoff parity games," in *IEEE Symposium on Logic in Computer Science (LICS)*, 2005, pp. 178–187.

[152] L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin, "Faster algorithms for mean-payoff games," *Formal methods in system design*, vol. 38, no. 2, pp. 97–118, 2011.

[153] Y. P. Leong and P. Prabhakar, "Optimal control with regular objectives using an abstraction-refinement approach," in *American Controls Conf. (ACC)*, 2016, pp. 5161–5168. DOI: 10.1109/ACC.2016.7526478,

[154] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces.," in *IJCAI*, vol. 13, 2013, pp. 854–860.

[155] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification (preliminary report)," in *Proceedings of the Symposium on Logic in Computer Science*, Jun. 1986, pp. 332–344.

[156] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi, "Alternating refinement relations," in *Proceedings of the Ninth International Conference on Concurrency Theory (CONCUR'98)*, vol. 1466, Springer-Verlag, 1998, pp. 163–178.

[157] R. Bellman, "On a routing problem," DTIC Document, Tech. Rep., 1956.

[158] R. Bagnara, P. M. Hill, and E. Zaffanella, "The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems," *Science of Computer Programming*, vol. 72, no. 1–2, pp. 3–21, 2008.

[159] Y. Tazaki and J. Imura, "Discrete abstractions of nonlinear systems based on error propagation analysis," *IEEE Transactions on Automatic Control*, vol. 57, no. 3, pp. 550–564, Mar. 2012. DOI: 10.1109/TAC.2011.2161789.

[160] B. Yordanov, J. Tumová, I. Cerna, J. Barnat, and C. Belta, "Temporal logic control of discrete-time piecewise affine systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1491–1504, Jun. 2012. DOI: 10.1109/TAC.2011.2178328.

[161] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *International Conference on Computer Aided Verification*, Springer, 2000, pp. 154–169.

[162] A. Weber, M. Rungger, and G. Reissig, "Optimized state space grids for abstractions," *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5816–5821, 2016.

[163] P. Prabhakar, G. Dullerud, and M. Viswanathan, "Stability preserving simulations and bisimulations for hybrid systems," *IEEE Transactions on Automatic Control*, vol. 60, no. 12, pp. 3210–3225, 2015.

[164] A. Girard and G. J. Pappas, “Approximate bisimulation relations for constrained linear systems,” *Automatica*, vol. 43, no. 8, pp. 1307–1317, 2007.

[165] ——, “Approximation metrics for discrete and continuous systems,” *IEEE Transactions on Automatic Control*, vol. 52, no. 5, pp. 782–798, 2007.