Funnel Libraries for Real-Time Robust Feedback Motion Planning

by

Anirudha Majumdar

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2016

Certified by .....
Russ Tedrake
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Funnel Libraries for Real-Time Robust Feedback Motion Planning

by

Anirudha Majumdar

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

We consider the problem of generating motion plans for a robot that are guaranteed to succeed despite uncertainty in the environment, parametric model uncertainty, and disturbances. Furthermore, we consider scenarios where these plans must be generated in real-time, because constraints such as obstacles in the environment may not be known until they are perceived (with a noisy sensor) at runtime. Our approach is to pre-compute a library of "funnels" along different maneuvers of the system that the state is guaranteed to remain within (despite bounded disturbances) when the feedback controller corresponding to the maneuver is executed. The resulting *funnel library* is then used to sequentially compose motion plans at runtime while ensuring the safety of the robot. A major advantage of the work presented here is that by explicitly taking into account the effect of uncertainty, the robot can evaluate motion plans based on how vulnerable they are to disturbances.

We demonstrate and validate our method using extensive hardware experiments on a small fixed-wing airplane avoiding obstacles at high speed ($\sim$12 mph), along with thorough simulation experiments of ground vehicle and quadrotor models navigating through cluttered environments. To our knowledge, the resulting hardware demonstrations on a fixed-wing airplane constitute one of the first examples of provably safe and robust control for robotic systems with complex nonlinear dynamics that need to plan in realtime in environments with complex geometric constraints.

The key computational engine we leverage is *sums-of-squares (SOS) programming*. While SOS programming allows us to apply our approach to systems of relatively high dimensionality (up to approximately 10-15 dimensional state spaces), scaling our approach to higher dimensional systems such as humanoid robots requires a different set of computational tools. In this thesis, we demonstrate how *DSOS and SDSOS programming*, which are recently introduced alternatives to SOS programming, can be employed to achieve this improved scalability and handle control systems with as many as 30-50 state dimensions.

Thesis Supervisor: Russ Tedrake
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I am extremely grateful to my advisor Russ Tedrake for giving me the chance to work at his lab first as a visiting undergraduate student and then as a Ph.D. student. Russ's "internal compass" for choosing problems to work on, deep insights into challenging problems, and quick flashes of intuition have been a constant source of inspiration. His positive outlook and enthusiasm have created the ideal environment in which to thrive and a model towards which to strive. I am very grateful to Russ for providing advice not just on technical matters, but also on career and life in equal measures. It has been an incredible pleasure and honor to work with him.

I would also like to thank my thesis committee: Tomas Lozano-Perez, Emilio Frazzoli, and Claire Tomlin. They have been a fantastic resource over the course of my years as a graduate student and have helped guide the work presented in this thesis with excellent questions, pointers, and advice.

I would like to acknowledge a number of people whose help was crucial in obtaining the results presented in this thesis. In particular, I would like to thank Mike Posa for help formulating the QCQP for shifting funnels presented in Chapter 6.1, and Adam Bry and Charlie Richter for help setting up the USB RC transmitter used in the experiments described in Section 8.

I have been fortunate to work with a number of amazing collaborators. I would like to thank Amir Ali Ahmadi for an extremely fruitful set of collaborations. Amir Ali taught me how to view problems through the computational lens and has been a fantastic source for new ideas. His ability to simplify problems to their essence has been a joy to observe and learn from. The work presented in Chapter 9 of this thesis is a result of joint work with Amir Ali.

It has also been a privilege to collaborate with Andy Barry, Mark Tobenkin, Ram Vasudevan, and Hongkai Dai. Andy's advice on choosing the fixed-wing hardware platform to perform experiments with and his help with the launching mechanism for the airplane were invaluable in obtaining the results presented in Chapter 8. Mark helped write my first ever

sums-of-squares program and answered my incessant questions during my first few weeks as an undergrad researcher at the lab and beyond. Mark's help with implementing the code for DSOS and SDSOS programming was also crucial in obtaining the results presented in Chapter 9. I learned a great deal of new mathematics during my time working with Ram. My conversations with him on measure theory, functional analysis, and algebraic topology were extremely enriching and provided me with a set of technical tools that will no doubt be very valuable going forwards. I am grateful to Hongkai for giving me the chance to explore problems in grasping and manipulation with him. Hongkai's incredible capacity to work long hours have been an inspiration throughout my time as a graduate student and have pushed me to work that extra bit longer myself. I have lost count of the number of times a quick late night conversation with him has helped solve a problem that I was stuck on.

I would also like to thank the other members of the Robot Locomotion Group for inspiration, advice, help, and entertainment during my time here. It has been an honor to work with people who will no doubt lead our field in years to come. Ian Manchester, Zack Jackowski, Michael Levashov, John Roberts, and Alec Shkolnik were immensely helpful during the beginning of my time at the lab. I have also had numerous enlightening conversations with Mike Posa, Frank Permenter, Joe Moore, Scott Kuindersma, Tim Jenks, Robin Deits, Twan Koolen, Pete Florence, John Carter, Andres Valenzuela, Lucas Manuelli, Pat Marion, Greg Izatt, Zico Kolter, Elena Glassman, Jacob Steinhardt, and Kanako Miura. I would like to thank Benoit Landry, Ned Burnell, and Gerardo Bledt for working with me as UROPs. I also want to express my gratitude towards Mieke Moran and Kathy Bates for making my stay at the Robot Locomotion Group much easier in terms of administrative matters, be it making flight reservations for a conference or applying for a visa.

My years as an undergraduate at Penn were crucial in my development as a researcher. I would like to thank Dan Koditschek, Dan Lee, and Vijay Kumar for being advisors and mentors during my time at Penn. My four years at the KodLab prepared me for a research career in ways that no course could have. I will always be grateful to Dan K. for providing me with an atmosphere in which I could learn and thrive, patiently listening to my half-baked

ideas, and gently guiding me through the research path. Dan has been an immensely valuable source of sage advice and guidance over the years. I would also like to thank Hal Komsuoglu, Shai Revzen, Clark Haynes and Kevin Galloway for mentorship during my four years at the KodLab. My experience as a member of Dan Lee's RoboCup team was also invaluable. Dan's mentorship on RoboCup allowed me to work with state-of-the-art humanoid robots and gave me a sense for all the challenges associated with real hardware. Vijay Kumar was officially my senior design advisor, but was a mentor throughout my undergrad years. I will always remember the time he gave a tour of the GRASP lab to a freshman who walked into his office with no appointment on a busy day.

My time at Jamnabai Narsee School in Mumbai played an important role in preparing me for the academic road ahead. In particular, I would like to thank Faisal Azmi for impressing upon me the importance of mathematics at an early age and providing a lot of encouragement along the way.

I would like to thank my parents and sister for unconditional love and support over the years. Without them, none of this would be possible. Thank you for providing me the platform on which to work towards my dreams.

Finally, I would like to thank the MIT community as a whole. I dreamed of studying and working at MIT since I was eleven years old. It is a rare privilege to be able to live one's childhood dreams and rarer still for the reality to surpass the dreams.

Contents

List of Figures

List of Tables

Chapter 1

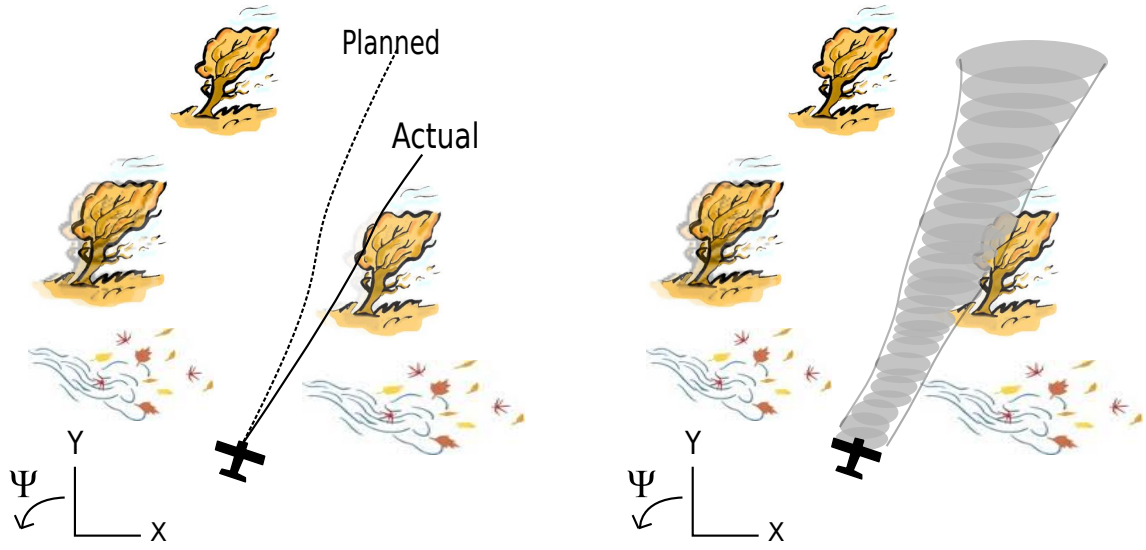
Introduction

Imagine an unmanned aerial vehicle (UAV) flying at high speed through a cluttered environment in the presence of wind gusts, a legged robot traversing rough terrain, or a mobile robot grasping and manipulating previously unlocalized objects in the environment. These applications demand that the robot move through (and in certain cases interact with) its environment with a very high degree of agility while still being in close proximity to obstacles. Such systems today lack guarantees on their safety and can fail dramatically in the face of uncertainty in their environment and dynamics.

The tasks mentioned above are characterized by three main challenges. First, the dynamics of the system are nonlinear, underactuated, and subject to constraints on the input (e.g. torque limits). Second, there is a significant amount of uncertainty in the dynamics of the system due to disturbances and modeling error. Finally, the geometry of the environment that the robot is operating in is unknown until runtime, thus forcing the robot to plan in *real-time*.

1.1 Contributions

In this thesis, we address these challenges by combining approaches from motion planning, feedback control, and tools from Lyapunov theory and convex optimization in order to per-



(a) A plane deviating from its nominal planned trajectory due to a heavy cross-wind.

(b) The "funnel" of possible trajectories.

Figure 1-1: Not accounting for uncertainty while planning motions can lead to disastrous consequences.

form robust real-time motion planning in the face of uncertainty. In particular, in an offline computation stage, we first design a finite library of open loop trajectories. For each trajectory in this library, we use tools from convex optimization (*sums-of-squares programming* in particular) to design a controller that explicitly attempts to minimize the size of the worst case reachable set of the system given a description of the uncertainty in the dynamics and bounded external disturbances. This control design procedure yields an outer approximation of the reachable set, which can be visualized as a "funnel" around the trajectory, that the closed-loop system is guaranteed to remain within. A cartoon of such a funnel is shown in Figure 1-1(b). Finally, we provide a way of *sequentially composing* these robust motion plans at runtime in order to operate in a provably safe manner in previously unseen environments.

One of the most important advantages that our approach affords us is the ability to choose between the motion primitives in our library in a way that takes into account the dynamic effects of uncertainty. Imagine a UAV flying through a forest that has to choose

between two motion primitives: a highly dynamic roll maneuver that avoids the trees in front of the UAV by a large margin or a maneuver that involves flying straight while avoiding the trees only by a small distance. An approach that neglects the effects of disturbances and uncertainty may prefer the former maneuver since it avoids the trees by a large margin and is therefore "safer". However, a more careful consideration of the two maneuvers could lead to a different conclusion: the dynamic roll maneuver is far more susceptible to wind gusts and perturbations to the initial state than the second one. Thus, it may in fact be more robust to execute the second motion primitive. Further, it may be possible that neither maneuver is guaranteed to succeed and it is safer to abort the mission and simply transition to a hover mode. Our approach allows robots to make these critical decisions, which are essential if robots are to move out of labs and operate in real-world environments.

We demonstrate and validate our approach using thorough simulation experiments of ground vehicle and quadrotor models navigating through cluttered environments, along with extensive hardware experiments on a small fixed-wing airplane avoiding obstacles at high speed ($\sim$12 miles per hour) in an indoor motion capture environment. To the best of our knowledge, the resulting hardware demonstrations on a fixed-wing airplane constitute one of the first examples of provably safe and robust control for robotic systems with complex nonlinear dynamics that need to plan in real-time in cluttered environments.

The key computational engine we leverage in this thesis is sums-of-squares (SOS) programming. While SOS programming allows us to apply our approach to systems of relatively high dimensionality (up to approximately 10-15 dimensional state spaces), scaling our approach to higher dimensional systems such as humanoid robots requires a different set of computational tools. In this thesis, we demonstrate how *DSOS and SDSOS programming* [3, 4], which are recently introduced alternatives to SOS programming, can be employed to achieve this improved scalability and handle control systems with as many as 30-50 state dimensions. Problems of this scale are significantly beyond what SOS programming can currently handle.

1.2 Outline

The outline of the thesis is as follows. Chapter 2 discusses prior work; Chapter 3 provides a brief background on semidefinite and sums-of-squares (SOS) programming, which are used heavily throughout the thesis; Chapter 4 shows how to use SOS programming to compute funnels; Chapter 5 introduces the notion of a funnel library; Chapter 6 describes our algorithm for using funnel libraries for real-time robust planning in environments that have not been seen by the robot before; Chapter 7 presents extensive simulation results on a ground vehicle model and compares our approach with an approach based on trajectory libraries; Chapter 7 also considers a quadrotor model and shows how one can use our approach to guarantee collision-free flight in certain environments; Chapter 8 presents hardware experiments on a small fixed-wing airplane in order to demonstrate and validate our approach; Chapter 9 describes how DSOS and SDSOS programming can be used to handle systems in the 30-50 dimensional range; Chapter 10 concludes the thesis with a discussion of challenges and open problems.

Chapter 2

Relevant Work

2.1 Motion Planning

Motion planning has been the subject of significant research in the last few decades and has enjoyed a large degree of success in recent years. Planning algorithms like the Rapidly-exploring Randomized Tree (RRT) [54], RRT* [53], and related trajectory library approaches [60] [37] [99] can handle large state-space dimensions and complex differential constraints. These algorithms have been successfully demonstrated on a wide variety of hardware platforms [57] [93] [92] [90]. However, an important challenge is their inability to explicitly reason about uncertainty and feedback. Modeling errors, state uncertainty and disturbances can lead to failure if the system deviates from the planned nominal trajectories.

The motion planning aspect of our approach draws inspiration from the vast body of work that is focused on computing motion primitives in the form of trajectory libraries. For example, trajectory libraries have been used in diverse applications such as humanoid balance control [60], autonomous ground vehicle navigation [92], grasping [15] [32], and UAV navigation [34] [12]. The Maneuver Automaton [37] attempts to capture the formal properties of trajectory libraries as a hybrid automaton, thus providing a unifying theoretical framework. Maneuver Automata have also been used for real-time motion planning with static and dynamic obstacles [38]. Further theoretical investigations have focused on the

offline generation of diverse but sparse trajectories that ensure the robot's ability to perform the necessary tasks online in an efficient manner [43]. More recently, tools from sub-modular sequence optimization have been leveraged in the optimization of the sequence and content of trajectories evaluated online [32, 31].

Robust motion planning has also been a very active area of research in the robotics community. Early work [22] [63] [48] [58] focused on settings where the dynamics of the system are not dominant and one can treat the system as a kinematic one. The problem is then one of planning paths through configuration space that are robust to uncertainty in the motion of the robot and geometry of the workspace. Our work with funnels takes inspiration from the early work on fine-motion planning, where the notions of *funnels* [68] and *preimage backchaining* [63] (also known as *goal regression* or *sequential composition*) were first introduced. The theme of robust kinematic motion planning has persisted in recent work [72] [44] [67], which deals with uncertainty in the geometry of the environment and obstacles.

2.2 Planning under Uncertainty

In settings where the dynamics of the system must be taken into account (e.g., for underactuated systems), the work on *planning under uncertainty* attempts to address the challenges associated with uncertainty in the dynamics, state, and geometry of the environment. In particular, chance-constrained programming [26] provides an elegant mathematical framework for reasoning about stochastic uncertainty in the dynamics, initial conditions, and obstacle geometry [18] [79] [104] and allows one to generate motion plans with bounds on the probability of collision with obstacles. In settings where the state of the system is partially observable and there is significant uncertainty in the observations, one can extend this framework to plan in the *belief space* of the system (i.e., the space of distributions over states) [107] [108]. While these approaches allow one to explicitly reason about uncertainty in the system, they are typically restricted to handling linear dynamical systems with Gaussian uncertainty. This is due to the computational burden of solving chance constrained

problems for nonlinear and non-Gaussian systems.

Similarly, other approaches for belief space planning [24] [85] [88] [16] [1] [82] also approximate the belief state as a Gaussian distribution over state space (however, see [84] for an exception to this) for computational efficiency and hence the true belief state is not tracked. Thus, in general, one does not have robustness guarantees. The approach we take in this work is to assume that disturbances/uncertainty are *bounded* and provide explicit bounds on the reachable set to facilitate safe operation of the *nonlinear* system.

More generally, the rich literature on Partially Observable Markov Decision Processes (POMDPs) [51] is also relevant here. POMDPs present an elegant mathematical framework for reasoning about uncertainty in state and dynamics. However, we note that our focus in this work is on dynamical systems with continuous state and action spaces, whereas the POMDP literature typically focuses on discretized state/action spaces for the most part.

2.3 Reachable Sets

Reachability analysis for linear and nonlinear control systems has a long history in the controls community. For *linear* systems subject to bounded disturbances, there exist a number of techniques for efficiently computing (approximations of) backwards and forwards reachable sets [56] [42] [109]. One can apply techniques from linear reachability analysis to conservatively approximate reachable sets of nonlinear systems by treating nonlinearities as bounded disturbances [9]. This idea has been used in [8] for performing online safety verification for ground vehicles. A similar idea was used in [7] to perform online safety verification for UAVs and to decide when the UAV should switch to an emergency maneuver (a "loiter circle"). In this thesis we will also compute outer approximations of reachable sets (which we refer to as "funnels"). However, the approach we present here is not based on a linearization of the system and thus has the potential to be less conservative for highly nonlinear systems. Further, the scope of our work extends beyond verification; the emphasis here is on safe real-time *planning* with funnels.

Approximations of reachable sets for nonlinear systems can be computed via Hamilton-

Jacobi-Bellman (HJB) differential game formulations [73]. This method was used in [41] for designing motion primitives for making a quadrotor perform an autonomous backflip in a 2D plane. While this approach handles unsafe sets that the system is not allowed to enter, it is assumed that these sets are specified *a priori*. In this thesis, we are concerned with scenarios in which unsafe sets (such as obstacles) are not specified until runtime and must thus be reasoned about *online*. Further, techniques for computing reachable sets based on the HJB equation have historically suffered from the curse of dimensionality since they rely on discretizations of the state space of the system. Hence, unless the system under consideration has special structure (e.g., decoupled systems [27]), these methods have difficulty scaling up beyond approximately 5-6 dimensional state spaces.

An approach that is closely related to our work is the work presented in [78]. The authors propose a randomized planning algorithm in the spirit of RRTs that explicitly reasons about disturbances and uncertainty. Specifications of input to output stability with respect to disturbances provide a parameterization of "tubes" (analogous to our "funnels") that can be composed together to generate motion plans that are collision-free. The factors that distinguish the approach we present in this thesis from the one proposed in [78] are our focus on the *real-time* aspect of the problem and use of sums-of-squares programming as a way of computing reachable sets. In [78], the focus is on generating safe motion plans when the obstacle positions are known *a priori*. Further, we provide a general technique for computing and explicitly minimizing the size of tubes.

Another approach that is closely related to ours is Model Predictive Control with Tubes [69]. The idea is to solve the optimal control problem online with guaranteed "tubes" that the trajectories stay in. A closely related idea is that of "flow-tubes", which have been used for high-level planning for autonomous systems [59]. However, these methods are generally limited to linear systems.

2.4 Lyapunov Theory and SOS programming

A critical component of the work presented here is the computation of "funnels" for nonlinear systems via Lyapunov functions. The metaphor for thinking about Lyapunov functions as defining funnels was introduced to the robotics community in [25], where funnels were *sequentially composed* in order to produce dynamic behaviors in a robot. However, computational tools for automatically searching for Lyapunov functions were lacking until very recently. In recent years, sums-of-squares (SOS) programming has emerged as a way of checking the Lyapunov function conditions associated with each funnel [80]. The technique relies on the ability to check nonnegativity of multivariate polynomials by expressing them as a sum of squares of polynomials. This can be written as a semidefinite optimization program and is amenable to efficient computational algorithms such as interior point methods [80]. Assuming polynomial dynamics, one can check that a polynomial Lyapunov candidate, $V(x)$, satisfies $V(x) > 0$ and $\dot{V}(x) < 0$ in some region B_r. Importantly, the same idea can be used for computing funnels along time-indexed trajectories of a system [102] [103]. In this thesis, we will use a similar approach to synthesize feedback controllers that explicitly seek to minimize the effect of disturbances on the system by minimizing the size of the funnel computed along a trajectory. Thus, we are guaranteed that if the system starts off in the set of given initial conditions, it will remain in the computed "funnel" even if the model of the dynamics is uncertain and the system is subjected to bounded disturbances.

The ability to compute funnels using SOS programming was leveraged by the LQR-Trees algorithm [102] for *feedback motion planning* for nonlinear systems. The algorithm works by creating a tree of locally stabilizing controllers which can take any initial condition in some bounded region in state space to the desired goal. However, LQR-Trees lack the ability to handle scenarios in which the task and environment are unknown till runtime: the offline precomputation of the tree does not take into account potential runtime constraints like obstacles, and an online implementation of the algorithm is computationally infeasible.

The SOS programming approach has also been used to guarantee obstacle avoidance conditions for nonlinear systems by computing *barrier certificates* [86] [13]. Barrier functions

are similar to Lyapunov functions in spirit, but are used to guarantee that trajectories starting in some given set of initial conditions will never enter an "unsafe" region containing obstacles. This approach, however, is limited to settings where the locations and geometry of obstacles are known beforehand since the barrier certificate one computes depends on this data and computing barrier certificates in real-time using SOS programming is not computationally feasible at the present time.

Chapter 3

Background

In this chapter we provide a brief background on the key computational tools that will be employed throughout this thesis.

3.1 Semidefinite Programming (SDP)

Semidefinite programs (SDPs) form an important class of convex optimization problems. They are optimization problems over the space of symmetric positive semidefinite (psd) matrices. Recall that a $n \times n$ symmetric matrix Q is positive semidefinite if $x^T Q x \geq 0,\ \forall x \in \mathbb{R}^n$. Denoting the set of $n \times n$ symmetric matrices as $\mathbf{S}^n$, a SDP in standard form is written as:

$$\begin{aligned} \min_{X \in \mathbf{S}^n} \quad & \langle C, X \rangle \\ \text{s.t.} \quad & \langle A_i, X \rangle = b_i \quad \forall i \in \{1, \ldots, m\}, \\ & X \succeq 0, \end{aligned} \tag{3.1}$$

where $C, A_i \in \mathbf{S}^n$ and $\langle X, Y \rangle := \mathrm{Tr}(X^T Y) = \sum_{i,j} X_{ij} Y_{ij}$. In other words, a SDP involves minimizing a cost function that is linear in the elements of the decision matrix X subject to

linear and positive semidefiniteness constraints on X.

Semidefinite programming includes Linear Programming (LP), Quadratic Programming (QP) and Second-Order Cone Programming (SOCP) as special cases. As in these other cases, SDPs are amenable to efficient numerical solution via interior point methods. The interested reader may wish to consult [106] and [19, Chapter 2] for a more thorough introduction to SDPs.

3.2 Sums-of-Squares (SOS) Programming

An important application of SDPs is to check nonnegativity of polynomials. The decision problem associated with checking polynomial nonnegativity is NP–hard in general [80]. However, the problem of determining whether a polynomial is a sum-of-squares (SOS), which is a sufficient condition for nonnegativity, is amenable to efficient computation. A polynomial p in indeterminates[1] $x_1, x_2, \ldots, x_n$ is SOS if it can be written as $p(x) = \sum_{i=1}^{m} q_i^2(x)$ for a set of polynomials $\{q_i\}_{i=1}^m$. This condition is *equivalent* to the existence of a positive semidefinite (psd) matrix Q that satisfies:

$$p(x) = v(x)^T Q v(x), \ \forall x \in \mathbb{R}^n, \tag{3.2}$$

where $v(x)$ is the vector of all monomials with degree less than or equal to half the degree of p [80]. Note that the equality constraint (3.2) imposes linear constraints on the elements of the matrix Q that come from matching coefficients of the polynomials on the left and right hand sides. Thus, semidefinite programming can be used to certify that a polynomial is a sum of squares. Indeed, by allowing the coefficients of the polynomial p to be decision variables, we can solve optimization problems over the space of SOS polynomials of some fixed degree. Such optimization problems are referred to as sums-of-squares (SOS) programs. The interested reader is referred to [80] and [19, Chapters 3,4] for a more thorough introduction

[1]Throughout this thesis, the variables x that a polynomial $p(x)$ depend on will be referred to as "indeterminates". This is to distinguish these variables from *decision variables* in our optimization problems, which will typically be the coefficients of the polynomial.

to SOS programming.

In addition to being able to prove global nonnegativity of polynomials, the SOS programming approach can also be used to demonstrate nonnegativity of polynomials on basic semialgebraic sets (i.e., sets described by a finite number of polynomial inequalities and equalities). Suppose we are given a set $\mathcal{B}$:

$$\mathcal{B} = \{x \in \mathbb{R}^n \mid g_{eq,i}(x) = 0,\ g_{ineq,j}(x) \geq 0\}, \tag{3.3}$$

where $g_{eq,i}$ and $g_{ineq,j}$ are polynomials for $i \in \{1, ..., N_{eq}\}, j \in \{1, ..., N_{ineq}\}$. We are interested in showing that a polynomial p is nonnegative on the set $\mathcal{B}$:

$$x \in \mathcal{B} \implies p(x) \geq 0. \tag{3.4}$$

We can write the following SOS constraints in order to impose (3.4):

$$q(x) := p(x) - \overbrace{\sum_{i=1}^{N_{eq}} L_{eq,i}(x) g_{eq,i}(x) - \sum_{j=1}^{N_{ineq}} L_{ineq,j}(x) g_{ineq,j}(x)}^{r(x)} \text{ is SOS}, \tag{3.5}$$

$$L_{ineq,j}(x) \text{ is SOS}, \forall j \in \{0, ..., N_{ineq}\}. \tag{3.6}$$

Here, the polynomials $L_{eq,i}$ and $L_{ineq,j}$ are "multiplier" polynomials analogous to Lagrange multipliers in constrained optimization. In order to see that (3.5) and (3.6) are sufficient conditions for (3.4), note that when a point x satisfies $g_{eq,i}(x) = 0$ and $g_{ineq,j}(x) \geq 0$ for $i \in \{1, ..., N_{eq}\}, j \in \{1, ..., N_{ineq}\}$ (i.e., when $x \in \mathcal{B}$) then the term $r(x)$ is non-positive. Hence, for $q(x)$ to be nonnegative (which must be the case since q is SOS), $p(x)$ must be nonnegative. Thus, we have the desired implication in (3.4). This process for using multipliers to impose nonnegativity constraints on sets is known as the generalized S-procedure [80] and will be used extensively in Chapter 4 for computing funnels.

Chapter 4

Computing Funnels

In this chapter we describe how the tools from Chapter 3 can be used to compute outer approximations of reachable sets ("funnels") around trajectories of a nonlinear system. The approach in Chapter 4.1 builds on the work presented in [103, 102] while Chapters 4.3.1 and 4.3.2 are based on [66] and [65] respectively. In contrast to this prior work however, we consider the problem of computing outer approximations of forwards reachable sets as opposed to inner approximations of backwards reachable sets. This leads to a few subtle differences in the cost functions of our optimization problems.

Consider the following dynamical system:

$$\dot{x} = f(x(t), u(t)), \tag{4.1}$$

where $x(t) \in \mathbb{R}^n$ is the state of the system at time t and $u(t) \in \mathbb{R}^m$ is the control input. Let $x_0 : [0, T] \to \mathbb{R}^n$ be the nominal trajectory that we would like the system to follow and $u_0 : [0, T] \to \mathbb{R}^m$ be the nominal open-loop control input. Defining new coordinates $\bar{x} = x - x_0(t)$ and $\bar{u} = u - u_0(t)$, we can rewrite the dynamics (4.1) in these variables as:

$$\dot{\bar{x}} = \dot{x} - \dot{x}_0 = f(x_0(t) + \bar{x}(t), u_0(t) + \bar{u}(t)) - \dot{x}_0. \tag{4.2}$$

We will first consider the problem of computing funnels for a closed-loop system subject

to no uncertainty. To this end, we assume that we are given a feedback controller $\bar{u}_f(t, \bar{x})$ that corrects for deviations around the nominal trajectory (we will consider the problem of designing feedback controllers later in this chapter). We can then write the closed-loop dynamics of the system as:

$$\dot{\bar{x}} = f_{cl}(t, \bar{x}(t)). \tag{4.3}$$

Given a set of initial conditions $\mathcal{X}_0 \subset \mathbb{R}^n$ with $x_0(0) \in \mathcal{X}_0$, our goal is to find a tight outer approximation of the set of states the system may evolve to at time $t \in [0, T]$. In particular, we are concerned with finding sets $F(t) \subset \mathbb{R}^n$ such that:

$$\bar{x}(0) \in \mathcal{X}_0 \implies \bar{x}(t) \in F(t), \ \forall t \in [0, T]. \tag{4.4}$$

Definition 1. *A* funnel *associated with a closed-loop dynamical system $\dot{\bar{x}} = f_{cl}(t, \bar{x}(t))$ is a map $F : [0, T] \to \mathcal{P}(\mathbb{R}^n)$ from the time-interval $[0, T]$ to the power set (i.e., the set of subsets) of $\mathbb{R}^n$ such that the sets $F(t)$ satisfy the condition* (4.4) *above.*

Thus, with each time $t \in [0, T]$, the funnel associates a set $F(t) \subset \mathbb{R}^n$. We will parameterize the sets $F(t)$ as sub-level sets of nonnegative time-varying functions $V : [0, T] \times \mathbb{R}^n \to \mathbb{R}^+$:

$$F(t) = \{\bar{x}(t) \in \mathbb{R}^n | V(t, \bar{x}(t)) \leq \rho(t)\}. \tag{4.5}$$

Letting $\mathcal{X}_0 \subset F(0, \bar{x})$, the following constraint is a sufficient condition for (4.4):

$$V(t, \bar{x}) = \rho(t) \implies \dot{V}(t, \bar{x}) < \dot{\rho}(t), \ \forall t \in [0, T]. \tag{4.6}$$

Here, $\dot{V}$ is computed as:

$$\dot{V}(t, \bar{x}) = \frac{\partial V(t, \bar{x})}{\partial \bar{x}} f_{cl}(t, \bar{x}) + \frac{\partial V(t, \bar{x})}{\partial t}. \tag{4.7}$$

Intuitively, the constraint (4.6) says that on the boundary of the funnel (i.e., when $V(t, \bar{x}) =$

$\rho(t)$), the function V grows slower than ρ. Hence, states on the boundary of the funnel remain within the funnel. This intuition is formalized in [102, 103].

While *any* function that satisfies (4.6) provides us with a valid funnel, we are interested in finding *tight* outer approximations of the reachable set. A natural cost function for encouraging tightness is the volume of the sets $F(t)$. Combining this cost function with our constraints, we obtain the following optimization problem:

$$\begin{aligned} \inf_{V,\rho} \quad & \int_0^T vol(F(t))\ \mathrm{dt} \\ \text{s.t.} \quad & V(t,\bar{x}) = \rho(t) \implies \dot{V}(t,\bar{x}) < \dot{\rho}(t),\ \forall t \in [0,T], \\ & \mathcal{X}_0 \subset F(0,\bar{x}). \end{aligned} \tag{4.8}$$

4.1 Numerical implementation using SOS programming

Since the optimization problem (4.8) involves searching over spaces of functions, it is infinite dimensional and hence not directly amenable to numerical computation. However, we can use the SOS programming approach described in Chapter 3 to obtain finite dimensional optimization problems in the form of semidefinite programs (SDPs). We first concentrate on implementing the constraints in (4.8). We will assume that the initial condition set $\mathcal{X}_0$ is a semi-algebraic set (i.e., described in terms of polynomial inequalities):

$$\mathcal{X}_0 = \{\bar{x} \in \mathbb{R}^n \mid g_{0,i}(\bar{x}) \geq 0,\ \forall i = 1, \ldots, N_0\}. \tag{4.9}$$

Then the constraints in (4.8) can be written as:

$$V(t,\bar{x}) = \rho(t) \implies \rho(t) - \dot{V}(t,\bar{x}) > 0 \tag{4.10}$$

$$g_{0,i}(\bar{x}) \geq 0\ \forall i \in \{1, \ldots, N_0\} \implies \rho(0) - V(0,\bar{x}) \geq 0. \tag{4.11}$$

If we restrict ourselves to polynomial dynamics and polynomial functions V and ρ, these constraints are precisely in the form of (3.4) in Chapter 3.2. We can thus apply the procedure described in Chapter 3.2 and arrive at the following sufficient conditions for (4.10) and (4.11):

$$\dot{\rho}(t) - \dot{V}(t,\bar{x}) - L(t,\bar{x})[V(t,\bar{x}) - \rho(t)] - L_t(t,\bar{x})[t(T-t)] \text{ is SOS}, \tag{4.12}$$

$$\rho(0) - V(0,\bar{x}) - \sum_{i}^{N_0} L_{0,i}(\bar{x}) g_{0,i}(\bar{x}) \text{ is SOS}, \tag{4.13}$$

$$L_t(t,\bar{x}), L_{0,i}(\bar{x}) \text{ are SOS}, \forall i \in \{1,\ldots,N_0\}. \tag{4.14}$$

As in Chapter 3.2, the polynomials L, L_t and $L_{0,i}$ are "multiplier" polynomials whose coefficients are decision variables.

Next, we focus on approximating the cost function in (4.8) using semidefinite programming. This can be achieved by sampling in time and replacing the integral with the finite sum $\sum_{k=1}^{N} vol(F(t_k))$. In the special case where the function V is quadratic in $\bar{x}$:

$$V(t_k,\bar{x}) = \bar{x}^T S_k \bar{x}, \;\; S_k \succeq 0, \tag{4.15}$$

the set $F(t_k)$ is an ellipsoid and we can use semidefinite programming (SDP) to directly minimize the volume by maximizing the determinant of S_k (recall that the volume of the ellipsoid $F(t_k)$ is a monotonically decreasing function of the determinant of S_k). Note that while the problem of maximizing the determinant of a psd matrix is not directly a problem of the form (3.1), it can be transformed into such a form [14, Chapter3]. Further note that the fact that our cost function can be handled directly in the SDP framework is in distinction to the approaches for computing inner approximations of backwards reachable sets [102] [103] [65]. This is because the determinant of a psd matrix is a *concave* function and hence *minimizing* the determinant is not a convex problem. Hence, in the previous work, the authors used heuristics for maximizing volume.

In the more general case, we can minimize an upper bound on the cost function

$\sum_{k=1}^{N} vol(F(t_k))$ by introducing ellipsoids $\mathcal{E}(t_k)$:

$$\mathcal{E}(t_k) = \{\bar{x} \in \mathbb{R}^n | \bar{x}^T S_k \bar{x} \leq 1,\ S_k \succeq 0\} \tag{4.16}$$

such that $F(t_k) \subset \mathcal{E}(t_k)$ and minimizing $\sum_{k=1}^{N} vol(\mathcal{E}(t_k))$. The containment constraint can be equivalently expressed as the constraint:

$$V(t_k, \bar{x}) \leq \rho(t_k) \implies \bar{x}^T S_k \bar{x} \leq 1, \tag{4.17}$$

and can thus be imposed using SOS constraints:

$$1 - \bar{x}^T S_k \bar{x} - L_{\mathcal{E},k}(\bar{x})[\rho(t_k) - V(t_k, \bar{x})] \text{ is SOS}, \tag{4.18}$$

$$L_{\mathcal{E},k}(\bar{x}) \text{ is SOS}. \tag{4.19}$$

Combining our cost function with the constraints (4.12) - (4.14), we obtain the following optimization problem:

$$\inf_{V,\rho,L,L_t,L_{0,i},S_k,L_{\mathcal{E},k}} \quad \sum_{k=1}^{N} vol(\mathcal{E}(t_k)) = \sum_{k=1}^{N} vol(\{\bar{x} | \bar{x}^T S_k \bar{x} \leq 1\}) \tag{4.20}$$

$$\text{s.t.} \quad \dot{\rho}(t) - \dot{V}(t, \bar{x}) - L(t, \bar{x})[V(t, \bar{x}) - \rho(t)] - L_t(t, \bar{x})[t(T - t)] \text{ is SOS}, \tag{4.21}$$

$$\rho(0) - V(0, \bar{x}) - \sum_{i}^{N_0} L_{0,i}(\bar{x}) g_{0,i}(\bar{x}) \text{ is SOS},$$

$$1 - \bar{x}^T S_k \bar{x} - L_{\mathcal{E},k}(\bar{x})[\rho(t_k) - V(t_k, \bar{x})] \text{ is SOS}, \qquad \forall k \in \{1, \ldots, N\},$$

$$S_k \succeq 0, \qquad \forall k \in \{1, \ldots, N\},$$

$$L_t(t, \bar{x}), L_{0,i}(\bar{x}), L_{\mathcal{E},k}(\bar{x}) \text{ are SOS}, \forall i \in \{1, \ldots, N_0\},\ \forall k \in \{1, \ldots, N\}.$$

While this optimization problem is finite dimensional, it is non-convex in general since the first constraints are *bilinear* in the decision variables (e.g., the coefficients of the polynomials

L and V are multiplied together in the first constraint). To apply SOS programming, we require the constraints to be linear in the coefficients of the polynomials we are optimizing. However, note that when V and ρ are fixed, the constraints are linear in the other decision variables. Similarly, when the multipliers L and $L_{\mathcal{E},k}$ are fixed, the constraints are linear in the remaining decision variables. Thus, we can efficiently perform this optimization by alternating between the two sets of decision variables $(L, L_t, L_{0,i}, S_k, L_{\mathcal{E},k})$ and $(V, \rho, L_t, L_{0,i}, S_k)$. In each step of the alternation, we can optimize our cost function $\sum_{k=1}^{N} vol(\mathcal{E}(t_k))$. These alternations are summarized in Algorithm 1. Note that the algorithm requires an initialization for V and ρ. We will discuss how to obtain these in Chapter 4.4.

1: Initialize V and ρ.
2: $cost_{prev} = \infty$;
3: converged = false;
4: **while** ¬converged **do**
5: **STEP 1** : Minimize $\sum_{k=1}^{N} vol(\mathcal{E}(t_k))$ by searching for multiplier polynomials $(L, L_t, L_{0,i}, L_{\mathcal{E},k})$ and S_k while fixing V and ρ.
6: **STEP 2** : Minimize $\sum_{k=1}^{N} vol(\mathcal{E}(t_k))$ by searching for $(V, \rho, L_t, L_{0,i}, S_k)$ while fixing L and $L_{\mathcal{E},k}$.
7: $cost = \sum_{k=1}^{N} vol(\mathcal{E}(t_k))$;
8: **if** $\frac{cost_{prev} - cost}{cost_{prev}} < \epsilon$ **then**
9: converged = true;
10: **end if**
11: $cost_{prev} = cost$;
12: **end while**

Algorithm 1: Funnel Computation

Remark 1. *It is easy to see that Algorithm 1 converges (though not necessarily to an optimal solution). Each iteration of the alternations is guaranteed to achieve a cost function that is at least as good as the previous iteration (since the solution from the previous iteration is a valid one). Hence, the sequence of optimal values in each iteration form a monotonically*

non-increasing sequence. Combined with the fact that the cost function is bounded below by 0, we conclude that this sequence converges and hence that Algorithm 1 terminates.

4.2 Approximation via time-sampling

As observed in [103] in practice it is often the case that the nominal trajectory $x_0 : [0, T] \to \mathbb{R}^n$ is difficult to approximate with a low degree polynomial in time. This can lead to the constraint (4.21) in the problem (4.20) having a high degree polynomial dependence on t. Thus it is often useful to implement an approximation of the optimization problem (4.20) where the condition (4.10) is checked only at a finite number of sample points $t_k \in [0, T],\ k \in \{1, \dots, N\}$. We can use a piecewise linear parameterization of ρ and can thus compute:

$$\dot{\rho}(t_k) = \frac{\rho(t_{k+1}) - \rho(t_k)}{t_{k+1} - t_k}. \tag{4.22}$$

Similarly we can parameterize the function V by polynomials $V_k(\bar{x})$ at each time sample and compute:

$$\frac{\partial V(t, \bar{x})}{\partial t} \approx \frac{V_{k+1}(\bar{x}) - V_k(\bar{x})}{t_{k+1} - t_k}. \tag{4.23}$$

We can then write the following modified version of the problem (4.20):

$$\begin{aligned}
\inf_{V_k, \rho, L_k, L_{0,i}, S_k, L_{\mathcal{E},k}} \quad & \sum_{k=1}^{N} vol(\mathcal{E}(t_k)) = \sum_{k=1}^{N} vol(\{\bar{x} | \bar{x}^T S_k \bar{x} \leq 1\}) && && (4.24)\\
\text{s.t.} \quad & \dot{\rho}(t_k) - \dot{V}_k(\bar{x}) - L_k(\bar{x})[V_k(\bar{x}) - \rho(t_k)], && \forall k \in \{1, \dots, N\},\\
& \rho(t_1) - V_1(\bar{x}) - \sum_{i}^{N_0} L_{0,i}(\bar{x}) g_{0,i}(\bar{x}) \text{ is SOS}, && \\
& 1 - \bar{x}^T S_k \bar{x} - L_{\mathcal{E},k}(\bar{x})[\rho(t_k) - V_k(\bar{x})] \text{ is SOS}, && \forall k \in \{1, \dots, N\},\\
& S_k \succeq 0, && \forall k \in \{1, \dots, N\},\\
& L_{0,i}(\bar{x}), L_{\mathcal{E},k}(\bar{x}) \text{ are SOS}, && \forall i \in \{1, \dots, N_0\},\ \forall k \in \{1, \dots, N\}.
\end{aligned}$$

This program does not have any algebraic dependence on the variable t and can thus provide significant computational gains over (4.20). However, it does not provide an exact funnel certificate. One would hope that with a sufficiently fine sampling in time, one would recover exactness. Partial results in this direction are provided in [103] along with numerical examples showing that the loss of accuracy from the sampling approximation can be quite small in practice.

The problem (4.24) is again bilinear in the decision variables. It is linear in the two sets of decision variables $(L_k, L_{0,i}, S_k, L_{\mathcal{E},k})$ and $(V_k, \rho, L_{0,i}, S_k)$. Thus, Algorithm 1 can be applied directly to (4.24) with the minor modification that V and ρ are replaced by their time-sampled counterparts and the multipliers (L, L_t) are replaced by the multipliers L_k.

4.3 Extensions to the basic algorithm

Next we describe several extensions to the basic framework for computing funnels described in Chapter 4.1. Chapter 4.3.1 discusses the scenario in which the dynamics of the system are subject to bounded disturbances/uncertainty, Chapter 4.3.2 considers the problem of synthesizing feedback controllers that explicitly attempt to minimize the size of the funnel, Chapter 4.3.3 demonstrates how to handle input saturations, and Chapter 4.3.4 considers a generalization of the cost function.

4.3.1 Uncertainty in the dynamics

Suppose that the dynamics of the system are subject to an uncertainty term $w(t) \in \mathbb{R}^d$ that models external disturbances or parametric model uncertainties. The closed-loop dynamics (4.3) can then be modified to capture this uncertainty:

$$\dot{\bar{x}} = f_{cl}(t, \bar{x}(t), w(t)). \tag{4.25}$$

We will assume that the dynamics f_{cl} depend polynomially on w. Given an initial condition set $\mathcal{X}_0 \subset \mathbb{R}^n$ as before, our goal is to find sets $F(t)$ such that $x(t)$ is guaranteed to be in $F(t)$

for any valid disturbance profile:

$$\bar{x}(0) \in \mathcal{X}_0 \implies \bar{x}(t) \in F(t), \forall t \in [0,T], \forall w : [0,T] \to \mathcal{W}. \tag{4.26}$$

Parameterizing the sets $F(t)$ as sub-level sets of nonnegative time-varying functions $V : [0,T] \times \mathbb{R}^n \to \mathbb{R}^+$ as before, the following condition is sufficient to ensure (4.26):

$$V(t,\bar{x}) = \rho(t) \implies \dot{V}(t,\bar{x},w) < \dot{\rho}(t), \forall t \in [0,T], \forall w(t) \in \mathcal{W}, \tag{4.27}$$

where $\dot{V}$ is computed as:

$$\dot{V}(t,\bar{x},w) = \frac{\partial V(t,\bar{x})}{\partial \bar{x}} f_{cl}(t,\bar{x},w) + \frac{\partial V(t,\bar{x})}{\partial t}. \tag{4.28}$$

This is almost identical to the condition (4.6), with the exception that the function V is required to decrease on the boundary of the funnel for every choice of disturbance. Assuming that the set $\mathcal{W}$ is a semi-algebraic set $\mathcal{W} = \{w \in \mathbb{R}^d \mid g_{w,j}(w) \geq 0, \forall j = 1,\ldots,N_w\}$, the optimization problem (4.20) is then easily modified by replacing condition (4.21) with the following constraints:

$$\begin{aligned} \dot{\rho}(t) - \dot{V}(t,\bar{x},w) - L(t,\bar{x},w)[V(t,\bar{x}) - \rho(t)] - L_t(t,\bar{x},w)[t(T-t)] \ldots \\ \ldots - \sum_{j=1}^{N_w} L_{w,j}(t,\bar{x},w) g_{w,j}(w) \text{ is SOS}, \\ L_{w,j}(t,\bar{x},w) \text{ is SOS}, \forall j = \{1,\ldots,N_w\}. \end{aligned} \tag{4.29}$$

These SOS constraints now involve polynomials in the indeterminates $t, \bar{x}$ and w. Since these constraints are linear in the coefficients of the newly introduced multipliers $L_{w,j}$, Algorithm 1 can be directly applied to the modified optimization problem by adding $L_{w,j}$ to the list of polynomials to be searched for in both Step 1 and Step 2 of the iterations. Similarly, the time-sampled approximation described in Chapter 4.2 can also be applied to (4.29).

4.3.2 Feedback control synthesis

So far we have assumed that we have been provided with a feedback controller that corrects for deviations around the nominal trajectory. We now consider the problem of *optimizing* the feedback controller in order to minimize the size of the funnel. We will assume that the system is control affine:

$$\dot{x} = f(x(t)) + g(x(t))u(t), \tag{4.30}$$

and parameterize the control policy as a polynomial $\bar{u}_f(t, \bar{x})$. We can thus write the dynamics in the $\bar{x}$ coordinates as:

$$\dot{\bar{x}} = f(x_0(t) + \bar{x}(t)) + g(x(t))[u_0(t) + \bar{u}_f(t, \bar{x})] - \dot{x}_0. \tag{4.31}$$

The feedback controller can then be optimized by adding the coefficients of the polynomial $\bar{u}_f(t, \bar{x})$ to the set of decision variables in the optimization problem (4.20) while keeping all the constraints unchanged. Note that $\bar{u}_f$ appears in the constraints only through $\dot{V}$, which is now bilinear in the coefficients of V and $\bar{u}_f$ since:

$$\dot{V}(t, \bar{x}) = \frac{\partial V(t, \bar{x})}{\partial \bar{x}} \dot{\bar{x}} + \frac{\partial V(t, \bar{x})}{\partial t}. \tag{4.32}$$

With the (coefficients of) the feedback controller $\bar{u}_f$ as part of the optimization problem, note that the constraints of the problem (4.20) are now bilinear in the two sets of decision variables $(L, L_t, L_{0,i}, S_k, L_{\mathcal{E},k}, \bar{u}_f)$ and $(V, \rho, L_t, L_{0,i}, S_k)$. Thus, in principle we could use a bilinear alternation scheme similar to the one in Algorithm 1 and alternatively optimize these two sets of decision variables. However, in this case we would not be searching for a controller that explicitly seeks to minimize the size of the funnel (since the controller would not be searched for at the same time as V or ρ, which define the funnel). To get around this issue, we add another step in each iteration where we optimize our cost function $\sum_{k=1}^{N} vol(\mathcal{E}(t_k))$ by searching for $(\bar{u}_f, \rho, L_t, L_{0,i}, S_k)$ while keeping $(V, L, L_{\mathcal{E},k})$ fixed. This allows us to search for $\bar{u}_f$ and ρ at the same time, which can significantly improve the quality of

the controllers and funnels we obtain. These steps are summarized in Algorithm 2. By a reasoning identical to the one in Remark 1 it is easy to see that the sequence of optimal values produced by Algorithm 2 converges.

1: Initialize V and ρ.
2: $cost_{prev} = \infty$;
3: converged = false;
4: **while** ¬converged **do**
5: **STEP 1** : Minimize $\sum_{k=1}^{N} vol(\mathcal{E}(t_k))$ by searching for controller $\bar{u}_f$ and $(L, L_t, L_{0,i}, L_{\mathcal{E},k}, S_k)$ while fixing V and ρ.
6: **STEP 2** : Minimize $\sum_{k=1}^{N} vol(\mathcal{E}(t_k))$ by searching for controller $\bar{u}_f$ and $(\rho, L_t, L_{0,i}, S_k)$ while fixing $(V, L, L_{\mathcal{E},k})$.
7: **STEP 3** : Minimize $\sum_{k=1}^{N} vol(\mathcal{E}(t_k))$ by searching for $(V, \rho, L_t, L_{0,i}, S_k)$ while fixing $(L, L_{\mathcal{E},k}, \bar{u}_f)$.
8: $cost = \sum_{k=1}^{N} vol(\mathcal{E}(t_k))$;
9: **if** $\frac{cost_{prev} - cost}{cost_{prev}} < \epsilon$ **then**
10: converged = true;
11: **end if**
12: $cost_{prev} = cost$;
13: **end while**

Algorithm 2: Feedback Control Synthesis

We note that the approach for taking into account uncertainty described in Chapter 4.3.1 can easily be incorporated into Algorithm 2. Similarly, by parameterizing the controller $\bar{u}_f$ as polynomials $u_{f,k}(\bar{x})$ at the times t_k, we can also apply the time-sampled approximation described in Chapter 4.2.

4.3.3 Actuator saturations

A. Approach 1

Our approach also allows us to incorporate actuator limits into the verification procedure. Although we examine the single-input case in this section, this framework is easily extended to handle multiple inputs. Let the control input $u(t)$ at time t be mapped through the following control saturation function:

$$s(u(t)) = \begin{cases} u_{max} & \text{if } u(t) \geq u_{max} \\ u_{min} & \text{if } u(t) \leq u_{min} \\ u(t) & \text{o.w.} \end{cases}$$

Then, in a manner similar to [102], a piecewise analysis of $\dot{V}(t, \bar{x})$ can be used to check the Lyapunov conditions are satisfied even when the control input saturates. Defining:

$$\dot{V}_{min}(t, \bar{x}) := \frac{\partial V(t, \bar{x})}{\partial \bar{x}} \Big(f(x_0 + \bar{x}) + g(x_0 + \bar{x}) u_{min} \Big) + \frac{\partial V(t, \bar{x})}{\partial t}, \tag{4.33}$$

$$\dot{V}_{max}(t, \bar{x}) := \frac{\partial V(t, \bar{x})}{\partial \bar{x}} \Big(f(x_0 + \bar{x}) + g(x_0 + \bar{x}) u_{max} \Big) + \frac{\partial V(t, \bar{x})}{\partial t}, \tag{4.34}$$

we must check the following conditions:

$$u_0(t) + \bar{u}_f(t, \bar{x}) \leq u_{min},\ V(t, \bar{x}) = \rho(t) \implies \dot{V}_{min}(t, \bar{x}) < \dot{\rho}(t), \tag{4.35}$$

$$u_0(t) + \bar{u}_f(t, \bar{x}) \geq u_{max},\ V(t, \bar{x}) = \rho(t) \implies \dot{V}_{min}(t, \bar{x}) < \dot{\rho}(t), \tag{4.36}$$

$$u_{min} \leq u_0(t) + \bar{u}_f(t, \bar{x}) \leq u_{max},\ V(t, \bar{x}) = \rho(t) \implies \dot{V}(t, \bar{x}) < \dot{\rho}(t), \tag{4.37}$$

where u_0 is the open-loop control input and $\bar{u}_f$ is the feedback controller as before. These conditions can be enforced by adding additional multipliers to the optimization program (4.20) or its time-sampled counterpart (4.24).

B. Approach 2

Although one can handle multiple inputs via the above method, the number of SOS conditions grows exponentially with the number of inputs (3^m conditions for $\dot{V}$ are needed in general to handle all possible combinations of input saturations). Thus, for systems with a large number of inputs, an alternative formulation was proposed in [65] that avoids this exponential growth in the size of the SOS program at the cost of adding conservativeness to the size of the funnel. Given limits on the control vector $u \in \mathbb{R}^m$ of the form $u_{min} < u < u_{max}$, we can ask to satisfy:

$$\bar{x} \in F(t) \implies u_{min} < u_0(t) + \bar{u}_f(t, \bar{x}) < u_{max}, \ \forall t \in [0, T]. \tag{4.38}$$

This constraint implies that the applied control input remains within the specified bounds inside the verified funnel (a conservative condition). The number of extra constraints grows linearly with the number of inputs (since we have one new condition for every input), thus leading to smaller optimization problems.

4.3.4 A more general cost function

We end our discussion of extensions to the basic algorithm for computing funnels presented in Chapter 4.1 by considering a generalization of the cost function (volume of the funnel) we have used so far. In particular, it is sometimes useful to minimize the volume of the funnel *projected* onto a subspace of the state space. Suppose this projection map is given by $\pi : \mathbb{R}^n \to \mathbb{R}^{n_p}$ with a corresponding $n_p \times n$ projection matrix P. For an ellipsoid $\mathcal{E} = \{\bar{x} \in \mathbb{R}^n \mid \bar{x}^T S_k \bar{x} \leq 1\}$, the projected set $\pi(\mathcal{E})$ is also an ellipsoid $\mathcal{E}_p = \{\bar{x} \in \mathbb{R}^{n_p} \mid \bar{x}^T S_k^{(p)} \bar{x} \leq 1\}$ with:

$$S_k^{(p)} = (P S_k^{-1} P^T)^{-1}. \tag{4.39}$$

Recall that the ability to minimize the volume of the ellipsoid $\mathcal{E}$ using SDP relied on being able to maximize the determinant of S_k. In order to minimize the volume of $\mathcal{E}_p$, we would

have to maximize $\det(S_k^{(p)})$, which is a complicated (i.e. nonlinear) function of S_k. Hence, in each iteration of Algorithm 1 we linearize the function $\det(S_k^{(p)})$ with respect to S_k at the solution of S_k from the previous iteration and maximize this linearization instead. The linearization of $\det(S_k^{(p)})$ with respect to S_k at a nominal value $S_{k,0}$ can be explicitly computed as:

$$\mathrm{Tr}\Big(P^T(PS_{k,0}^{-1}P^T)^{-1}PS_{k,0}^{-1}S_kS_{k,0}^{-1}\Big), \tag{4.40}$$

where Tr refers to the trace of the matrix.

4.4 Implementation details

We end this chapter on computing funnels by discussing a few important implementation details.

4.4.1 Trajectory generation

An important step that is necessary for the success of our approach to computing funnels is the generation of a dynamically feasible open-loop control input $u_0 : [0, T] \mapsto \mathbb{R}^m$ and corresponding nominal trajectory $x_0 : [0, T] \mapsto \mathbb{R}^n$. A method that has been shown to work well in practice and scale to high dimensions is the direct collocation trajectory optimization method [17]. While this is the approach we use for the results in Chapter 7, other methods like the Rapidly Exploring Randomized Tree (RRT) or its asymptotically optimal version, RRT* can be used too [54, 53].

4.4.2 Initializing V and ρ

Algorithms 1 and 2 require an initial guess for the functions V and ρ. In [102], the authors use the Lyapunov function candidate associated with a time-varying LQR controller. The control law is obtained by solving a Riccati differential equation:

$$-\dot{S}(t) = Q - S(t)B(t)R^{-1}B^TS(t) + S(t)A(t) + A(t)^TS(t) \tag{4.41}$$

with final value conditions $S(t) = S_f$. Here $A(t)$ and $B(t)$ describe the time-varying linearization of the dynamics about the nominal trajectory x_0. The matrices Q and R are positive-definite cost-matrices. The function:

$$V_{guess}(t, \bar{x}) = (x - x_0(t))^T S(t)(x - x_0(t)) = \bar{x}^T S(t)\bar{x} \tag{4.42}$$

is our initial Lyapunov candidate. Setting ρ to a quickly increasing function such as an exponential is typically sufficient to obtain a feasible initialization.

Chapter 5

Funnel Libraries

5.1 Sequential composition

One can think of funnels computed using the machinery described in Chapter 4 as *robust* motion primitives (the robustness is to initial conditions and uncertainty in the dynamics). While we could define a *funnel library* simply as a collection $\mathcal{F}$ of funnels and associated feedback controllers, it will be fruitful to associate some additional structure with $\mathcal{F}$. In particular, it is useful to know how funnels can be *sequenced* together to form composite robust motion plans. In order to consider this more formally, we will first introduce the notion of *sequential composition* of funnels defined in [25].

Definition 2. [25] *An ordered pair* (F_1, F_2) *of funnels* $F_1 : [0, T_1] \to \mathcal{P}(\mathbb{R}^n)$ *and* $F_2 : [0, T_2] \to \mathcal{P}(\mathbb{R}^n)$ *is* sequentially composable *if* $F_1(T_1) \subset F_2(0)$.

In other words, two funnels are sequentially composable if the "outlet" of one is contained within the "inlet" of the other. A pictorial depiction of this is provided in Figure 5-1. We note that the sequential composition of two such funnels is itself a funnel.

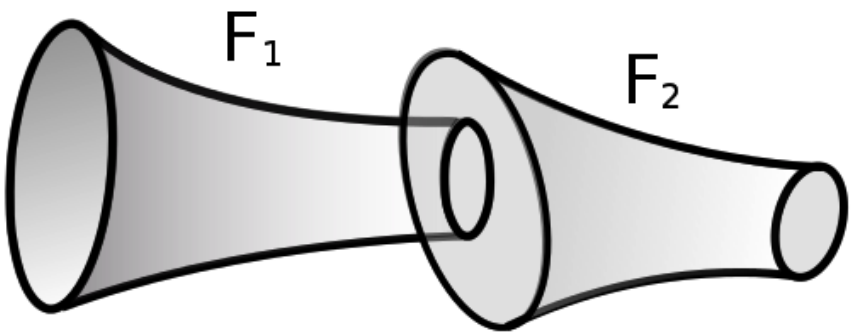


Figure 5-1: The ordered pair of funnels (F_1, F_2) is sequentially composable. The outlet of F_1 is contained within the inlet of F_2, i.e., $F_1(T_1) \subset F_2(0)$.

5.2 Exploiting invariances in the dynamics

For our purposes here, it is useful to introduce a slightly generalized notion of sequential composability that will allow us to exploit *invariances* (continuous symmetries) in the dynamics. In particular, the dynamics of large classes of mechanical systems such as mobile robots are often invariant under certain transformations of the state space. For Lagrangian systems, the notion of "cyclic coordinates" captures such invariances. A cyclic coordinate is a (generalized) coordinate of the system that the Lagrangian does *not* depend on. We can then write the dynamics of the system $\dot{x} = f(x(t), u(t))$ as a function of a state vector $x = [x_c, x_{nc}]$ which is partitioned into cyclic coordinates x_c and non-cyclic coordinates x_{nc} in such a way that the dynamics only depend on the non-cyclic coordinates:

$$\dot{x} = f(x_{nc}(t), u(t)). \tag{5.1}$$

For example, the dynamics of a quadrotor or fixed-wing airplane (expressed in an appropriate coordinate system) do not depend on the $x - y - z$ position of the system or the yaw angle.

Invariance of the dynamics of the system also implies that if a curve $t \mapsto (x(t), u(t))$ is a valid solution to the dynamics $\dot{x} = f(x(t), u(t))$, then so is the transformed solution:

$$t \mapsto (\Psi_c(x(t)), u(t)), \tag{5.2}$$

where Ψ_c is a transformation of the state vector along the cyclic coordinates. This allows us to make the following important observation.

Remark 2. *Suppose we are given a system whose dynamics are invariant to transformations Ψ_c along cyclic coordinates x_c. Let $F : [0, T] \to \mathcal{P}(\mathbb{R}^n)$ given by $t \mapsto F(t)$ be a funnel associated with this system. Then the transformed funnel given by $t \mapsto \Psi_c(F(t))$ is also a valid funnel. Hence, one can in fact think of invariances in the dynamics giving rise to an* infinite *family of funnels parameterized by shifts $\Psi_c(F)$ of a funnel F along cyclic coordinates of the system.*

Note that here we have implicitly assumed that the feedback controller:

$$u_f(t, x) = u_0(t) + \bar{u}_f(t, \bar{x}) = u_0(t) + \bar{u}_f(t, x - x_0(t)) \tag{5.3}$$

associated with the funnel has also been transformed to:

$$u_0(t) + \bar{u}_f(t, x - \Psi_c(x_0(t))). \tag{5.4}$$

In other words, we have transformed the reference trajectory we are tracking by Ψ_c. Henceforth, when we refer to transformations of funnels along cyclic coordinates we will implicitly assume that the feedback controller has also been appropriately modified in this manner.

These observations allow us to define a generalized notion of sequential composition that exploits invariances in the dynamics. We will refer to this notion as sequential composition *modulo invariances (MI).*

Definition 3. *An ordered pair (F_1, F_2) of funnels $F_1 : [0, T_1] \to \mathcal{P}(\mathbb{R}^n)$ and $F_2 : [0, T_2] \to \mathcal{P}(\mathbb{R}^n)$ is* sequentially composable modulo invariances (MI) *if there exists a transformation Ψ_c of the state along cyclic coordinates such that $F_1(T_1) \subset \Psi_c\big(F_2(0)\big)$.*

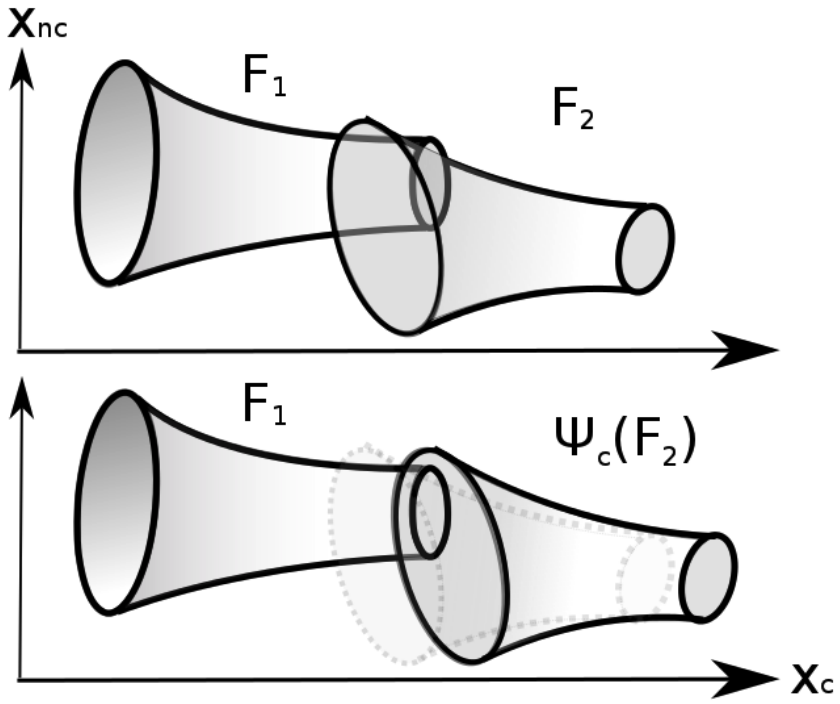


Figure 5-2: Sequential composition modulo invariances. The top row of the figure shows two funnels that are *not* sequentially composable in the sense of Definition 2. However, as shown in the bottom row of the figure, they *are* sequentially composable in the more general sense of Definition 3. By shifting the funnel F_2 (whose outline is plotted using dotted lines for reference) in the cyclic coordinate (x_c), we can ensure that the outlet of F_1 lies in the inlet of this shifted funnel $\Psi_c(F_2)$.

Informally, two funnels F_1 and F_2 are sequentially composable in this generalized sense if one can shift F_2 along the cyclic coordinates of the system and ensure that its inlet contains the outlet of F_1. Figure 5-2 provides a pictorial depiction of this.

One may think of sequential composability MI of funnels as being analogous to the compatibility condition required for sequencing trajectories in the library of a Maneuver Automaton [37]. Let π_{nc} denote the projection operator that maps a state $x = [x_c, x_{nc}]$ to the non-cyclic coordinates x_{nc}. In order to be able to sequence together two trajectories $x_1 : [0, T_1] \to \mathbb{R}^n$ and $x_2 : [0, T_2] \to \mathbb{R}^n$, one requires:

$$\pi_{nc}(x_1(T_1)) = \pi_{nc}(x_2(0)). \tag{5.5}$$

Note, however that imposing this compatibility condition on the nominal trajectories asso-

ciated with two funnels is neither necessary nor sufficient for the funnels being sequentially composable MI. Sequentially composability MI is concerned with the compatibility between funnels themselves and not the underlying nominal trajectories, which distinguishes our notion of compatibility between maneuvers from that of [37].

5.3 Runtime composability

The two notions of sequential composability we have considered so far allow us to produce new funnels from a given set of funnels by stitching them together appropriately in an offline preprocessing stage. We now introduce another notion of composability that is particularly important for reasoning about how funnels can be executed sequentially at runtime. We will refer to this notion as *runtime composability*.

Definition 4. *An ordered pair* (F_1, F_2) *of funnels* $F_1 : [0, T_1] \to \mathcal{P}(\mathbb{R}^n)$ *and* $F_2 : [0, T_2] \to \mathcal{P}(\mathbb{R}^n)$ *is* runtime composable *if for all* $x_{out} \in F_1(T_1)$*, there exists a transformation* Ψ_c *of the state along cyclic coordinates such that* $x_{out} \in \Psi_c\big(F_2(0)\big)$.

In other words, for any state x_{out} in the outlet of F_1, one can shift F_2 along cyclic coordinates and ensure that its inlet contains x_{out}. Hence, we can guarantee that it will be possible to execute the funnel F_2 (after appropriate shifting in the cyclic coordinates) once the funnel F_1 has been executed (though the *particular* shift Ψ_c required depends on x_{out} and thus will not be known until runtime). Hence, runtime composability of funnels allows us to exploit invariances in the dynamics of the system at runtime and effectively reuse our robust motion plans in different scenarios. As a simple example, a UAV flying through a cluttered environment can reuse a funnel computed for a certain starting position by shifting the funnel so its inlet contains the UAV's current state.

Remark 3. *It is easy to see from Definition 4 that two funnels* F_1 *and* F_2 *are runtime*

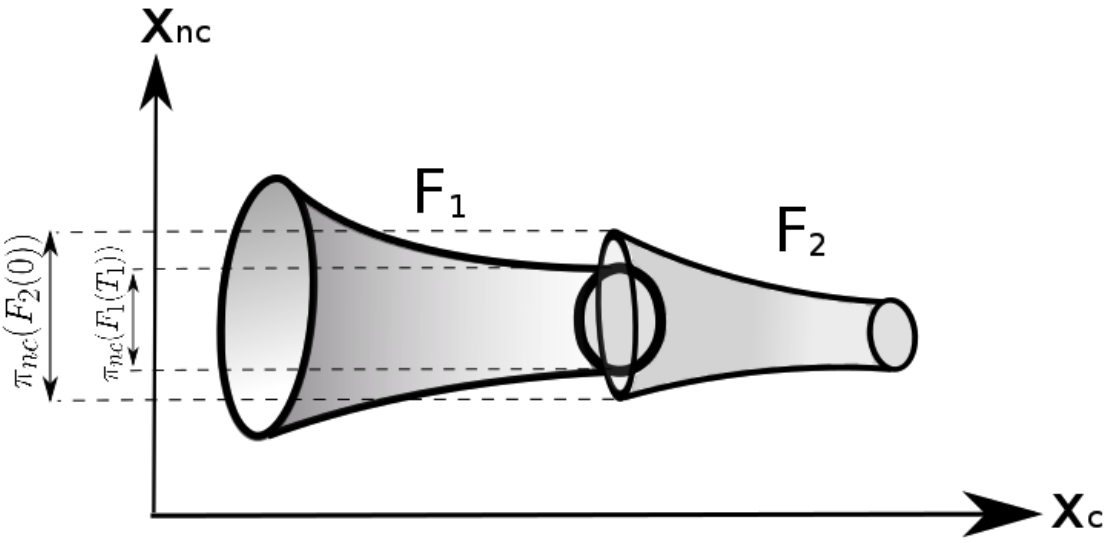


Figure 5-3: The figure shows two funnels that are *not* sequentially composable MI (since the inlet of funnel F_2 isn't large enough in the x_c dimension). However, since $\pi_{nc}(F_1(T_1)) \subset \pi_{nc}(F_2(0))$, they *are* runtime composable.

composable if and only if:

$$\forall x_{out} = [x_c, x_{nc}] \in F_1(T_1),\ \exists x_{0,c}\ s.t.\ [x_{0,c}, x_{nc}] \in F_2(0). \tag{5.6}$$

This condition is simply stating that we can shift the point x_{out} along cyclic coordinates in such a way that it is contained in the inlet of F_2. This condition in turn is equivalent to:

$$\pi_{nc}(F_1(T_1)) \subset \pi_{nc}(F_2(0)), \tag{5.7}$$

where as before π_{nc} denotes the projection onto the non-cyclic coordinates of the state space.

Figure 5-3 shows two funnels that are *not* sequentially composable MI. However, since $\pi_{nc}(F_1(T_1)) \subset \pi_{nc}(F_2(0))$, they *are* runtime composable.

We end our discussion on composability of funnels by noting the following relationship between the three notions of sequential composability we have discussed.

Remark 4. *The three notions of sequential composability we have discussed are related as follows:*

$$\textit{Sequential composability} \implies \textit{Sequential composability MI} \implies \textit{Runtime composability.}$$

The first implication is immediate from Definitions 2 and 3. The second implication follows from the following reasoning:

$$\begin{aligned} & F_1(T_1) \subset \Psi_c\big(F_2(0)\big) && \text{(Sequential composability MI, ref. Definition 3)} \\ \implies & \pi_{nc}(F_1(T_1)) \subset \pi_{nc}\big(\Psi_c(F_2(0))\big) = \pi_{nc}(F_2(0)) && \text{(Runtime composability, ref. Remark 3).} \end{aligned}$$

5.4 Checking composability

Given two funnels $F_1 : [0, T_1] \to \mathcal{P}(\mathbb{R}^n)$ and $F_2 : [0, T_2] \to \mathcal{P}(\mathbb{R}^n)$ defined as $F_1(t) = \{\bar{x} \in \mathbb{R}^n \mid V_1(t, \bar{x}) \leq \rho_1(t)\}$ and $F_2(t) = \{\bar{x} \in \mathbb{R}^n \mid V_2(t, \bar{x}) \leq \rho_2(t)\}$ for polynomials V_1 and V_2, this section describes how we can check sequential composability, sequential composability MI and runtime composability in an offline preprocessing stage.

5.4.1 Sequential composability

Sequential composability of (F_1, F_2) is equivalent to the following condition:

$$V_1(T_1, \bar{x}) \leq \rho_1(T_1) \implies V_2(0, \bar{x}) \leq \rho_2(0). \tag{5.8}$$

We can thus apply the generalized S-procedure (described in Chapter 3.2) and check sequential composability using the following simple SOS program:

$$\begin{aligned} \text{Find} \quad & L(\bar{x}) \\ \text{s.t.} \quad & \rho_2(0) - V_2(0,\bar{x}) - L(\bar{x})(\rho_1(T_1) - V_1(T_1,\bar{x})) \text{ is SOS}, \\ & L(\bar{x}) \text{ is SOS}. \end{aligned} \tag{5.9}$$

5.4.2 Sequential composition MI

In order to check sequential composability MI, we need to search for a shift Ψ_c along cyclic coordinates of the state such that $F_1(T_1) \subset \Psi_c(F_2(0))$. For the important special case in which the sets $F_1(T_1)$ and $F_2(0)$ are ellipsoids (corresponding to $V_1(T_1,\bar{x})$ and $V_2(0,\bar{x})$ being quadratic[1] in $\bar{x}$), we can cast this search as a semidefinite program (SDP). Suppose the sets $F_1(T_1)$ and $F_2(0)$ are given by:

$$F_1(T_1) = \{x \in \mathbb{R}^n \mid (x - x_1(T_1))^T S_1 (x - x_1(T_1)) \leq 1\} \tag{5.10}$$

$$F_2(0) = \{x \in \mathbb{R}^n \mid (x - x_2(0))^T S_2 (x - x_2(0)) \leq 1\}, \tag{5.11}$$

where $x_1 : [0, T_1] \to \mathbb{R}^n$ and $x_2 : [0, T_2] \to \mathbb{R}^n$ are the nominal trajectories around which the funnels were computed and S_1 and S_2 are positive definite matrices. We would like to search for a shift $\Delta_c \in \mathbb{R}^n$ (where the components of Δ_c corresponding to the non-cyclic coordinates are set to zero) such that the ellipsoid:

$$\Psi_c(F_2(0)) = \{x \in \mathbb{R}^n \mid [x - (x_2(0) + \Delta_c)]^T S_2 [x - (x_2(0) + \Delta_c)] \leq 1\} \tag{5.12}$$

is contained within the ellipsoid $F_1(T_1)$. The first step in obtaining the desired SDP is to note that the set $\Psi_c(F_2(0))$ can be represented equivalently as the image of the unit ball

[1]Note that the restriction to quadratic $V_1(T_1,\bar{x})$ and $V_2(0,\bar{x})$ is a relatively mild one. We are not imposing any conditions on the degree of V_1 and V_2 at times other than the endpoints.

under an affine map:

$$\Psi_c(F_2(0)) = \{Bu + x_2(0) + \Delta_c \mid \|u\|_2 \leq 1\}, \tag{5.13}$$

where $B = \text{chol}(S_2))^{-1}$. Here, $\text{chol}(S_2)$ is the Cholesky factorization of S_2 (guaranteed to exist and be invertible since S_2 is positive definite). Such a representation is a standard trick in semidefinite programming (see for example [21, Chapter 8.4.2]).

Introducing the notation $b := -S_1 x_1(T_1)$ and $c := x_1(T_1)^T S_1 x_1(T_1) - 1$, the condition that the ellipsoid $F_1(T_1)$ is a subset of the ellipsoid $\Psi_c(F_2(0))$ is then *equivalent* to being able to find a scalar $\lambda > 0$ such that the following matrix semidefiniteness constraint holds ([21, Chapter 8.4.2]):

$$\begin{bmatrix} -\lambda - c + b^T S_1^{-1} b & 0_{1\times n} & (x_2(0) + \Delta_c + S_1^{-1} b)^T \\ 0_{n\times 1} & \lambda I_{n\times n} & B \\ x_2(0) + \Delta_c + S_1^{-1} b & B & S_1^{-1} \end{bmatrix} \succeq 0. \tag{5.14}$$

Here, the matrices I and 0 represent the identity matrix and the all-zeros matrix respectively. Since this semidefiniteness condition is linear in λ and Δ_c, the problem of searching for these decision variables subject to $\lambda > 0$ and (5.14) is a SDP. This SDP will be feasible if and only if F_1 and F_2 are sequentially composable MI.

The problem of checking sequential composability MI in the more general non-ellipsoidal case is not directly amenable to such a SDP based formulation. However, if we *fix* Ψ_c, then the problem of checking sequential composability MI reduces to the problem of checking sequential composability. Thus, we can use the SOS program (5.9) to verify if a *given* Ψ_c yields the desired containment constraint $F_1(T_1) \subset \Psi_c(F_2(0))$. One natural choice is to set Ψ_c such that $\pi_c(\Psi_c(x_2(0))) = \pi_c(x_1(T_1))$, where π_c is the projection of the state onto the cyclic coordinates. Intuitively, this corresponds to shifting the funnel F_2 so that the start of its nominal trajectory is lined up along cyclic coordinates with the end of the nominal trajectory of F_1. If the SOS program is infeasible, a local search around this Ψ_c could yield the desired shift.

5.4.3 Runtime composability

In order to check runtime composability of F_1 and F_2, we need to check the inclusion $\pi_{nc}(F_1(T_1)) \subset \pi_{nc}(F_2(0))$. For the important special case where the sets $F_1(T_1)$ and $F_2(0)$ are ellipsoids, we can compute the projections exactly. This is because the projection of an ellipsoid onto a linear subspace is also an ellipsoid (see equation (4.39) for the exact formula for the projected ellipsoid). Checking if a given ellipsoid contains another is a straightforward application of semidefinite programming [21, Example B.1, Appendix B].

For the more general case, one might hope for a SOS programming based condition for checking $\pi_{nc}(F_1(T_1)) \subset \pi_{nc}(F_2(0))$. However, the existential quantifier inherent in the projection (see the equivalent condition (5.6)) makes it challenging to formulate such SOS conditions. Nevertheless, there exist general purpose tools such as *quantifier elimination* [29] for checking quantified polynomial formulas such as (5.6). While the worst-case complexity of doing general purpose quantifier elimination is poor, software packages such as QEPCAD [23] (or dReal [39], which is based on a different theoretical framework) can often work well in practice for specialized problems. We note however that in the examples considered in Sections 7 and 8 we will only be using funnels with ellipsoidal inlets and outlets and thus will not be concerned with this complexity.

5.5 Funnel library

A simple but useful generalization of the notions of composability introduced above can be obtained by checking the associated containment conditions at a given time τ_1 rather than at time T_1. For example, we will say that the ordered pair of funnels (F_1, F_2) is sequentially composable at time τ_1 if $F_1(\tau_1) \subset F_2(0)$. We will use similar terminology for the other notions of composability. Given a collection $\mathcal{F}$ of funnels associated with a dynamical system, we will associate a directed graph $\mathcal{G}(\mathcal{F})$ whose vertices correspond to funnels $F \in \mathcal{F}$. Two vertices corresponding to funnels F_i and F_j are connected by a directed edge (F_i, F_j) if and only if the ordered pair (F_i, F_j) is runtime composable at some specified time τ_i. We will

sometimes refer to τ_i as the execution time of funnel F_i.

Definition 5. *A* funnel library FL *associated with a given dynamical system is a tuple* $FL = (\mathcal{F}, \mathcal{G}(\mathcal{F}), \mathcal{C}, \{\tau_i\})$*, where* $\mathcal{F}$ *is a set of funnels for the dynamical system,* $\mathcal{G}(\mathcal{F})$ *is the directed graph representing which funnels are runtime composable,* $\mathcal{C}$ *is the set of feedback controllers associated with the funnels in* $\mathcal{F}$*, and* $\{\tau_i\}$ *is the set of execution times.*

Note that while we do not impose restrictions such as connectedness or strong connectedness on the graph $\mathcal{G}(\mathcal{F})$, it may be useful to impose such conditions based on the task at hand. For example, for tasks which require continuous operation (such as a UAV navigating indefinitely through a forest or a factory arm continuously placing objects onto a conveyor belt), we should require that $\mathcal{G}(\mathcal{F})$ be strongly connected (since if $\mathcal{G}(\mathcal{F})$ is not strongly connected, we are ruling out the possibility of executing certain funnels in the future). On a similar note, properties of the graph such as its *diameter* or *girth* may be related to the efficiency with which a certain task can be accomplished. Identifying which graph theoretic properties should be imposed on $\mathcal{G}(\mathcal{F})$ for different tasks is an interesting research avenue, but we do not pursue this in the present work.

Chapter 6

Real-time Planning with Funnels

Given a funnel library FL $= (\mathcal{F}, \mathcal{G}(\mathcal{F}), \mathcal{C}, \{\tau_i\})$ computed offline, we can proceed to use it for robust real-time planning in previously unseen environments. The robot's task specification may be in terms of a goal region that must be reached (as in the case of a manipulator arm grasping an object), or in terms of a nominal direction the robot should move in while avoiding obstacles (as in the case of a UAV flying through a forest or a legged robot walking over rough terrain). For the sake of concreteness, we adopt the latter task specification although one can adapt the contents of this chapter to the former specification. We further assume that the robot is provided with regions in the configuration space that obstacles are guaranteed to lie in and that the robot's sensors only provide this information up to a finite spatial horizon around the robot. Our task is to choose funnels from our library in a way that avoids obstacles while moving forward in the nominal direction.

The key step in our real-time planning approach is the selection of a funnel from the funnel library that doesn't intersect any obstacles in the environment. This selection process is sketched in the ReplanFunnel algorithm (Algorithm 3). Given the current state x of the robot and the locations and geometry of obstacles $\mathcal{O}$ in the environment, the algorithm searches through the funnels in the library that are runtime composable with the previous funnel that was executed. We assume that the funnels are (totally) ordered in a preference list. As an example, this ordering can be based on likely progress towards the goal or by

aggressiveness of the maneuvers (less aggressive maneuvers are given preference) for a UAV. For each funnel F_i, the algorithm tries to find a shift Ψ_c along cyclic coordinates of the system such that the shifted funnel $\Psi_c(F_i)$ satisfies two properties: (i) the current state x is contained in the inlet of the shifted funnel, (ii) the projection of the shifted funnel onto the coordinates of the state space corresponding to the configuration space[1] doesn't intersect any obstacles in $\mathcal{O}$. If we are able to find such a shift, we are *guaranteed* that the system will remain collision-free when the funnel is executed despite the uncertainties and disturbances that the system is subjected to.

The simplest way to try to choose Ψ_c is to set it such that the nominal trajectory x_i associated with the funnel lines up with the current state in the cyclic coordinates (i.e., using the notation of Chapter 5, $\pi_c(\Psi_c(x_i(0))) = \pi_c(x)$). Intuitively, this corresponds to shifting the funnel so that it is executed from the current location of the robot. We can then use standard collision-checking libraries such as the Bullet Collision Detection & Physics Library [30] to check if $\Psi_c(F_i)$ (projected onto the configuration space) intersects any obstacles. We will discuss more sophisticated ways of finding Ψ_c in Chapter 6.1.

If the search for a collision-free funnel in Algorithm 3 doesn't succeed, we assume that there is a failsafe maneuver that can be executed to keep the robot safe. For a ground vehicle, this could entail coming to a stop while for a quadrotor or fixed-wing airplane this may involve transitioning to a hover or propellor-hang mode. In certain cases, it is possible to derive geometric conditions on the environment that the robot will operate in (e.g., constraints on obstacle size and gaps between obstacles) that *guarantee* that a collision-free funnel will always be found by Algorithm 3 if the environment satisfies these conditions. We will see an example of this in Chapter 7.2 for a quadrotor system navigating through a forest of polytopic obstacles.

Algorithm 4 provides a sketch of the real-time planning and control loop, which applies the ReplanFunnel algorithm in a receding-horizon manner. At every control cycle, the robot's sensors provide it with a state estimate and report the locations and geometry of the set of obstacles $\mathcal{O}$ in the sensor horizon. The algorithm triggers a replanning of funnels if any of the

[1]Note that these projections can be computed in the offline computation stage.

1: Inputs: x (current state of system), $\mathcal{O}$ (reported obstacles in environment), previousFunnel (previous funnel that was executed)
2: **for** $i = 1, \ldots, \#(\mathcal{F})$ such that (previousFunnel, F_i) $\in \mathcal{G}(\mathcal{F})$ **do**
3: success $\Leftarrow$ Find a shift Ψ_c along cyclic coordinates such that x is contained in the inlet of $\Psi_c(F_i)$ and $\Psi_c(F_i)$ is collision-free w.r.t. $\mathcal{O}$
4: **if** success **then**
5: **return** $\Psi_c(F_i)$
6: **end if**
7: **end for**
8: **return** $F_{failsafe}$

Algorithm 3: ReplanFunnel

following three criteria are met: (i) if the system has executed the current funnel F_i for the associated execution time τ_i, (ii) if the current state of the system is no longer in the funnel being executed, or (iii) if the current funnel being executed is no longer collision-free. In principle, (ii) should not happen. However, in practice this can happen if the system received a disturbance that was larger than the ones taken into account for the funnel computations. Option (iii) can happen if the robot's sensors report new obstacles that were previously unseen.

1: $x \Leftarrow$ Initialize current state of the robot
2: $\mathcal{O} \Leftarrow$ Initialize obstacles in sensor horizon
3: $F_i \in \mathcal{F} \Leftarrow$ Initialize current planned funnel
4: **for** $t = 0, \ldots$ **do**
5: $x \Leftarrow$ Update current state of robot
6: $\mathcal{O} \Leftarrow$ Update obstacles in sensor horizon
7: replan $\Leftarrow$ Check if we have finished executing current funnel F_i for the associated execution time τ_i
8: insideFunnel $\Leftarrow$ Check if current state is still inside the current funnel F_i being executed
9: collisionFree $\Leftarrow$ Check if current funnel F_i is still collision-free with $\mathcal{O}$
10: **if** replan **or** ¬insideFunnel **or** ¬collisionFree **then**
11: $F_i \Leftarrow$ ReplanFunnels($x, \mathcal{O}$, F_i)
12: **else**
13: apply feedback control input u associated with current funnel F_i
14: **end if**
15: **end for**

Algorithm 4: Real-time planning loop

6.1 Shifting funnels at runtime

The main step in Algorithm 3 is the search for a shift Ψ_c along cyclic coordinates such that the shifted funnel $\Psi_c(F_i)$ will be collision-free with respect to the obstacles in $\mathcal{O}$ while containing the current state x in its inlet:

$$\begin{aligned} \text{Find} \quad & \Psi_c && (6.1) \\ \text{s.t.} \quad & x \in \Psi_c(F_i(0)), && (6.2) \\ & \pi_{conf}(\Psi_c(F_i)) \cap \mathcal{O} = \emptyset. \end{aligned}$$

Here, π_{conf} is the projection onto the configuration space variables of the state space. This optimization problem is non-convex in general since the free-space of the environment is non-convex. However, the number of decision variables is very small. In particular, if we parameterize the shift Ψ_c by a vector $\Delta_c \in \mathbb{R}^n$ (where the coordinates of Δ_c corresponding to the non-cyclic coordinates are set to zero) such that $\Psi_c(x) = x + \Delta_c$, the number of decision variables is equal to the number of cyclic coordinates of the system.

We can thus apply general-purpose nonlinear optimization tools such as gradient-based methods to solve this problem. In particular, the first constraint is equivalent to checking that the value of the Lyapunov function $V(0, \bar{x}) = V(0, x - (x_i(0) + \Delta_c))$ at time 0 is less than or equal to $\rho(0)$ (recall that the funnel is described as the $\rho(t)$ sub-level set of the function $V(t, \bar{x})$). As in Chapter 4, x_i here is the nominal trajectory corresponding to the funnel F_i. The second constraint can be evaluated using off-the-shelf collision-checking libraries.

Despite the small number of decision variables, for some applications general-purpose nonlinear optimization may be too slow to run in real-time. For the important special case where the cyclic coordinates form a subset of the configuration space variables (e.g., for a UAV), we propose another approach that allows us to search over a restricted family of shifts Ψ_c but has the advantage of being posed using *convex* quadratic constraints. The first observation is that the containment constraint (6.2) is a convex quadratic constraint for the

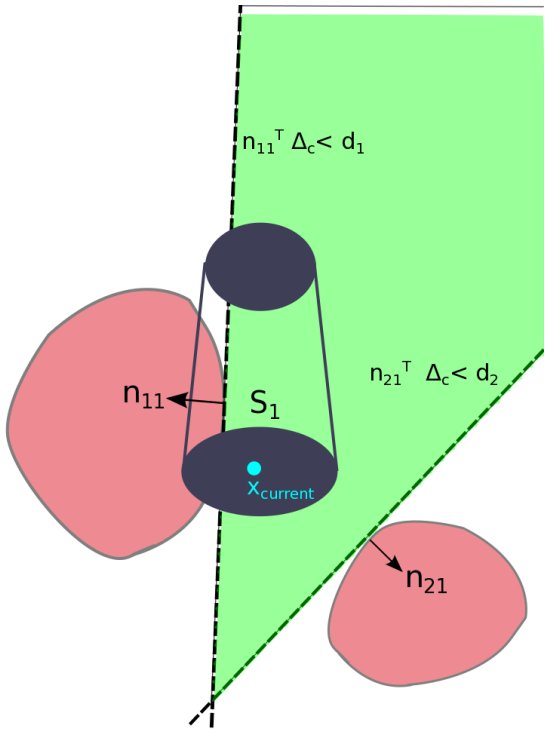


Figure 6-1: The two red regions are obstacles. The green region represents the set of allowable shifts of the funnel segment that satisfy the linear constraints given by the collision normals and collision distances.

case where the inlet of the funnel is an ellipsoid (i.e., the function $V(0, \bar{x})$ is a positive definite quadratic). If the inlet is not an ellipsoid, we can find an ellipsoidal inner approximation by solving a simple SOS program offline.

Next, we seek to find a set of convex constraints that will guarantee non-collision of the funnel. We will assume that $\pi_{conf}(F_i)$ is represented as a union of convex segments $\mathcal{S}_k$ and that the obstacles in the environment are also convex (we assume that non-convex obstacles have been decomposed into convex segments). For each funnel segment, we can find collision normals n_{jk} and collision distances d_{jk} to each obstacle $o_j \in \mathcal{O}$ (these are easily extracted from a collision-checking software). Figure 6-1 provides an illustration of this for a single convex segment $\mathcal{S}_1$. The two regions colored in red are obstacles. The collision normals are n_{11} and n_{21}. By definition, collision normals provide us with constraints such that any shift Δ_c of the segment $\mathcal{S}_1$ satisfying the linear constraint $n_{jk}^T \Delta_c < d_{jk}$ will be collision-free. The

region corresponding to this is shaded in green in the figure.

The constraints described above (containment of the current state in an ellipsoid and the linear constraints given by the collision normals) form a special case of convex Quadratically Constrained Quadratic Programs (QCQPs), for which there exist very mature software packages. For our examples in Chapter 7, we use the FORCES Pro package [35]. The package generates solver code tailored to the specific optimization problem at hand and is faster in our experience than using general-purpose convex QCQP solvers.

6.2 Global Planning

Algorithm 4 employed a receding horizon strategy for real-time planning. For tasks such as robot navigation through previously unmapped environments, such a replanning-based strategy is unavoidable since the robot's sensors will only report obstacles in the environment in some finite sensor horizon around the robot. Thus, the robot will need to replan as it makes progress through the environment and more obstacles are reported by its sensors. However, in certain scenarios where the robot has access to a larger portion of the environment, a planning strategy that is more *global* in nature may be appropriate. In general, the funnel primitives provide a discrete action space which can be searched by any heuristic planner - the primary considerations here are the additional constraint of containment of the current state in the inlet and the moderately more significant cost of collision checking. Here we briefly mention a few ways in which global planning may be achieved.

Grid-based planners: One can build on existing trajectory library-based planning approaches to quickly find collision-free sequences of funnels that minimize a certain performance criterion (e.g., distance traversed by the nominal trajectories in the sequence). For example, [99] uses a variant of the grid-based planning algorithm $A^\star$ to search through possible sequences of trajectories in a library. The additional cost of extending such an approach for planning with funnels is the cost that comes from pruning sequences of funnels that are in collision with obstacles in the environment.

Maneuver automata: The Maneuver Automaton [37] provides an alternative trajectory library-based approach that exploits continuous symmetries (such as shift invariance for ground vehicles or UAVs) to achieve efficient planning. The approach relies on having a number of "trim trajectories" and maneuvers that transition between them. One can then efficiently plan sequences of trims and maneuvers that start and end at prescribed points in space by solving a set of equations that has the same structure as an inverse kinematics problem. In order to extend this approach for real-time planning with funnels, one needs to (potentially approximately) solve the inverse kinematics problem subject to the constraint that the funnel sequence is collision-free.

Planning backwards: The real-time planning approaches we have discussed so far all plan forward in time. However, in cases where there is a well-defined goal set that one needs to reach, it may be more efficient to plan backwards. This kind of planning is a direct analog of the *preimage backchaining* approach [63] in the motion planning literature. One way to perform this backwards search would be to employ a randomized strategy similar to the Rapidly-exploring Randomized Tree (RRT). In particular, we can grow a tree of funnels backwards from the goal set by using the funnels in our funnel library as primitives for the extension operator for the RRT. One can again exploit invariances in the dynamics when performing this extension operation by randomly sampling shifts/rotations of funnels (while still maintaining sequential composability and non-collision of funnels). The termination criterion for this RRT-like algorithm is the containment of the current state of the robot in the tree of funnels. This is very similar to the LQR-Trees approach [102], with the following differences: (i) one would only use a pre-computed library of funnels in the tree rather than computing new funnels (which would be computationally infeasible for real-time implementation), and (ii) the extension operator in the tree would take into account non-collision of funnels with obstacles in the environment.

Chapter 7

Examples

In this chapter we apply the techniques presented in this thesis on two simulation examples. We will consider a hardware example in Chapter 8. The computations in this chapter were performed on a 3.4 GHz desktop computer with 16 GB RAM and 4 cores.

7.1 Ground Vehicle Model

As our first example we consider a ground vehicle model based on the Dubins car [36] navigating an environment of polytopic obstacles. A pictorial depiction of the model is provided in Figure 7-1. The vehicle is constrained to move at a fixed forward speed and can control the second derivative of its yaw angle ψ. We introduce uncertainty into the model by assuming that the speed of the vehicle is only known to be within a bounded range and is potentially time-varying. The full non-linear dynamics of the system are then given by:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ \dot{\psi} \end{bmatrix}, \qquad \dot{\mathbf{x}} = \begin{bmatrix} -v(t)\sin\psi \\ v(t)\cos\psi \\ \dot{\psi} \\ u \end{bmatrix} \tag{7.1}$$

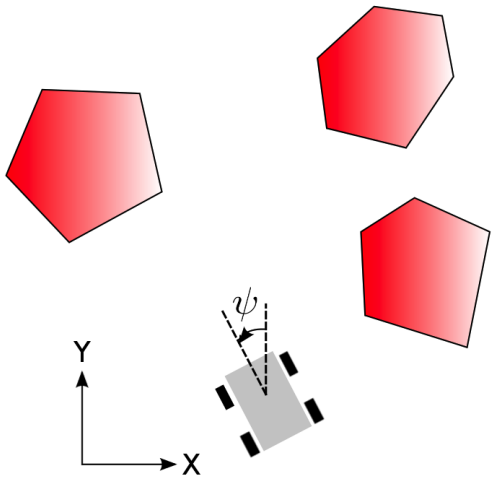


Figure 7-1: Illustration of the ground vehicle model.

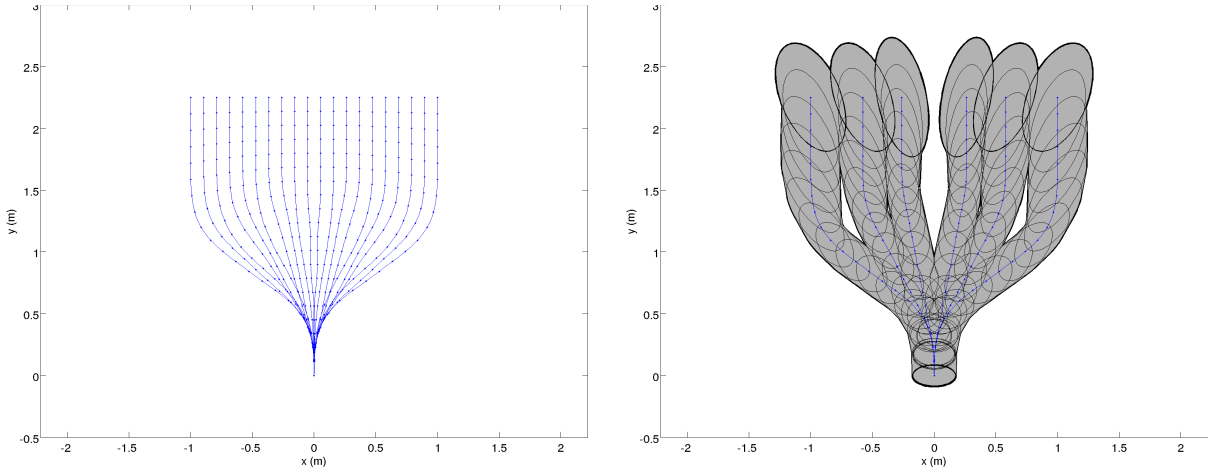
with the speed of the plane $v(t) \in [9.0, 11.0]$ m/s. The control input is bounded in the range $[-1000, 1000]$ rad/s^2.

The trajectory library, $\mathcal{T}$, computed for the ground vehicle consists of 20 trajectories and is shown in Figure 7-2(a). The trajectories $x_i(t) : [0, T_i] \mapsto \mathbb{R}^4$ and the corresponding nominal open-loop control inputs were obtained via the direct collocation trajectory optimization method [17] for the vehicle dynamics in (7.1) with $v(t) = 10$ m/s. The initial state $x_i(0)$ was constrained to be $[0, 0, 0, 0]$ and the final state $x_i(T_i)$ was varied in the x direction while keeping the y component fixed. We locally minimized a cost of the form:

$$J = \int_0^{T_i} [1 + u_0(t)^T R(t) u_0(t)]\, dt$$

where R is a positive-definite matrix. We constrained the nominal control input to be in the range $[-500, 500]$ rad/s^2 to ensure that feedback controllers computed around the nominal trajectories do not immediately saturate.

For each $x_i(t)$ in $\mathcal{T}$ we obtain controllers and funnels using the method described in Chapter 4. In order to obtain polynomial dynamics, we computed a (time-varying) degree 3 Taylor expansion of the dynamics of the system around the nominal trajectory. We note that with the right change of coordinates, one can express the dynamics of this system



(a) (b)

Figure 7-2: The plot on the left shows the trajectory library for the ground vehicle model. The plot on the right shows a selection of funnels from the funnel library projected onto the $x - y$ plane.

directly as a polynomial. In particular, we can introduce new indeterminates s and c for $\sin(\psi)$ and $\cos(\psi)$, and impose the constraint that $s^2 + c^2 = 1$ (this equality constraint is easily imposed in the sums-of-squares programming framework). However, this increases the dimensionality of the state space and in practice we find that the time-varying Taylor approximation accurately captures the nonlinearities of the system.

The approach from Chapter 4.3.2 along with the time-sampled approximation described in Chapter 4.2 (with 15 time samples) was used to synthesize a (time-varying) linear feedback controller around each trajectory. The methods described in Chapter 4.3.1 and 4.3.3 were used to take into account the parametric uncertainty and input saturations that the system is subject to. As described in Chapter 4.4.2, we used a time-varying LQR controller to initialize the funnel computations. The computation time for each funnel was approximately 5-10 minutes. A subset of the funnels is shown in Figure 7-2(b). Note that the four-dimensional funnels have been projected down to the $x - y$ dimensions for the purpose of visualization. The directed graph $\mathcal{G}(\mathcal{F})$ that encodes real-time composability between funnels (ref. Chapter 5.5) is fully connected.

The resulting funnel library was employed by Algorithm 4 for planning in real-time through environments with randomly placed obstacles. Figure 7-3 shows the funnels that

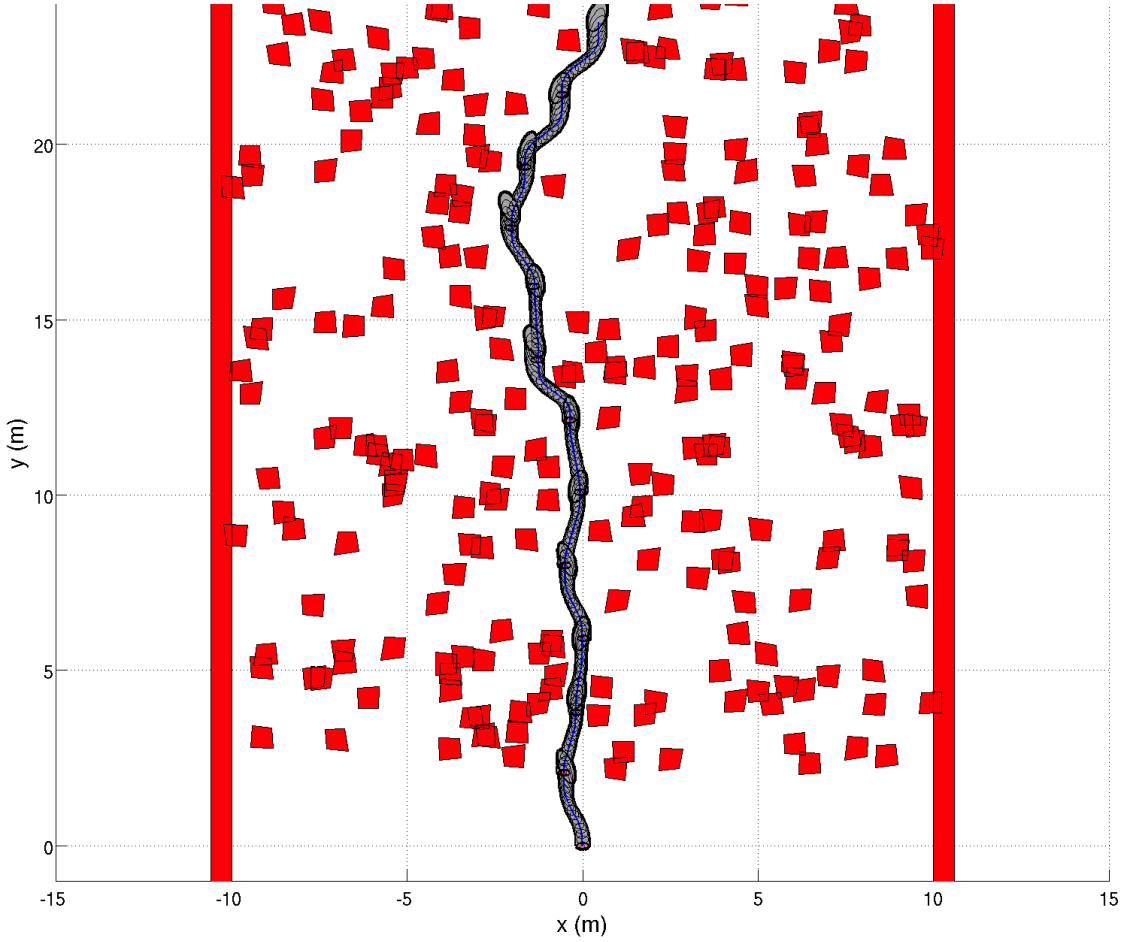


Figure 7-3: This plot shows the funnels that were executed in order for the ground vehicle model to traverse a randomly generated obstacle environment using the funnel library based real-time planning approach.

were executed in order to traverse a representative environment. The obstacle positions were randomly generated from a spatial Poisson process (with a density/rate parameter of 0.6 obstacles per m^2). Two further "barrier" obstacles were placed on the sides of the environment to prevent the vehicle from leaving the region containing obstacles. The planner was provided with a sensor horizon of 3m in the y direction (forward) and $\pm$2m in the x direction (side-to-side) relative to the position of the vehicle. Only obstacles in this sensor window were reported to the planner at every instant in time. The execution times τ_i for each funnel were set such that replanning occurred once 80% of the funnel was executed. The parametric uncertainty in the speed of the vehicle was taken into account in our simulation

by randomly choosing a speed $v(t) \in \{9.0, 11.0\}$ m/s after every execution time period has elapsed. The real-time planner employs the QCQP-based algorithm from Chapter 6.1 for shifting funnels in the $x-y$ directions (we did not exploit invariances in the yaw dimension of the state space in order to ensure that the vehicle moves in the forward direction and doesn't veer off to the sides). We use the FORCES Pro solver [35] for our QCQP problems. This resulted in our implementation of the real-time planner running at approximately $40 - 50$ Hz.

We performed extensive simulation experiments to compare our funnel library based robust planning approach with a more traditional trajectory library based method. In order to facilitate a meaningful comparison, we used the underlying trajectory library corresponding to our funnel library. The trajectory-based planner employs essentially the same outer-loop as our funnel-based approach (Algorithm 4). The key difference is that the planner chooses which maneuver to execute by evaluating which trajectory has the maximal clearance from the obstacles (as measured by Euclidean distance):

$$\max_{i} \min_{t,j} \text{dist}(x_i(t), o_j) \tag{7.2}$$

where $x_i(t) \in \mathcal{T}$ is a trajectory in the library and $o_j \in \mathcal{O}$ is an obstacle in the environment. Once a maneuver is chosen based on this metric, a time-varying LQR feedback controller computed along this trajectory is applied (the same LQR controller that is used to initialize the funnel computations).

To compare the two planners, we generated 100 obstacle environments randomly as described previously. For each environment, we ran the different planning algorithms until there was a collision of the vehicle with an obstacle. The distance in the y direction at the time of collision was recorded for each run. Figure 7-4 compares the performance of the different approaches. For each distance on the x-axis of the plot the height of the bar indicates the number of runs for which the vehicle traveled beyond this distance. As is evident from the plot, our funnel library based approach provides a significant advantage over the trajectory-based method.

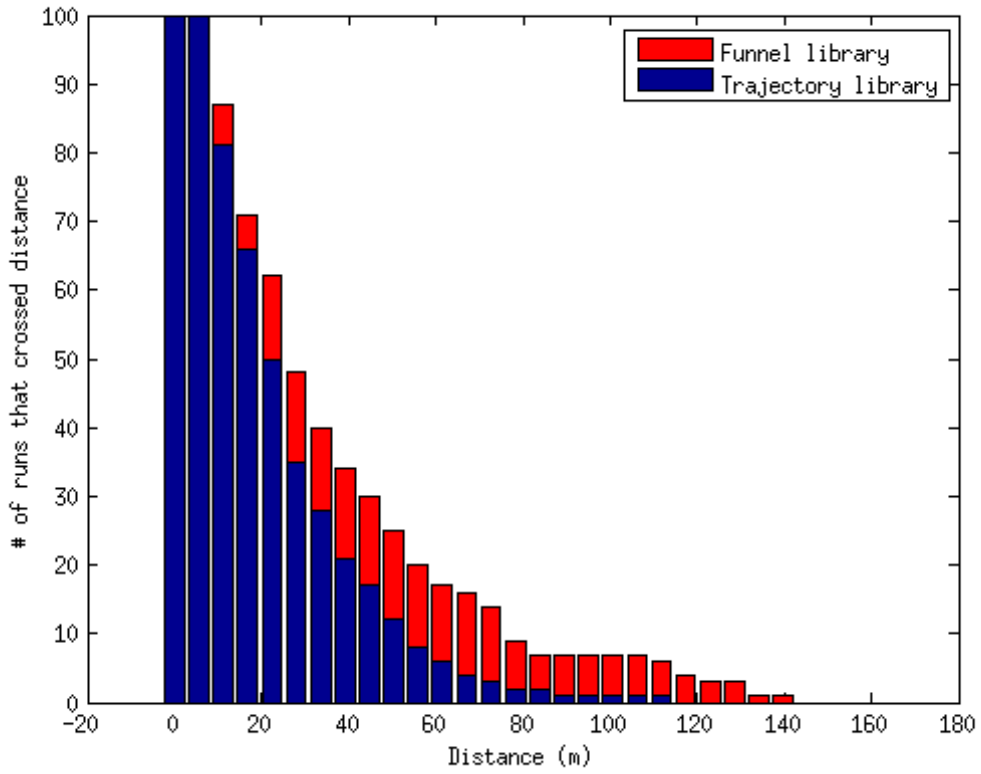


Figure 7-4: A bar plot comparing the performance of the funnel library approach with one based on a trajectory library.

This advantage can be partially understood by considering a specific example. In particular, Figure 7-5 demonstrates the utility of explicitly taking into account uncertainty during the planning process. There are two obstacles in front of the vehicle. The two options available to the plane are to fly straight in between the obstacles or to maneuver aggressively to the right and attempt to go around them. If the motion planner didn't take uncertainty into account and simply chose to maximize the clearance in terms of Euclidean distance from the nominal trajectory to the obstacles (see equation (7.2)), it would choose the trajectory that goes right around the obstacles. However, taking the funnels into account leads to a different decision: going straight in between the obstacles is guaranteed to be safe even though the distance to the obstacles is smaller. In contrast, the maneuver that avoids going in between obstacles is less robust to uncertainty and could lead to a collision. The utility of safety

guarantees in the form of funnels is especially important when the margins for error are small and making the wrong decision can lead to disastrous consequences.

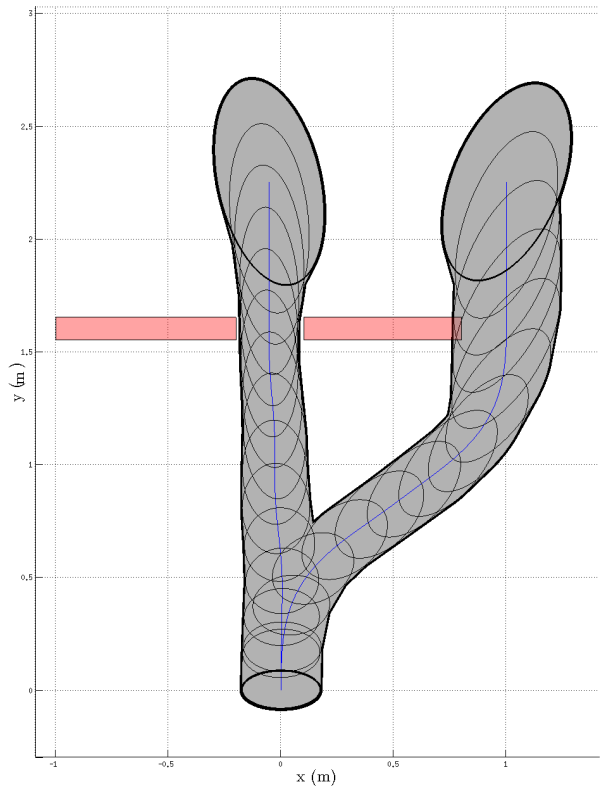


Figure 7-5: This figure shows the utility of explicitly taking uncertainty into account while planning. The intuitively more risky strategy of flying in between two closely spaced obstacles is guaranteed to be safe, while the path that avoids going in between obstacles is less robust to uncertainty and could lead to a collision.

7.2 Quadrotor Model

The next example we consider is a model of a quadrotor system navigating through a forest of polygonal obstacles. A visualization of the system is provided in Figure 7-6. The goal of this example is to demonstrate that we can derive simple geometric conditions on the environment that guarantee collision-free flight. In other words if the environment satisfies these conditions, the Algorithm 3 presented in Chapter 6 will always succeed in finding a collision-

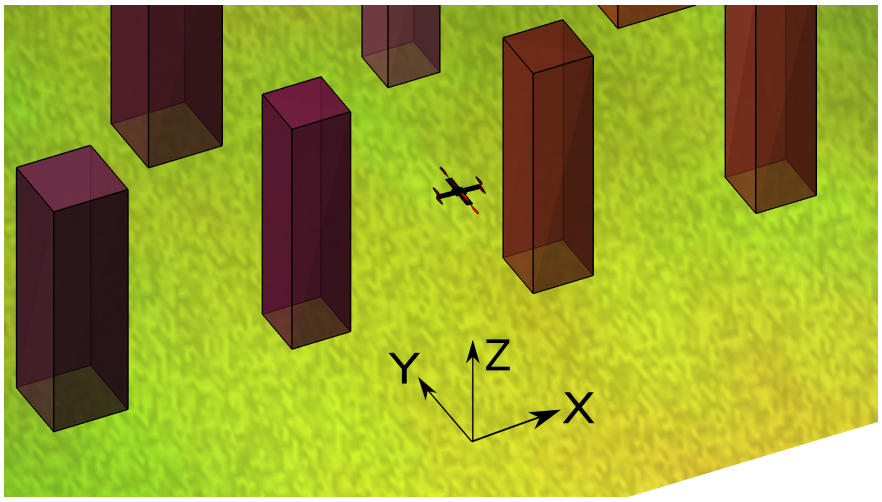


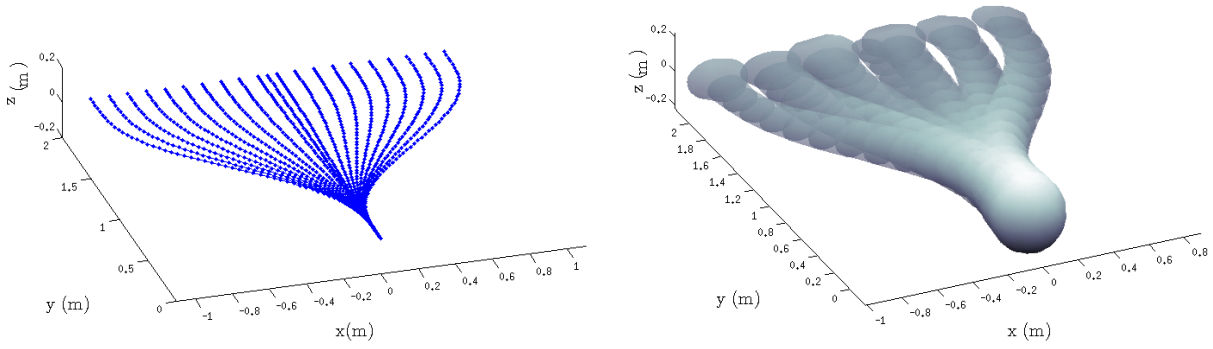
Figure 7-6: Visualization of the quadrotor system.

free funnel from the library and the quadrotor will fly forever through the environment with no collisions.

The quadrotor model has a 12 dimensional state space consisting of the x-y-z position of the centre of mass, the roll-pitch-yaw of the body, and the time derivatives of these configuration space variables. The dynamics model we use is identical to the one presented in [71] with an additional uncertainty term in the form of a "cross wind". We modeled this with a bounded uncertainty term on the acceleration of the x position: $\ddot{x} = \ddot{x}_{nominal} + \Delta$, with $\Delta \in [-0.1, 0.1]\ m/s^2$.

Figure 7-7(a) plots the trajectory library we use. The funnel library consists of 20 maneuvers and was created in a manner similar to the ground vehicle example. In particular, the trajectories $x_i(t) : [0, T_i] \mapsto \mathbb{R}^{12}$ and the corresponding nominal open-loop control inputs were obtained via the direct collocation trajectory optimization method [17]. The initial state $x_i(0)$ was constrained to have a forwards speed of 2 m/s (i.e, $x_i(0) = [0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0]$) and the final state $x_i(T_i)$ was varied in the x direction while keeping all other components of the state fixed. We locally minimized a cost of the form:

$$J = \int_0^{T_i} [1 + u_0(t)^T R(t) u_0(t)]\, dt$$



(a) (b)

Figure 7-7: The plot on the left shows the trajectory library for the quadrotor model. The plot on the right shows a selection of funnels from the funnel library projected onto the $x - y - z$ space.

where R is a positive-definite matrix. In addition to the 20 maneuvers, the library also consists of a "trim" trajectory corresponding to the quadrotor flying forward at a constant speed of 2 m/s.

For each $x_i(t)$ in $\mathcal{T}$ we obtain controllers and funnels using the method described in Chapter 4. We obtained time-varying Taylor expansions of degree 3 computed around the nominal trajectory. The approach from Chapter 4.3.2 along with the time-sampled approximation described in Chapter 4.2 (with 15 time samples) was used to synthesize a (time-varying) linear feedback controller around each trajectory. The methods described in Chapter 4.3.1 were used to take into account the parametric uncertainty that the system is subject to. The computation time for each funnel was approximately 20-25 minutes. The directed graph $\mathcal{G}(\mathcal{F})$ that encodes real-time composability between funnels (ref. Chapter 5.5) is fully connected. Moreover, the funnel corresponding to the trim trajectory is sequentially composable modulo invariances (ref. Chapter 5.2) with the other maneuvers in the library, allowing us to apply the trim trajectory before or after any of the maneuvers. A subset of the funnels is shown in Figure 7-7(b). Note that the twelve-dimensional funnels have been projected down to the $x - y - z$ dimensions for the purpose of visualization.

Given this funnel library $\mathcal{F}$, we will show how one can derive simple geometric conditions on the environment that guarantee that a collision-free funnel will always be found during real-time planning. In order to simplify the analysis, we will assume that the quadrotor

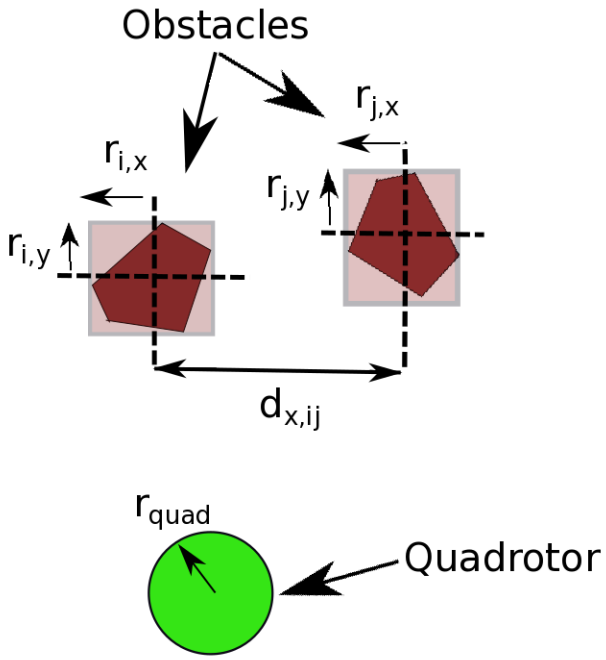


Figure 7-8: Obstacle bounding boxes and quadrotor radius.

is navigating through a 2.5D environment (i.e., the obstacles in the environment are 2D polygons extruded in the z-direction). But, we note that our analysis can be extended to fully three dimensional environments. We will treat the geometry of the quadrotor as a sphere of radius r_{quad}.

Since we are only considering 2.5D environments, it is sufficient to project the environment and funnels down to the $x-y$ plane. Consider (axis aligned) bounding boxes for the 2D obstacles (see Figure 7-8). Denote the center of the bounding box for obstacle o_i as $(o_{i,x,}, o_{i,y})$ for $i \in \{1, \ldots, N_{obs}\}$ (N_{obs} is the number of obstacles in the robot's sensor horizon). Suppose without loss of generality that the quadrotor's x-y-z position is $(0, 0, 0)$. We will first derive conditions on the distances $o_{i,y}$, $d_{x,ij} := |o_{i,x} - o_{j,x}|$, and sizes $r_{x,i}$, $r_{y,i}$ (ref. Figure 7-8) such that there exists a collision-free funnel in $\mathcal{F}$ *assuming* the quadrotor's x-y-z position is $(0, 0, 0)$. We will then extend this analysis to derive conditions that guarantee that a collision-free funnel will always be found as the quadrotor applies Algorithm 3 in a receding horizon manner to fly continuously through the environment.

For each funnel $F_i \in \mathcal{F}$ compute bounding boxes for the inlet and outlet of the funnel (see Figure 7-9(b)). We will refer to the dimensions of these bounding boxes as depicted in the figure as $f_{i,y}^{out}, f_{i,x}^{out}, f_{i,y}^{in}, f_{i,x}^{in}$. Define

$$f_x := \max(f_{1,x}^{out}, ..., f_{N,x}^{out}, f_{1,x}^{in}, ..., f_{N,x}^{in}) + r_{quad},$$

where N is the number of funnels in the library (21 in our case). Similarly, define:

$$f_y := \max(f_{1,y}^{out}, ..., f_{N,y}^{out}, f_{1,y}^{in}, ..., f_{N,y}^{in}) + r_{quad}.$$

Next, define δ as shown in Figure 7-9(a) as the distance (in the x-dimension) of the endpoints of trajectories in $\mathcal{T}$. Denote by Δ_x the distance (in the x-dimension) between the endpoints of the most aggressive trajectories in $\mathcal{T}$ and let Δ_y be the distance in the y-direction that each trajectory in the library covers (see Figure 7-9(a)).

Define $r_x := \max(r_{1,x}, ..., r_{N_{obs},x})$, $r_y := \max(r_{1,y}, ..., r_{N_{obs},y})$. Then consider the following conditions on the obstacles:

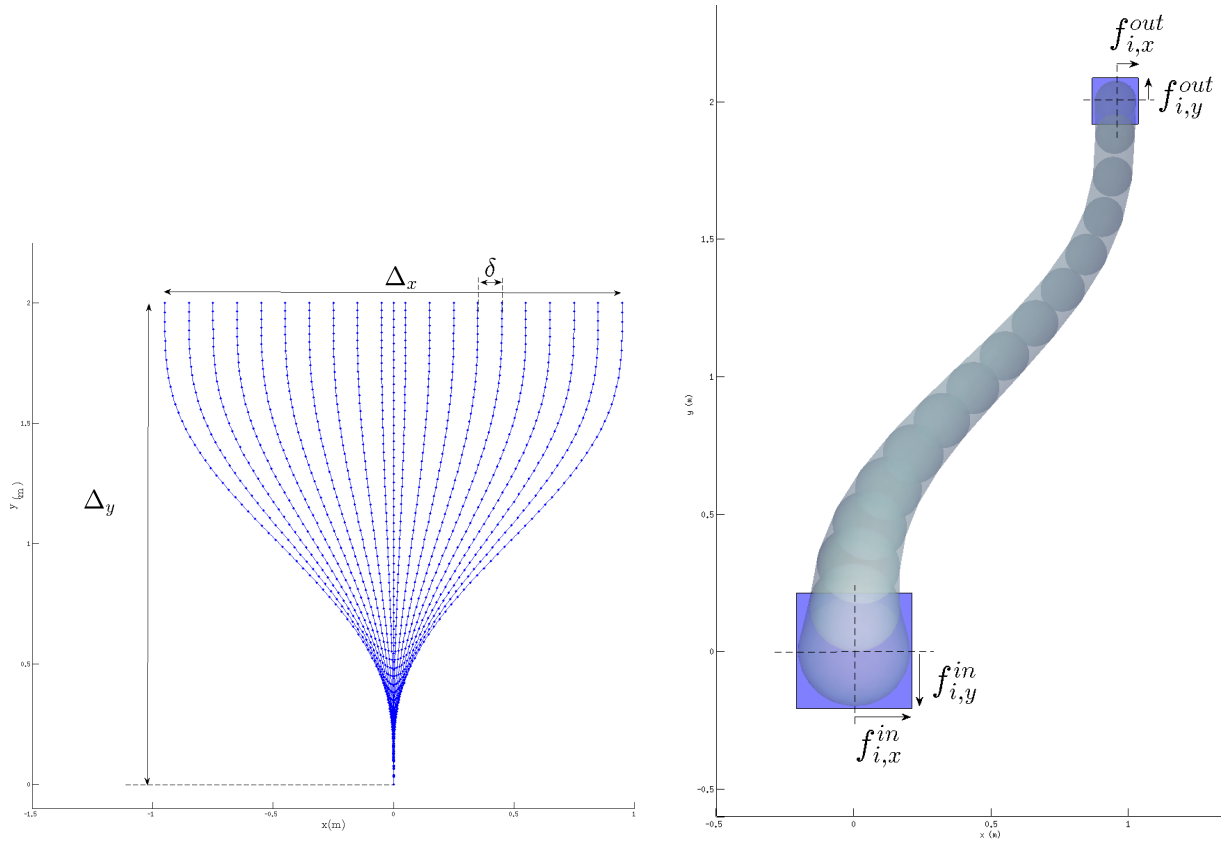
$$o_{i,y} > \Delta_y, \ \forall i \in \{1, \ldots, N_{obs}\} \tag{7.3}$$

$$r_y < f_y \tag{7.4}$$

$$r_x < \Delta_x - 2f_x \tag{7.5}$$

$$d_{x,ij} > 2f_x + \delta, \ \forall i, j \in \{1, \ldots, N_{obs}\}, \ i \neq j. \tag{7.6}$$

It is straightforward to see that these conditions imply the existence of a funnel in $\mathcal{F}$ that is collision-free when executed from the quadrotor's location at $(0, 0, 0)$ (i.e., Algorithm 3 will succeed assuming that the full 12 dimensional state of the quadrotor is contained within the inlet of all the funnels in $\mathcal{F}$). In particular, the condition (7.3) ensures that the funnels are not too close to the quadrotor's starting location. Condition (7.4) prevents an obstacle from extending so far in the y-direction that it collides with the middle segments



(a) (b)

Figure 7-9: Notation for geometry of trajectories and funnels.

of a funnel. Condition (7.5) ensures that obstacles do not extend so far in the x-direction that there is no funnel in the library that goes around the obstacle (e.g., imagine a wall in front of the quadrotor). And finally (7.6) ensures that the gap in the x-direction between any two obstacles is large enough for *some* funnel in $\mathcal{F}$ to fit through. Then conditions (7.5) and (7.6) ensure that this gap occurs in a portion of the space that is reachable by one of the funnels in the library (and not in some far off location in the x-direction where a funnel in $\mathcal{F}$ does not extend to).

We can now extend this analysis to the scenario in which the quadrotor is flying along continuously through the environment by applying Algorithm 3 in a receding horizon manner. We will assume that the sensor horizon of the quadrotor (i.e., the range in which obstacles are reported) is greater than Δ_y in the y-direction and greater than $\Delta_x/2$ in the x-direction. We then need to ensure that when the quadrotor finishes executing a funnel, it does not find itself in a position where there are obstacles that are very close to it. In order to do this, we will first replace the conditions (7.3) and (7.6) with the following condition on the separation between obstacles:

$$d_{y,ij} := |o_{i,y} - o_{j,y}| > \Delta_y + f_y \tag{7.7}$$

OR

$$d_{x,ij} > 2f_x + \delta. \tag{7.8}$$

This condition ensures that obstacles are separated enough in *either* the x or y directions. However, this is still not enough to prevent cases where the quadrotor executes a funnel and finds itself too close to an obstacle. In particular, suppose that the quadrotor starts off at location $(0, 0, 0)$ and that there is a single obstacle in the environment located at $(0, 1.1\Delta_y + f_y)$ (the size of the obstacle will not matter in this example). When the quadrotor applies Algorithm 3 at location $(0, 0, 0)$ to find a funnel, all the funnels in the library will be collision-free. Let us suppose that the algorithm chooses the funnel corresponding to the quadrotor flying straight and ends up in location $(0, \Delta_y)$. At this point, the obstacle is too close to the quadrotor and a collision-free funnel will not be found. To prevent cases such as

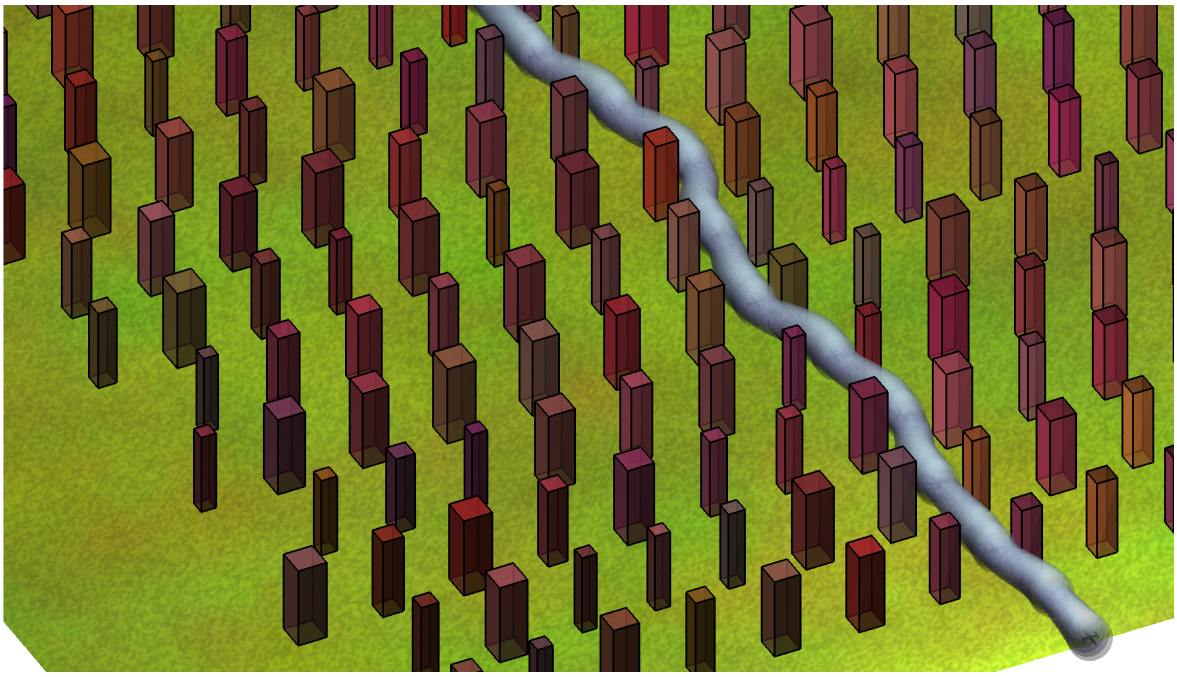


Figure 7-10: The plot shows the quadrotor maneuvering through a forest of polytopic obstacles in a collision-free manner. The environment satisfies simple geometric conditions that allow us to guarantee collision-free flight forever. Note that the visualized funnels have been inflated to take into account the size of the quadrotor.

this, we can use the trim maneuver in $\mathcal{F}$ corresponding to the quadrotor flying straight in order to "pad" the distance to the obstacles. In particular, we can apply the trim maneuver until the quadrotor is at distance Δ_y from the obstacle (in the y-direction) and *then* use Algorithm 3 to replan. This will guarantee that Algorithm 3 will succeed and thus our analysis is complete.

Figure 7-10 shows an example of the quadrotor system navigating through an environment that satisfies the geometric conditions derived above. The figure shows the sequence of funnels executed by the quadrotor to traverse the environment. Note that the funnels depicted in the plot have been inflated by r_{quad} in order to take into account the physical extent of the quadrotor.

We end this section by noting that our goal here has been to demonstrate the possibility of imposing geometric conditions on the environment that guarantee collision-free flight. Going forwards, our analysis may be varied or tightened in many ways. For example, we can

extend it in a straightforward manner to fully 3D environments. Further, we have assumed that the quadrotor is only planning a single funnel at a time (in contrast to sequences of funnels). Planning sequences of funnels may help loosen some of the restrictions on the environment that we have made in our analysis.

Chapter 8

Hardware experiments on a fixed-wing airplane

In this chapter, we will validate the key components of the approach presented in this thesis on a small fixed-wing airplane performing a challenging obstacle avoidance task. The goal of these hardware experiments is to answer the following important practical questions:

- Can we obtain models of a real-world challenging nonlinear dynamical system that are accurate enough to compute funnels that are valid in reality?
- Can we implement the real-time planning algorithm described in Chapter 6 to operate at the required rate given realistic computational constraints?
- Can we demonstrate our planning algorithm on a realistic and challenging obstacle avoidance task?

8.1 Hardware platform

The hardware platform chosen for these experiments is the SBach RC airplane manufactured by E-flite shown in Figure 8-1(a). The airplane is very light (76.6 g) and highly maneuverable, thus allowing for dramatic obstacle avoidance maneuvers in a tight space. The control inputs

to the SBach are raw servo commands to the control surfaces (ailerons, rudder, elevator) and a raw throttle setting. These commands are sent through a modified 2.4 GHz RC transmitter at an update rate of 50 Hz.

8.2 Task and experimental setup

The experimental setup is shown in Figure 8-1(b). The airplane is launched from a simple rubber-band powered launch mechanism at approximately $4-5$ m/s. The goal is to traverse the length of the room while avoiding the obstacles placed in the experimental arena. The airplane's planner is *not* informed where the obstacles are beforehand; rather, the obstacle positions and geometry are reported to the planner only once the airplane clears the launcher. This simulates the receding-horizon nature of realistic obstacle avoidance tasks where the obstacle positions are not known beforehand and planning decisions must be taken in real-time. The experiments are performed in a Vicon motion capture arena that reports the airplane and obstacle poses at 120 Hz. All the online computation is performed on an off-board computer with four Intel i7 2.9GHz processors and 16 GB RAM.

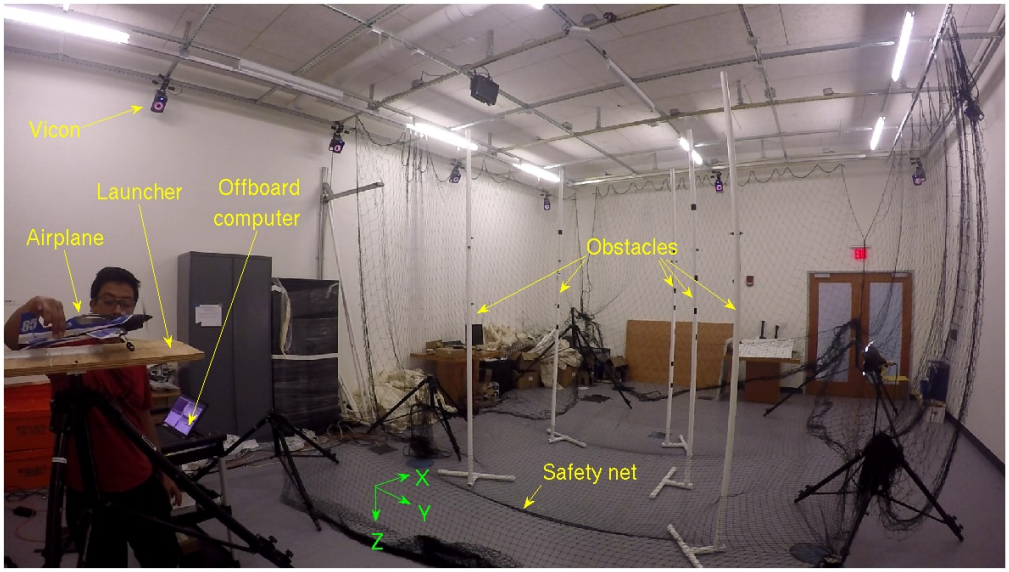
8.3 Modeling and system identification

Our dynamics model of the airplane is based on the model described in [95] ([98] is also a good reference for modeling fixed-wing airplanes). The model has 12 states:

$$\mathbf{x} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, P, Q, R].$$

Here, $+x$ is in the forward direction, $+y$ is to the right and $+z$ is downwards as depicted in Figure 8-1(b) (this is the standard North-East-Down coordinate frame used in aeronautics). The states ϕ, θ, ψ are the roll, pitch and yaw angles. The variables P, Q, R are the components of the angular velocity expressed in the body coordinate frame. The control inputs of the model are the angles of the ailerons, rudder and elevator, along with the speed of the

(a) The E-flite SBach RC airplane used for hardware experiments.



(b) Experimental setup and coordinate system.

Figure 8-1: Airplane hardware and experimental setup.

propellor. Thus, we have 4 control inputs since the ailerons are coupled to deflect in opposite directions by the commanded magnitude.

The airplane is treated as a rigid body with aerodynamic and gravitational forces acting on it. Here, we provide a brief description of our model of the aerodynamic forces:

Propellor thrust The thrust from the propellor is proportional to the square of the propellor speed. The constant of proportionality was obtained by hanging the airplane (with the propellor pointing downwards) from a digital fish scale and measuring the thrust produced for a number of different throttle settings.

Lift/drag on aerodynamic surfaces The lift and drag forces on the ailerons, rudder, elevator and tail of the airplane were computed using the flat-plate model. The flat-plate model was also used as a baseline for the lift and drag coefficients of the wings, but an angle-of-attack dependent correction term was added. This correction term was fit from experimental data obtained from passive (i.e., unactuated) flights in a manner similar to [74]. Since lift and drag forces are dependent on the airspeed over the aerodynamic surface, we need to take into account the effect of "propwash" (i.e. the airflow from the propellor). The relationships between the throttle speed command and the propellor downwash speed over the different control surfaces were measured using a digital anemometer in a manner similar to [95].

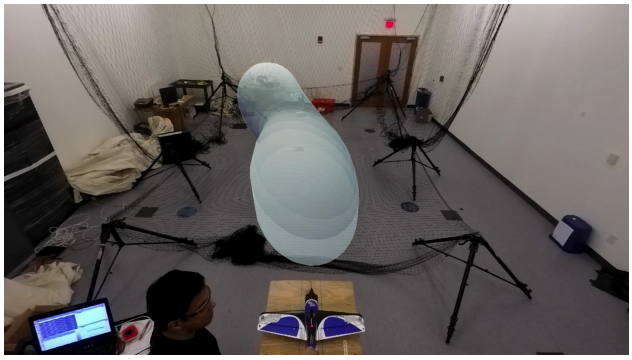
Body drag The drag on the airplane body is approximated as a quadratic drag term whose drag coefficient is fit from data.

As described above, many of the parameters in the model were obtained directly from physical experiments and measurements. However, some model parameters are more difficult to measure directly. These include the moments of inertia of the airplane and the coefficient of drag associated with the airplane body. The prediction-error minimization method in the

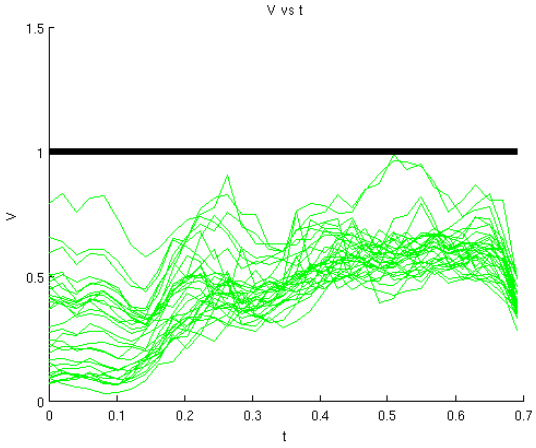
Matlab System Identification Toolbox [61] was used to fit these parameters and to fine-tune the measured parameters. In particular, we collected data from 15-20 flights (each lasting approximately 0.5–0.7 seconds) where the control inputs were excited using sinusoidal signals of varying frequency and amplitude.

8.4 Funnel validation

The first major goal of our hardware experiments was to demonstrate that our model of the SBach airplane is accurate enough to compute funnels that are truly meaningful for the hardware system. To this end, we computed a funnel (shown in Figure 8-2(a)) for the airplane using the approach described in Chapter 4. We first estimated the set of initial states that the launcher mechanism causes the airplane to start off in (here, by "initial state" we mean the state of the airplane as soon as it has cleared the launcher mechanism). This was done by fitting an ellipsoid around the initial states observed from approximately 50 experimental trials. We then used direct collocation trajectory optimization [17] to design an open-loop maneuver that makes the airplane bend towards the left. The initial state of the trajectory is constrained to be equal to the mean of the experimentally observed initial states and the control inputs are constrained to satisfy the limits imposed by the hardware. Next we computed a time-varying LQR (TVLQR) controller around this nominal trajectory. This controller was tuned largely in simulation to ensure good tracking of the trajectory from the estimated initial condition set. The resulting closed-loop dynamics were then Taylor expanded around the nominal trajectory to degree 3 in order to obtain polynomial dynamics. Finally, we used SOS programming to compute the funnel depicted in Figure 8-2(a) using the time-sampled approximation described in Section 4.2 (with 10 time samples). The inlet of the funnel was constrained to contain the experimentally estimated set of initial conditions. We observed that the tuned TVLQR controller does not saturate the control inputs for the most part and thus we did not find it necessary to take input saturations into account in our funnel computation. Further, we wanted to assess the validity of our funnel for the nominal dynamics model of the airplane and did not take into account any



(a) A depiction of the funnel that was validated on hardware. The funnel has been projected down to the $x - y - z$ coordinates of the state space and then reprojected onto the camera image.



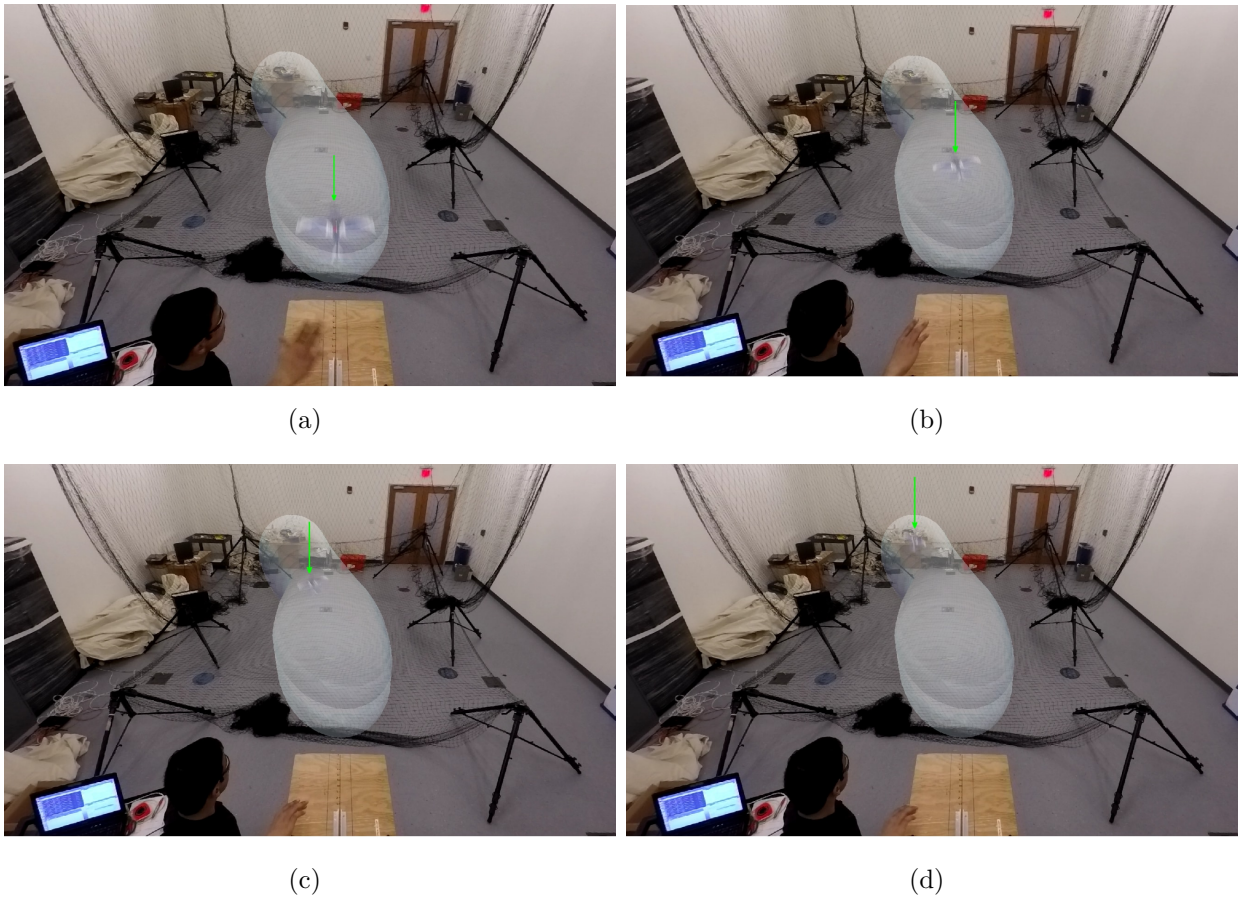
(b) The value of the Lyapunov function (V) plotted as a function of time for 30 different trials of the airplane started from different initial conditions in the inlet of the funnel. The 1-sublevel set of the Lyapunov function corresponds to the funnel (i.e., a Lyapunov function of 1 or less corresponds to the airplane being inside the funnel). All 30 of the trajectories remain inside the computed funnel for the entire duration of the maneuver.

Figure 8-2: Validating funnels on the fixed-wing airplane.

uncertainty in the model. The funnel computation took approximately 1 hour.

We validate the funnel shown in Figure 8-2(a) with 30 experimental trials of the airplane executing the maneuver corresponding to the funnel. The airplane is started off in different initial conditions in the inlet of the funnel and the TVLQR controller is applied for the duration of the maneuver. Figure 8-3 shows still images of a sample flight of the airplane executing the maneuver with the funnel superimposed onto the images. A video with a visualization of the funnel and flights through it is available online at https://youtu.be/cESFpLgSb50.

Figure 8-2(b) provides a more quantitative perspective on the flights. In particular, the figure shows the value of the Lyapunov function $V(\bar{x}, t)$ as a function of time achieved during the 30 experimental trials. Here, the Lyapunov function has been normalized so that the 1-level set corresponds to the boundary of the funnel. As the plot illustrates, all 30 trajectories remain inside the computed 12 dimensional funnel for the entire duration of the maneuver. This suggests that our model of the airplane is accurate enough to produce funnels that are



(a) (b)

(c) (d)

Figure 8-3: Still images of the SBach flying through a funnel. The funnel has been projected down to the $x - y - z$ coordinates of the state space and then reprojected onto the camera image.

indeed valid for the hardware system.

8.5 Obstacle avoidance experiments

The second major goal of our hardware experiments was to demonstrate the funnel library based real-time planning algorithm proposed in Chapter 6 on the obstacle avoidance task described in Chapter 8.2. Our first step was to design a rich trajectory library consisting of a large number of different maneuvers. We initialized the library with the maneuver from Chapter 8.4 and augmented it by computing trajectories with varying final states. In particular, the library consists of 40 trajectories which were obtained by discretizing the

final state in the y and z coordinates and using direct collocation trajectory optimization to compute locally optimal trajectories for the airplane model described in Chapter 8.3. The $x-y-z$ components of this trajectory library are depicted in Figure 8-4, with the trajectory from Chapter 8.4 highlighted in blue.

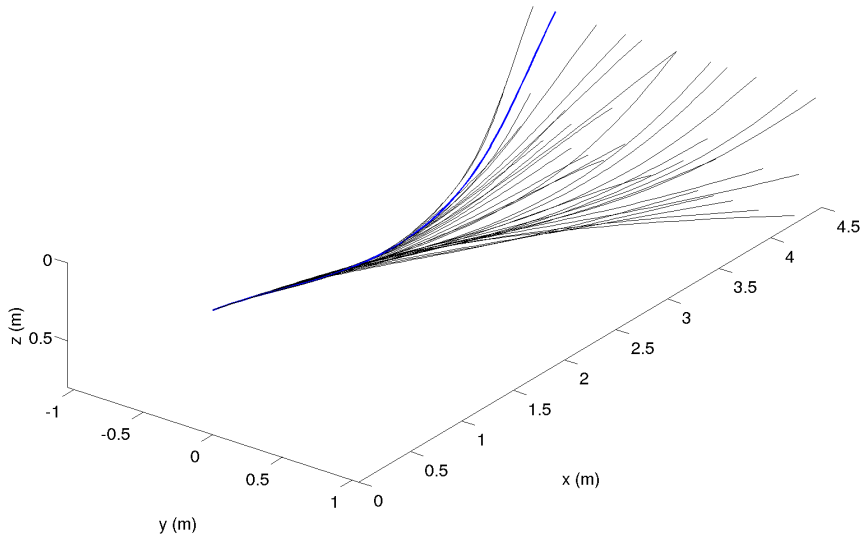


Figure 8-4: Trajectory library for the SBach airplane. The library consists of 40 different maneuvers. The maneuver from Chapter 8.4 is highlighted in blue.

For each trajectory in our library, we computed a TVLQR controller with the same state/action costs as the controller for the maneuver in Chapter 8.4. As in Chapter 8.4, we used SOS programming to compute funnels for each trajectory in the library. We note that since the dynamics of the airplane are symmetric with respect to reflection about the x axis, we were able to halve the amount of computation involved in constructing the trajectory/-funnel library by exploiting this symmetry.

As mentioned in Chapter 8.2, the positions and geometry of the obstacles are not reported to the planner until the airplane has cleared the launcher. This forces planning decisions to be made in real-time. We use the planning algorithm described in Chapter 6 to choose a funnel from our library. The planner employs the QCQP-based algorithm from Chapter 6.1 for shifting funnels in the $x-y-z$ directions. As in the other examples considered in this thesis, we use the FORCES Pro solver [35] for our QCQP problems. If no satisfactory

funnel is found by the planner, we revert to a "failsafe" option which involves switching off the propellor and gliding to a halt. A more sophisticated failsafe would be to attempt to transition to a "propellor-hang" mode. However, we found that our failsafe provides adequate protection to the airplane's hardware as it usually glides on to the safety net before colliding with any obstacles. Finally, we note that since the experimental arena is quite limited in space, we do not replan funnels once one has been chosen; the airplane executes the feedback controller corresponding to the chosen funnel for the whole duration of the flight.

We tested our approach on 15 different obstacle environments of varying difficulty. These environments are shown in Figures 8-5 and 8-6. The obstacles include poles of different lengths in varying orientations and also a hoop of diameter equal to 0.9 m (as a reference the airplane's wingspan is 0.44 m, thus only leaving 0.23 m of margin on either side of the airplane assuming it passes through the center of the hoop). We model the poles as cuboids with heights equal to that of the poles and widths equal to the diameter. The hoop is approximated with eight polytopic segments (see Figure 8-8).

Figure 8-7 presents a more quantitative perspective on the obstacle environments. Here we we have plotted a histogram of the gaps between obstacles in the environment and compared it to the wingspan of the airplane. In particular, for each obstacle in a given environment we consider the distance[1] to the closest obstacle that is at least 5 cm away (the 5 cm threshold is chosen to prevent obstacles that are right next to each other being considered as having a small gap. For example, the obstacle closest to one of the horizontal poles in the first environment in Figure 8-5 should be the other horizontal pole and not the adjacent vertical pole). As the histogram illustrates, a significant fraction (approximately 35%) of the gaps are less than the airplane wingspan and about 66% of the gaps are less than two wingspans. Of course, it is worth noting that not all of the gaps greater than the wingspan are in fact negotiable (e.g., obstacles placed far apart along the x direction).

A video of the airplane traversing a few representative environments is available online at https://youtu.be/cESFpLgSb50. Out of the 15 environments the airplane was able to

[1]Here, the distance between obstacles is defined in the usual way for sets. In particular, for any pair of obstacles O_1 and O_2, we define the distance between them as $\min_{x_1 \in O_1, x_2 \in O_2} \|x_1 - x_2\|_2$.

Figure 8-5: Environments 1-8 on which the online planning algorithm was tested. The obstacles include poles of different lengths in varying orientations and also a hoop of diameter equal to 0.9 m.

Figure 8-6: Environments 9-15 on which we tested our planning algorithm. The bottom most image shows the only failure case. Here the airplane brushed one of the obstacles on its way across the room.

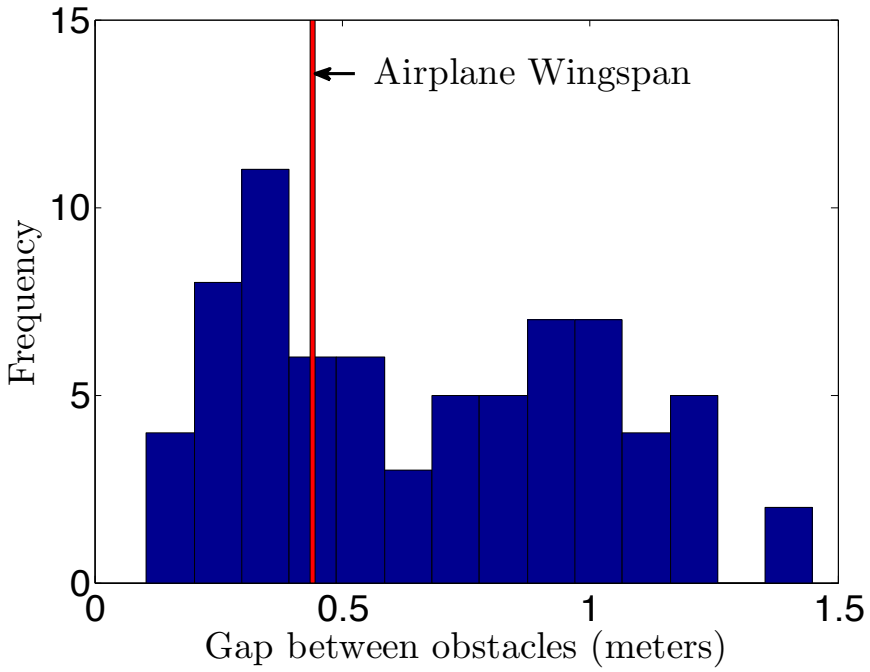


Figure 8-7: Histogram of gaps between objects in our test environments compared with the airplane's wingspan (0.44 m). As the plot shows, a significant number of the gaps between obstacles are less than then wingspan.

successfully negotiate 14 of them, thus demonstrating the efficacy of our real-time planner on this challenging task.

Figure 8-8 presents the output of our real-time planner on four of the more challenging environments. In particular we plot the funnel chosen by the planner (which have been shifted using the QCQP-based algorithm presented in Chapter 6.1) alongside an image sequence showing the airplane executing the plan. We note that the increased expressivity afforded to us by the ability to shift funnels in the cyclic coordinates has a large impact on the planner being able to find collision free funnels. Perhaps the best example of this is the environment which contains the hoop (top row of Figure 8-8). Without the ability to make small adjustments to the funnels, the chances of finding a collision free funnel are extremely low (we would require a funnel that passes almost exactly through the center of the hoop). Figure 8-9 compares the output of the planner with and without applying the QCQP-based algorithm for shifting funnels. As we can observe, the best funnel found in the former case

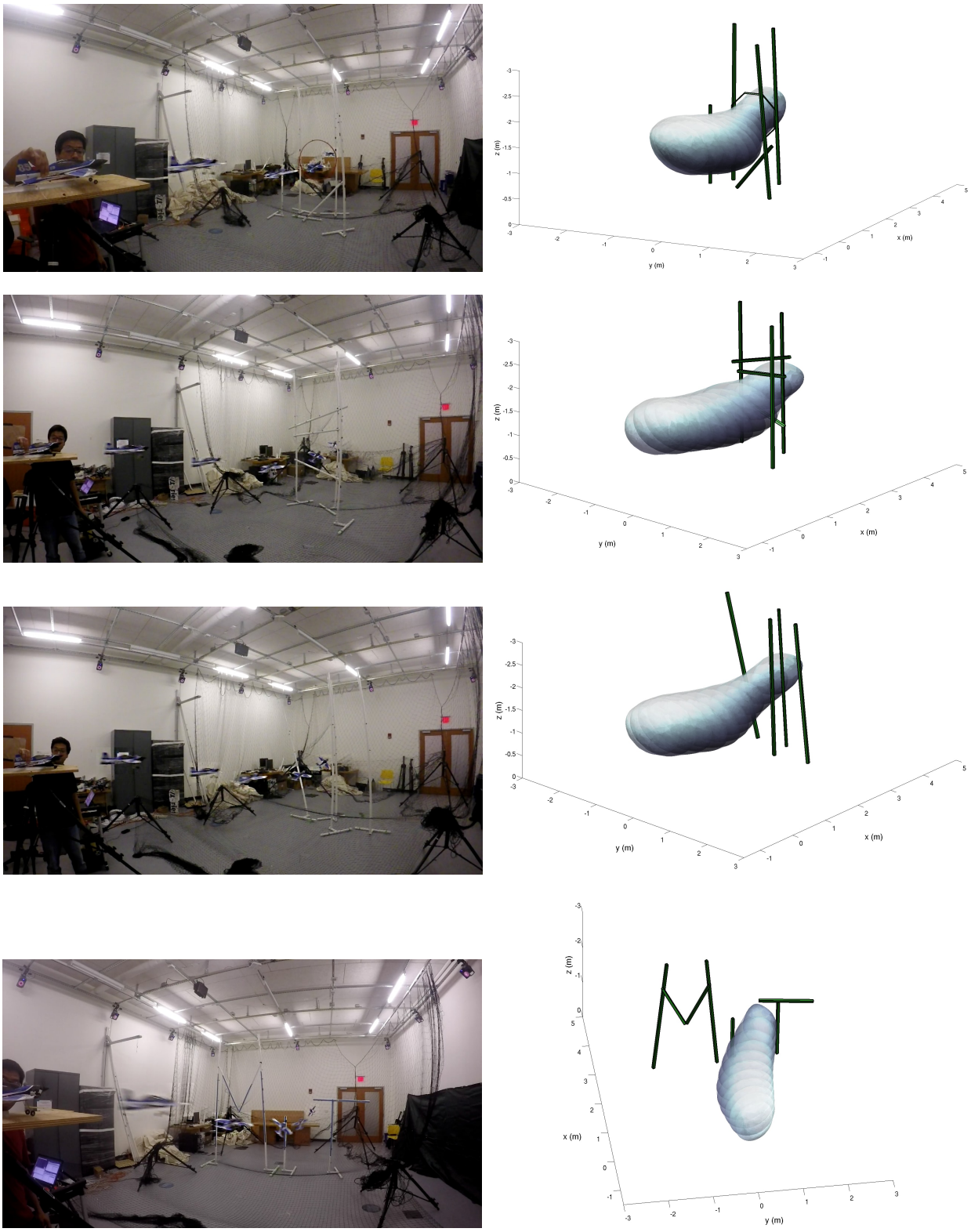


Figure 8-8: This figure depicts the planned funnel along with the polygonal obstacle representations for four of our fifteen test environments. Note that the funnels have been inflated to take into account the collision geometry of the airplane (modeled as a sphere of diameter equal to the wingspan).

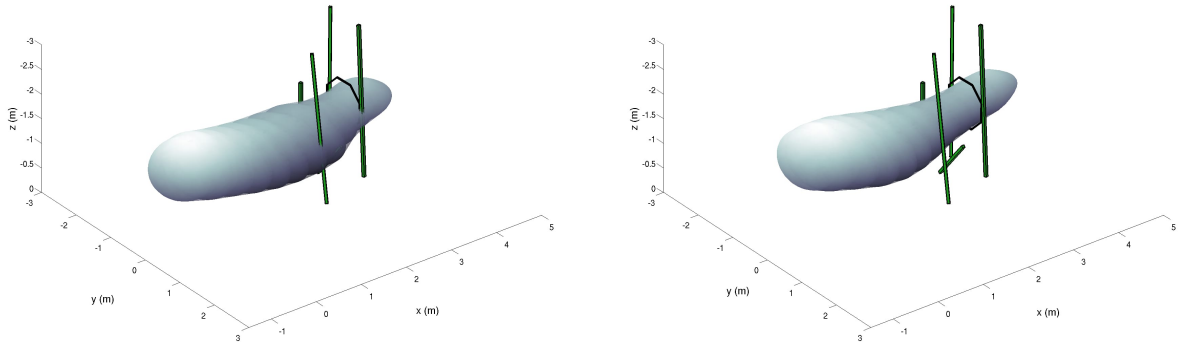


Figure 8-9: The QCQP-based algorithm for shifting funnels (Chapter 6.1) plays a crucial role in the planner's ability to find collision free funnels. Here we compare the output of the planner with and without applying the QCQP-based algorithm for shifting funnels. The best funnel found in the former case is well in collision with the obstacles while in the latter case the planner is able to find a collision free funnel. Note that the funnels chosen in the different cases correspond to different maneuvers.

is well in collision with the obstacles while in the latter case the planner is able to find a collision free funnel. This illustrates the crucial role that exploiting invariances plays in the success of the planner on this task.

As mentioned before, the airplane was able to successfully negotiate 14 out of our 15 environments. The single failure case occurred on the environment shown at the bottom of Figure 8-6, where the airplane clipped one of the poles close to the end of its flight. This failure can be attributed to the fact that the planner chose to execute one of a small handful of maneuvers that are more aggressive than the funnel we validated in Chapter 8.4. The controller saturates the control inputs significantly on this maneuver and hence violates our assumption about input limits not being reached. Hence this funnel is not entirely valid on the hardware system. While we could have taken actuator saturations into account while computing the funnels (see Chapter 4.3.3), we chose not to do so in order to reduce the computation time. In hindsight, a more careful treatment of the system would have included saturations when computing funnels.

Finally, we note that on a small number of occasions during our experiments we observed the "failsafe" option being employed by the planner (i.e., switching off the throttle and gliding to a halt). This was typically caused by failures in the launching mechanism such as the airplane making contact with the operator's hand on its way out of the launcher. Due

to the altered initial conditions in these cases the airplane is not in the inlet of most or all of the funnels and thus has to resort to the failsafe. On these occasions we simply repeated the experiment to obtain a successful flight.

8.6 Implementation details

We end this chapter by mentioning a few implementation details that were important in achieving the results presented above. First, while the Vicon motion tracking system provides accurate position and orientation estimates at 120 Hz, a finite difference of these measurements can lead to noisy estimates of the derivative states. For our experiments we filtered the finite differences using a simple Luenberger observer [64].

Second, we observed a delay of approximately 55 ms in our closed-loop hardware system. While there are several ways to explicitly take this into account by adding delay states to our airplane model, we accommodate for the delay during execution by simulating our model forwards by 55 ms from the estimated current state and using this simulated future state to compute the current control input. This simple strategy is a common one and has previously been found to be effective in a wide range of applications [47, 75, 94].

Chapter 9

Addressing the challenge of scalability with DSOS and SDSOS programming

The main computational tool we have employed throughout this thesis has been sums-of-squares (SOS) programming. In general, SOS programming has had a large impact on the control theory community since its advent over a decade ago [80] and has been used to tackle a wide variety of problems including feedback control synthesis, safety verification and computation of regions of attraction, invariant sets, and reachable sets for a broad class of nonlinear and hybrid systems [105, 49, 87, 28, 2, 46]. However, despite the wide acceptance of the SOS approach in the control and optimization communities, applications of the method considered in the literature typically involve systems of relatively modest dimension (approximately 5-10 states). In fact, to our knowledge, our use of SOS programming for computing funnels for the quadrotor (Chapter 7.2) and fixed-wing airplane (Chapter 8) systems are among the largest-scale control applications of this nature.

Thus, scalability of SOS programming is one of the key challenges that need to be addressed in order to apply the funnel library-based approach to more complex robotic systems such as humanoids. More generally, the limits on scalability of SOS programming have made its application to network control, power systems, multi-agent systems, and complex robotic systems extremely challenging. While control systems of higher dimension have been ad-

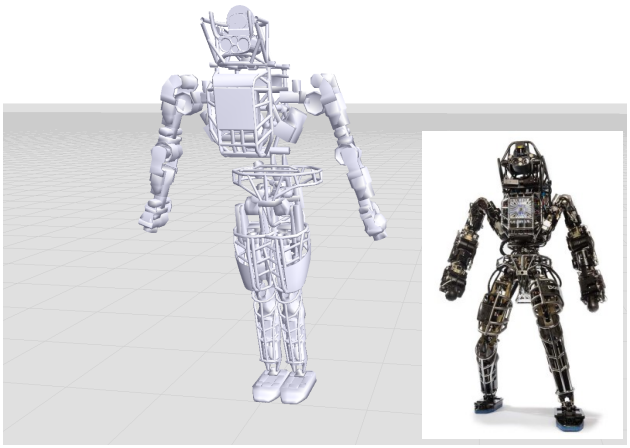


Figure 9-1: The methods presented in this chapter allow us to handle problems of dimensionality well beyond the reach of current SOS programming based approaches. For example, in Chapter 9.4.4, we design a balancing controller for a 30 state and 14 control input model of the ATLAS humanoid robot. A visualization of the model is shown in this figure, along with the hardware platform (inset) on which the parameters of the system are based. (Picture of robot reproduced with permission from Boston Dynamics.)

dressed using SOS programming in certain cases, they involve exploiting special structure (e.g., symmetry, sparsity) of the particular problem under consideration [81, 11, 10, 110]. For many real-world control applications, we would like to be able to handle problems of high dimensionality even when such structure is limited or not available. Further, even for smaller problems, being able to obtain an answer much more quickly than we currently can (perhaps at the cost of conservatism) would be of significant practical utility.

The limited scalability of the SOS approach is due in large part to the fact that, in general, SDPs are among the most expensive convex relaxations. At the current state of solver technology, it is not uncommon for the more practically-oriented user to want to avoid SDP-based approaches. For example, in the industry-motivated field of integer programming, the cutting-plane approaches used on real-life problems are almost exclusively based on linear programming (LP) or second order cone programming (SOCP) [77, 50]. Even though semidefinite cuts are known to be stronger, they are too expensive to be used even at the root node of branch-and-bound techniques for integer programming. In the field of SOS optimization, however, a sound alternative to SOS programming that can avoid SDP and take advantage of the existing mature and high-performance LP/SOCP solvers is lacking. This is precisely what we are after here.

In very recent work [4], Ahmadi and Majumdar provide new sufficient conditions for polynomial nonnegativity based on linear programming (LP) and second-order cone programming (SOCP) relaxations. The key insight is to replace the positive semidefiniteness constraint on the Gram matrix in the SOS approach with stronger conditions: *diagonal dominance* and *scaled diagonal dominance*. Equivalently, the set of polynomials that is being searched over is restricted to a subclass of sums-of-squares polynomials: the *Diagonally-Dominant SOS* (DSOS) and *Scaled-Diagonally-Dominant SOS* (SDSOS) polynomials (see Definition 8). This results in significantly cheaper optimization problems, though with more conservative solutions in general.

By leveraging the scalability of available LP/SOCP solvers, we demonstrate the efficacy of this approach on several high-dimensional problems that are currently well beyond the reach of SOS programming: computing a region of attraction for a 22 dimensional system, network analysis for an oscillator network with 50 nodes, searching for degree 3 controllers and degree 8 Lyapunov functions for an Acrobot system (with the resulting controller validated on a hardware platform), and synthesizing a balancing controller for a 30 state and 14 input model of the ATLAS humanoid robot shown in Figure 9-1. This last example may be viewed as a first step towards applying the funnel library approach presented in this thesis to a system of the complexity of a humanoid robot. We also present numerical experiments on smaller instances of our problems (where SOS techniques can be implemented) in order to demonstrate that the additional conservatism introduced by our methods can be small compared to SOS approaches.

This work also gives rise to several interesting theoretical questions related to the "gap" between (S)DSOS and nonnegative polynomials and how this gap may be reduced and even closed with the help of multiplier polynomials. This is analogous to the classical Positivstellensatz theorems for the gap between nonnegative polynomials and SOS polynomials. Since our goal in this chapter is to provide a clear practical exposition of the approach along with its control applications, only a highlight of a few theorems in this direction is presented in Chapter 9.3 and a more comprehensive treatment is to be found in [4].

9.1 Relevant Work

There have been many contributions to improvements in scalability of SOS programming. One approach has been to develop systematic techniques for taking advantage of problem structure, such as sparsity or symmetry of the underlying polynomials, to reduce the size of the SDPs. Examples include applications to network problems where connectivity information is known *a priori* [45], dynamical systems that can be decomposed and analyzed using smaller subsystems [10], and analysis of delayed linear systems with a low-rank delay coefficient matrix [110].

Another approach which also holds promise has been to design customized solvers for special classes of SDPs and avoid resorting to off-the-shelf interior point solvers. Examples in this direction include the work in [52], which proposes a method for solving large scale robust stability problems in a parallel computing environment with a customized interior point solver, and in [101], which offers a customized interior point algorithm for optimizing over the set of nonnegative trigonometric polynomials.

The approach we take in this chapter for enhancing scalability is orthogonal to the ones mentioned above (and can potentially later be combined with them). We propose to not work with the SOS decomposition to begin with, but employ computationally cheaper sufficient conditions for polynomial nonnegativity that are perhaps stronger than a SOS decompostion, but still provide useful solutions for various control applications.

A previous approach that has a similar spirit is the work in [40], which tackles the specific task of finding a lower bound on the minimum of a polynomial using geometric programming (GP). However, these GP-based conditions seem to be too strong and we show in [4] that this method is always outperformed by the SDSOS approach.

9.2 Diagonal Dominance and Scaled Diagonal Dominance

Recall from Chapter 3.2 that a polynomial $p(x)$ is a *sum-of-squares* (SOS) if and only if there exists a positive semidefinite symmetric matrix Q that satisfies

$$p(x) = v(x)^T Q v(x). \tag{9.1}$$

The vector $v(x)$ here is the vector of all monomials that have degree less than or equal to half the degree of p. As discussed in Chapter 3.2, the search for a matrix Q satisfying the linear constraints coming from (9.1) can be cast as a semidefinite program and solved using a variety of techniques (e.g. interior point methods).

In this chapter, we will denote the cone of SOS polynomials with degree d in n variables by $SOS_{n,d}$. Denoting the cone of nonnegative polynomials in n variables and degree d by $POS_{n,d}$, it is clear that $SOS_{n,d} \subseteq POS_{n,d}$. The key insight we will exploit here is to replace the condition that the symmetric matrix Q is positive semidefinite (psd) with stronger sufficient conditions in order to obtain inner approximations to the cone $SOS_{n,d}$. In particular, we will require Q to be either *diagonally dominant* (dd) or *scaled diagonally dominant* (sdd). We recall these definitions below.

Definition 6. *A symmetric matrix A is* diagonally dominant *(dd) if $a_{ii} \geq \sum_{j \neq i} |a_{ij}|$ for all i.*

We will refer to the set of $n \times n$ dd matrices as DD_n.

Remark 5. *It is clear from Definition 6 that the set DD_n has a polytopic description and can thus be optimized over using LP.*

Definition 7. *A symmetric matrix A is* scaled diagonally dominant *(sdd) if there exists an element-wise positive vector y such that:*

$$a_{ii} y_i \geq \sum_{j \neq i} |a_{ij}| y_j, \forall i.$$

Equivalently, A is sdd if there exists a positive diagonal matrix D such that AD (or equivalently, DAD) is dd.

The set of $n \times n$ sdd matrices will be denoted by SDD_n. We note that sdd matrices are sometimes referred to as *generalized diagonally dominant* matrices [20].

Remark 6. *The fact that diagonal dominance is a sufficient condition for positive semidefiniteness follows directly from Gershgorin's circle theorem. This also implies that sdd matrices are psd since the eigenvalues of DAD have the same sign as those of A when D is a diagonal matrix with positive entries. Hence, denoting the set of $n \times n$ symmetric positive semidefinite matrices (psd) as S_n^+, we have from the definitions above that:*

$$DD_n \subseteq SDD_n \subseteq S_n^+.$$

The next theorem, which is proved in [4], provides an important characterization of the set SDD_n.

Theorem 1 ([4])**.** *Denote the set of $n \times n$ symmetric matrices as S^n. Let $M_{2\times 2}^{ij} \in S^n$ denote the symmetric matrix with all entries zero except the elements $M_{ii}, M_{ij}, M_{ji}, M_{jj}$. Then, we have the following description of SDD_n:*

$$SDD_n = \left\{ A \in S^n : A = \sum_{i,j \neq i}^{i=n} M_{2\times 2}^{ij}, \begin{bmatrix} M_{ii} & M_{ij} \\ M_{ji} & M_{jj} \end{bmatrix} \succeq 0 \right\}.$$

Theorem 1 provides us a method to optimize over the set SDD_n using second order cone programming (SOCP), as the following theorem demonstrates.

Theorem 2. *The set of matrices SDD_n can be optimized over using second order cone programming (SOCP).*

Proof. Positive semidefiniteness of the 2×2 matrices in Theorem 1 is equivalent to the diagonal elements M_{ii}, M_{jj}, along with the determinant $M_{ii}M_{jj} - M_{ij}^2$, being nonnegative. This is a *rotated quadratic cone* constraint and can be imposed using SOCP [6]. □

9.3 DSOS and SDSOS Polynomials

We now introduce naturally motivated cones that are inner approximations of $POS_{n,d}$ and that lend themselves to LP and SOCP.

Definition 8 ([4])**.**

- *A polynomial p is* diagonally-dominant-sum-of-squares *(dsos) if it can be written as*

$$p = \sum_i \alpha_i m_i^2 + \sum_{i,j} \beta_{ij}^+ (m_i + m_j)^2 + \beta_{ij}^- (m_i - m_j)^2,$$

  for some monomials m_i, m_j and some constants $\alpha_i, \beta_{ij}^+, \beta_{ij}^- \geq 0$.

- *A polynomial p is* scaled-diagonally-dominant-sum-of-squares *(sdsos) if it can be written as*

$$p = \sum_i \alpha_i m_i^2 + \sum_{i,j} (\beta_i^+ m_i + \gamma_j^+ m_j)^2 + (\beta_i^- m_i - \gamma_j^- m_j)^2,$$

  for some monomials m_i, m_j and some constants $\alpha_i, \beta_i^+, \gamma_j^+, \beta_i^-, \gamma_j^- \geq 0$.

We denote the set of polynomials in n variables and degree d that are dsos and sdsos by $DSOS_{n,d}$ and $SDSOS_{n,d}$ respectively. The following inclusions are straightforward:

$$DSOS_{n,d} \subseteq SDSOS_{n,d} \subseteq SOS_{n,d} \subseteq POS_{n,d}.$$

Our terminology in Definition 8 comes from the following relationship between dsos and sdsos polynomials to the cones of dd and sdd matrices introduced in Chapter 3.

Theorem 3 ([4])**.**

- *A polynomial p of degree $2d$ is dsos if and only if it admits a representation as $p(x) = z^T(x)Qz(x)$, where $z(x)$ is the standard monomial vector of degree d, and Q is a dd matrix.*

- *A polynomial p of degree $2d$ is sdsos if and only if it admits a representation as $p(x) = z^T(x)Qz(x)$, where $z(x)$ is the standard monomial vector of degree d, and Q is a sdd matrix.*

Theorem 4. *The set $DSOS_{n,d}$ is polyhedral and the set $SDSOS_{n,d}$ has a second order cone representation. For any fixed d, optimization over $DSOS_{n,d}$ (resp. $SDSOS_{n,d}$) can be done with linear programming (resp. second order cone programming), of size polynomial in n.*

Proof. This follows directly from Remark 5 and Theorem 2, along with Theorem 3. The size of these programs is polynomial in n since the size of the Gram matrix is $\binom{n+d}{d} \times \binom{n+d}{d}$, which scales as n^d. □

We will refer to optimization problems with a linear objective posed over the cones $DSOS_{n,d}$ and $SDSOS_{n,d}$ as DSOS programs and SDSOS programs respectively.

9.3.1 Asymptotic Guarantees

Next, we briefly discuss how the "gap" between the cones $DSOS_{n,d}$, $SDSOS_{n,d}$ and $POS_{n,d}$ can be reduced and in some cases closed. In particular, we consider the use of "multipliers" similar to the Positivstellensatz multipliers employed in SOS programming.

Definition 9.

- *For a positive integer r, a polynomial p is* r-diagonally-dominant-sum-of-squares *(r-dsos) if the polynomial*
$$\left(\sum_i x_i^2\right)^r p$$
is dsos.
- *For a positive integer r, a polynomial p is* r-scaled-diagonally-dominant-sum-of-squares *(r-sdsos) if the polynomial*
$$\left(\sum_i x_i^2\right)^r p$$
is sdsos.

We denote the set of polynomials in n variables and degree d that are r-dsos and r-sdsos by r-$DSOS_{n,d}$ and r-$SDSOS_{n,d}$, respectively.

Note that the sets r-$DSOS_{n,d}$ and r-$SDSOS_{n,d}$ can also be optimized over using LP and SOCP respectively. The purpose of the multiplier $(\sum x_i^2)^r$ is to have a knob for trading off speed with accuracy of approximation. By increasing r, we obtain increasingly accurate inner approximations to the set of nonnegative polynomials. The following example shows that the LPs obtained from even small r can outperform the semidefinite programs resulting from SOS.

Example 1. *Consider the polynomial,* $p(x) = x_1^4x_2^2 + x_2^4x_3^2 + x_3^4x_1^2 - 3x_1^2x_2^2x_3^2$. *This polynomial is nonnegative but* not *a sum-of-squares [89]. However, there is an LP-based nonnegativity certificate since one can show that* $p \in$ 1-$DSOS$. *Hence,* 1-$DSOS \nsubseteq SOS$.

The following two theorems provide asymptotic guarantees on r-dsos (and hence r-sdsos) hierarchies. Their proofs rely on Positivstellensatz results from real algebraic geometry.

Theorem 5 ([4])**.** *Let p be an even form (i.e., a form where individual variables are raised to even degrees), with $p(x) > 0$ for all $x \neq 0$, then there exists an integer r such that $p \in$ r-$DSOS$.*

If we allow the use of general multipliers (in contrast to $\sum x_i^2$), we can relax the assumption of evenness.

Theorem 6 ([4])**.** *For any positive definite form p, there exists a form q such that q and pq are both dsos.*

Note that given p, the search for such a q (of a given degree) is a LP. Moreover, a feasible solution to this LP certifies nonnegativity of p.

9.4 Examples

This section demonstrates the scalability of the DSOS and SDSOS approach on four examples relevant to control and verification of dynamical systems. We compare runtimes and optimality of our approach with the SOS approach in cases where this is possible (i.e.,

smaller instances of problems). A software package written using the Systems Polynomial Optimization Toolbox (SPOTless) [70] includes a complete implementation of the presented methods and is available online[1]. The toolbox features very efficient polynomial algebra and allows us to setup the large-scale LPs and SOCPs arising from our examples.

We use MOSEK as our LP and SOCP solver. Runtimes for SDPs are reported for both the recently released MOSEK SDP solver and the very widely used SeDuMi solver. All code was run on a machine with four Intel i7 processors with a clock speed of 3.4 GHz and 16 GB RAM.

9.4.1 Regions of Attraction

In our first example, we consider the computation of regions of attraction (ROA), which is known to be a NP-hard problem [5]. The system we examine is the N-link pendulum depicted in Figure 9-2. This system has $2N$ states $x = [\theta_1, \ldots, \theta_N, \dot{\theta}_1, \ldots, \dot{\theta}_N]$ composed of the joint angles and their derivatives. There are $N-1$ control inputs (the joint closest to the base is not actuated). Each link of the pendulum is assumed to be a uniformly dense cylindrical rod of radius 5 cm with mass $m = 1$ kg and length $l = 1$ m. We take the unstable "upright" position of the system to be the origin of our state space and design a LQR controller in order to stabilize this equilibrium. The cost matrix Q and the action matrix R for the LQR controller are both diagonal, with $Q_{ii} = 10$ for $i = 1, \ldots, N$, $Q_{jj} = 1$ for $j = N+1, \ldots, 2N$, and $R_{ii} = 1$ for $i = 1, \ldots, N-1$.

2N (# states)	4	6	8	10	12	14	16	18	20	22
DSOS	< 1	0.44	2.04	3.08	9.67	25.1	74.2	200.5	492.0	823.2
SDSOS	< 1	0.72	6.72	7.78	25.9	92.4	189.0	424.74	846.9	1275.6
SOS (SeDuMi)	< 1	3.97	156.9	1697.5	23676.5	∞	∞	∞	∞	∞
SOS (MOSEK)	< 1	0.84	16.2	149.1	1526.5	∞	∞	∞	∞	∞

Table 9.1: Runtime comparisons (in seconds) for ROA computations on N-link system.

[1]Link to `SPOTless` software package:

https://github.com/spot-toolbox/spotless

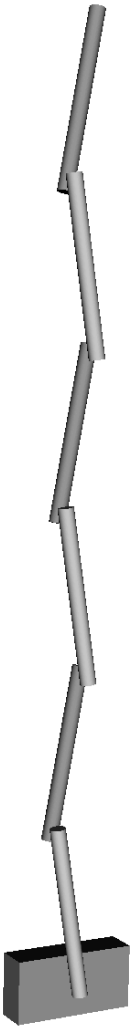
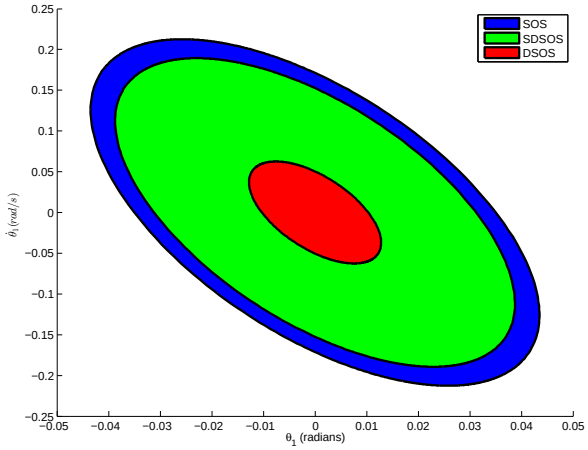
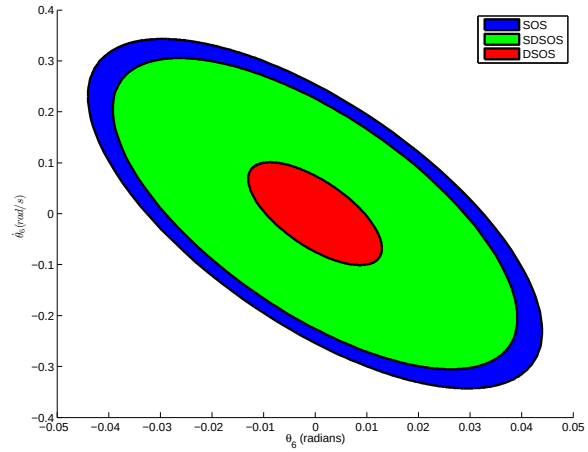


Figure 9-2: An illustration of the N-link pendulum system (with N = 6).



(a) θ_1-$\dot{\theta}_1$ subspace.



(b) θ_6-$\dot{\theta}_6$ subspace.

Figure 9-3: Comparisons of projections of the ROAs computed for the 6-link pendulum system using DSOS, SDSOS and SOS programming.

A polynomial approximation of the dynamics of the closed loop system is obtained by a Taylor expansion of degree 3. Denoting the resulting dynamics by $\dot{x} = f(x)$, if we can find a positive definite polynomial $V(x)$ such that the following condition holds:

$$V(x) \leq \rho \implies \dot{V}(x) < 0, \tag{9.2}$$

then the ρ sublevel set of $V(x)$ is an inner approximation of the ROA. We choose our Lyapunov function to be the cost-to-go function $V(x) = x^T S x$ of the LQR controller and attempt to maximize ρ. As described in [80], under the assumption that the Hessian of $\dot{V}(x)$ is positive definite at the origin, the following is a sufficient condition for (9.2):

$$(x^T x)(V(x) - \rho) + L(x)\dot{V}(x) \geq 0. \tag{9.3}$$

Here, $L(x)$ is a "multiplier" polynomial. By replacing the nonnegativity condition in (9.3) with a DSOS constraint, we obtain a LP:

$$\begin{aligned} \max_{\rho, L(x)} \quad & \rho \\ \text{s.t.} \quad & (x^T x)(V(x) - \rho) + L(x)\dot{V}(x) \in DSOS_{2N,6} \end{aligned} \tag{9.4}$$

Similarly, a SDSOS/SOS constraint yields a SOCP/SDP. The optimization in (9.4) is over ρ and $L(x)$ (degree 2).

We note that while there are different SOS programming based formulations for approximating the ROA that allow one to search over Lyapunov functions in addition to the scaling ρ (see, e.g., [105]), these typically lead to non-convex optimization problems. Algorithms for these formulations generally do not have convergence guarantees, thus making it difficult to distinguish between conservatism caused by the algorithm and conservatism inherent in the (S)DSOS condition. Hence, in order to facilitate a direct comparison of the DSOS, SDSOS and SOS approaches, we use the formulation presented above in (9.3), which involves solving a single convex optimization problem for each approach.

An important observation is that unlike the sets $POS_{n,d}$ and $SOS_{n,d}$, the sets $DSOS_{n,d}$ and $SDSOS_{n,d}$ are not invariant to affine coordinate transformations, i.e., a polynomial $p(A(x))$ is not necessarily DSOS (resp. SDSOS) even if $p(x)$ is DSOS (resp. SDSOS) and A is an affine transformation. Thus, performing coordinate transformations on the problem data (e.g., on the state variables of a dynamical system) can sometimes have an important effect. We explore this issue in this example by performing a preprocessing step that is intuitive and straightforward to implement. It can be used for problems involving the search for Lyapunov functions, and can potentially be extended to other problems as well. In particular, we find an invertible affine transformation that simultaneously diagonalizes the Hessians of $V(x)$ and $-\dot{V}(x)$ evaluated at the origin (this is always possible for two positive definite matrices). The intuition behind the coordinate change is that the functions $V(x)$ and $-\dot{V}(x)$ locally resemble functions of the form $x^T Dx$ (with D diagonal), which are DSOS polynomials that are "far away" from the boundary of the DSOS cone. We solve the optimization problem (9.3) after performing this coordinate transformation. The transformation is then be inverted to obtain ROAs in the original coordinate frame.

Table 9.1 compares the runtimes of the programs obtained using our approach with SOS programming for different values of $2N$ (number of states). The SOS programs obtained for $2N > 12$ are too large to run (due to memory constraints). In contrast, our approach allows us to handle almost twice as many states. Further, for cases where the SOS programs do run, the DSOS and SDSOS programs are significantly faster. In particular for the 12 state system, DSOS is approximately 2500 times faster than SOS using SeDuMi and 150 times faster than SOS using MOSEK, while SDSOS is 900 times faster in comparison to SeDuMi and 60 times faster than MOSEK for SOS.

Table 9.2 compares the optimal values of ρ obtained using the different methods. The values obtained using SDSOS programming are within approximately 80 to 90 percent of the values from SOS programming. Note that since the Lyapunov function is fixed in our case and we are only optimizing ρ, the ratio of optimal values of ρ is equal to the ratio of $(\text{Volume})^{\frac{1}{2N}}$ of the ROAs. While the result using DSOS are more conservative, they could

2N (states)	4	6	8	10	12
ρ_{dsos}/ρ_{sos}	0.38	0.45	0.13	0.12	0.09
ρ_{sdsos}/ρ_{sos}	0.88	0.84	0.81	0.79	0.79

Table 9.2: Comparison of optimal values for ROA problem on N-link pendulum.

still be useful in practice. Figure 9-3 compares projections of the ROAs obtained using the different methods for the 6-link pendulum.

9.4.2 Network Analysis

In this example, we consider Example 3 from [45]. The goal is to analyze the domain on which the Hamiltonian function V for a network of Duffing oscillators is positive definite:

$$V = \sum_{i=1}^{N} a_i(\frac{1}{2}y_i^2 - \frac{1}{4}y_i^4) + \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N} b_{ik}\frac{1}{4}(y_k - y_i)^4. \tag{9.5}$$

Here, N is the number of nodes in the network. The following condition can be used to establish an inner approximation of the domain on which V is positive definite:

$$p(y_1, \ldots, y_N) = V - \sum_{i=1}^{N} \lambda_i y_i^2(g - y_i^2) \geq 0, \tag{9.6}$$

where $\lambda_i > 0$ are scalar multipliers. These conditions imply that V is positive definite when $y_i^2 < g$.

In [45], the authors propose an approach that exploits network structure to eliminate monomials from the SOS program that results from replacing the inequality in (9.6) with a SOS condition. This makes the SOS program smaller (potentially at the cost of conservatism) and allows them to handle large network sizes.

By replacing the nonnegativity in (9.6) by a (S)DSOS condition, we can apply the techniques presented in this chapter. For the comparisons presented below, we do not preprocess the programs using the method presented in [45]. We demonstrate that we are able to handle network sizes considered in [45] *without* explicitly exploiting network structure. Of course,

N (nodes)	10	20	30	40	50
DSOS	0.24	0.76	2.86	8.74	21.35
SDSOS	0.26	1.07	5.02	17.81	46.64
SOS (SeDuMi)	3.91	16050.26	∞	∞	∞
SOS (MOSEK)	0.53	173.75	∞	∞	∞

Table 9.3: Runtimes (in seconds) for network problem.

N (nodes)	10	20	30	40	50
DSOS	0.94	0.74	0.71	0.69	0.66
SDSOS	1.58	1.49	1.48	1.48	1.48
SOS (SeDuMi)	2.00	2.00	NA	NA	NA
SOS (MOSEK)	2.00	2.00	NA	NA	NA

Table 9.4: Optimal values (g) for network problem.

we would expect to scale even better by using the approach in [45] as a preprocessing step.

As in [45], we randomly set $\frac{0.5}{N} \leq b_{ik} \leq \frac{1.5}{N}$, corresponding to a globally coupled network of Duffing oscillators. We also find the largest value of g for which the condition (9.6) is feasible by performing a bisection search in an outer loop. These results and runtimes are compared for (S)DSOS and SOS programming in Tables 9.4 and 9.3 respectively.

We find that our approach works with runtimes comparable to [45], even though we do not exploit the network structure. In contrast, the SDP solvers MOSEK and SeDuMi are significantly slower even for networks with $N = 20$. For N larger than 25, memory (RAM) constraints prevent the SDP from running.

In [45], the value of g is 1.8 for all network sizes. Thus, the optimal values we obtain using SDSOS programming are only slightly more conservative.

9.4.3 Hardware Experiments on Acrobot

The utility of our method is not restricted to high-dimensional systems. As we show in this example, we can design high degree polynomial feedback controllers using high degree Lyapunov functions for smaller systems with benefits in terms of running time as compared to SOS programming. We consider the Acrobot [96], which is a benchmark for control of

Figure 9-4: Picture of the Acrobot balancing in the upright configuration using the controller designed with SDSOS programming. A video of the controller in action is available online at https://www.youtube.com/watch?v=FeCwtvrD76I.

underactuated systems. The system is a special case of the N-link pendulum examined in Chapter 9.4.1 (with $N = 2$) and is actuated only at the joint between the two links of the robot (θ_2). The task is to design a polynomial feedback controller in order to stabilize the system about the unstable "upright" position. The hardware platform on which experiments are conducted is shown in Figure 9-4 balancing in this configuration using the controller designed with SDSOS programming (as described below).

System identification for the hardware platform was performed using the prediction error minimization method in MATLAB's System Identification Toolbox [61] in order to identify parameters of the model presented in [96]. The dynamics were then Taylor expanded around the equilibrium to degree 3 in order to obtain a polynomial vector field.

We use the method presented in [65] to design a balancing controller for the system. In particular, we search for a degree 8 Lyapunov function $V(x)$ and a degree 3 feedback controller $u(x)$ in order to maximize the size of the region of attraction (ROA) of the resulting

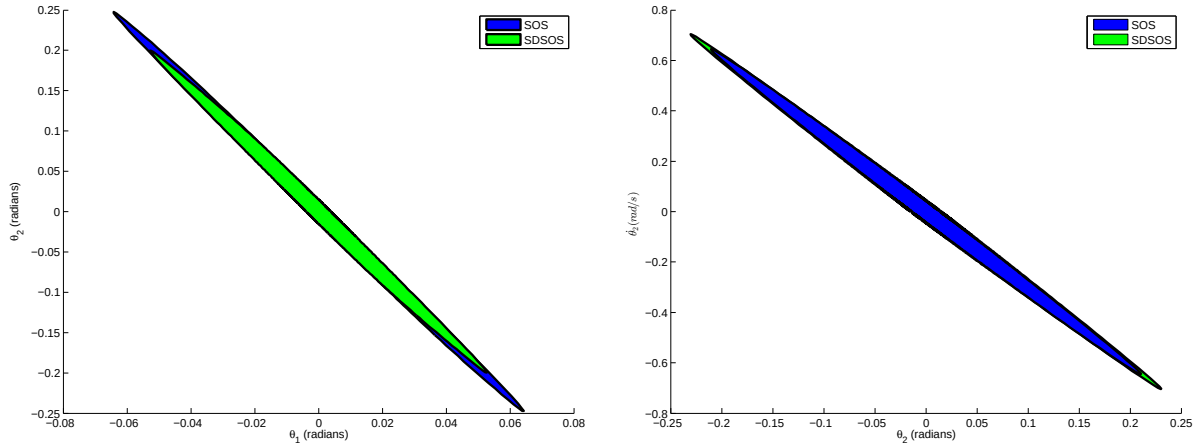
closed-loop system. The DSOS version of the optimization problem is:

$$\begin{aligned} \max_{\rho, L(x), V(x), u(x)} \quad & \rho \\ \text{s.t.} \quad & V(x) \in DSOS_{4,8} \\ & -\dot{V}(x) + L(x)(V(x) - \rho) \in DSOS_{4,10} \\ & L(x) \in DSOS_{4,4} \\ & \sum_j V(e_j) = 1. \end{aligned} \tag{9.7}$$

Here, $L(x)$ is a non-negative multiplier term and e_j is the j-th standard basis vector for the state space $\mathbb{R}^n$. It is easy to see that the above conditions are sufficient for establishing $B_\rho = \{x \in \mathbb{R}^4 \mid V(x) \leq \rho\}$ as an inner estimate of the region of attraction for the system. When $x \in B_\rho$, the second constraint implies that $\dot{V}(x) < 0$ (since $L(x)$ is constrained to be non-negative). The last constraint normalizes $V(x)$.

The optimization problem (9.7) is not convex in general since it involves conditions that are bilinear in the decision variables. This is similar to the funnel computations we encountered in Chapter 4 and can be solved by iteratively optimizing groups of decision variables. Each step in the iteration is then a DSOS program (or a SDSOS/SOS program if the constraints in (9.7) are replaced by DSOS/SOS constraints). This iterative procedure is analogous to the one used in Chapter 4 for computing funnels and is described in more detail in [65]. The procedure can be initialized with the Lyapunov function from a LQR controller and a small enough value of ρ. The LQR Lyapunov function can also be used to perform a coordinate transformation in a manner identical to the one described in Chapter 9.4.1. Finally, we note that in order to use the approach described above on the Acrobot hardware platform, it is also important to take into account the torque limit of the system.The optimization problem (9.7) can be modified in a straightforward manner to account for torque limits as described in [65, Section IV A].

Figure 9-5 presents two-dimensional slices of the ROA resulting from the approach described above using SDSOS and SOS programming (DSOS programming yields very con-



(a) θ_1-θ_2 subspace. (b) θ_2-$\dot{\theta}_2$ subspace.

Figure 9-5: Comparisons of slices of ROAs computed for the Acrobot system using SDSOS and SOS programming.

servative results on this example and we do not present the results here). The ROA from SDSOS programming is only slightly conservative on the $\theta_1 - \theta_2$ slice and is in fact slightly larger than the SOS ROA on the $\theta_2 - \dot{\theta}_2$ slice[2] Each iteration of the algorithm takes approximately 40 seconds with SDSOS, while SOS takes 1825 seconds with SeDuMi and 148 seconds with MOSEK. Convergence of the algorithm is observed between 5 and 10 iterations for SDSOS and SOS. Thus, we obtain significant gains in running times with very little loss in quality of the solution.

We validated the performance of the balancing controller from SDSOS programming by implementing it on the hardware platform shown in Figure 9-4. An open-loop controller that swings up the system from the downright configuration to the upright one was designed using the direction collocation trajectory optimization approach [17]. A time-varying LQR controller was then designed to correct for deviations from this nominal trajectory. At the end of the trajectory, the robot switches to our balancing controller. We performed 30 consecutive experimental trials of the robot swinging up and balancing. The balancing controller had a success rate of 100% on these trials. A video of the controller in action is

[2]Note that the iterative algorithm we employ here is not guaranteed to converge to the global optimum of the problem. Hence, the ROA from SDSOS will not necessarily be a subset of the SOS ROA.

available online at https://www.youtube.com/watch?v=FeCwtvrD76I.

We end this example by noting that attempting to use SOS programming to search for a higher degree controller (e.g., degree 5) resulted in numerical errors from the SDP solvers. This prevented us from running more than 2 or 3 iterations of the algorithm. In contrast, we did not run into such numerical issues with SDSOS programming. This highlights another benefit of our approach; in addition to smaller optimization problems, we also obtain programs that seem to be better conditioned numerically and are easier to work with due to the maturity of existing LP and SOCP solvers.

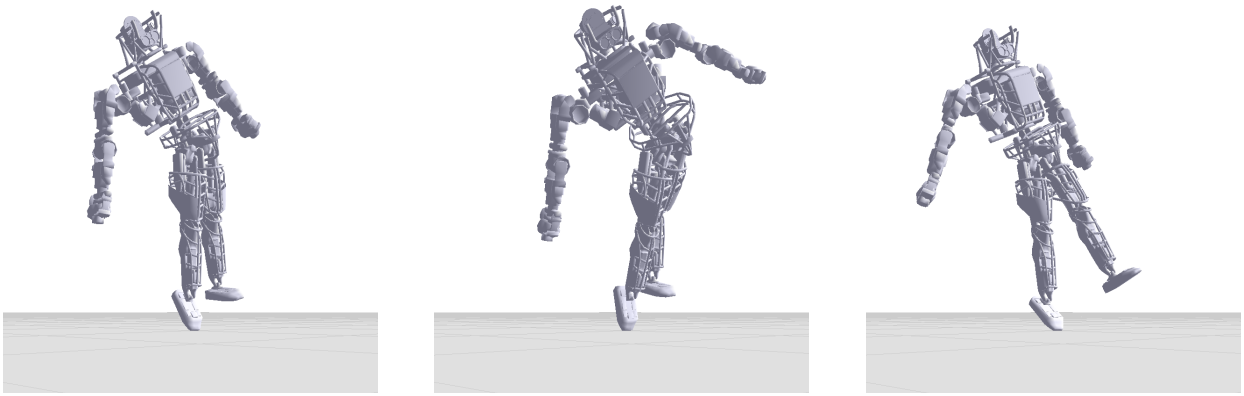
9.4.4 Control Synthesis for Humanoid Robot

In our final example, we apply our approach to synthesize a feedback controller for a humanoid robot. Control of humanoid robots presents a significant challenge due to the nonlinear dynamics of the system and high dimensionality of the state space. Here we use the approach described in this chapter to design a balancing controller for a model of the ATLAS robot shown in Figure 9-1. This may be viewed as a first step towards applying the funnel library approach presented in this thesis to a system of the complexity of a humanoid robot. The robot was designed and built by Boston Dynamics Inc. and was used by a number of teams at the 2015 DARPA Robotics Challenge [55].

Our model of the robot is based on physical parameters of the hardware platform and has 30 states and 14 inputs. The task considered here is to balance the robot on its right toe. In order to do this, we make a few simplifying assumptions. First, we assume that the interface of the right foot and the ground is mediated by a pin joint at the toe. This joint is not actuated and its axis is parallel to the ground, constraining the foot to pitch up and down relative to the ground. Next, we ignore collisions between the different links of the robot. Finally, we do not take into account input saturations.

The balancing controller is constructed using the approach described in Chapter 9.4.3 with SDSOS programming. We Taylor expand the dynamics about the equilibrium to degree 3 in order to obtain polynomial dynamics and search for a linear controller using a degree

2 Lyapunov function. Each iteration takes approximately 4.5 minutes and convergence occurs in 4-5 iterations. We also used DSOS programming to design a controller, but we do not present these results here due to space constraints. We note that SOS programming is unable to handle this system due to memory (RAM) constraints. Figure 9-6 demonstrates the performance of the resulting controller from SDSOS programming by plotting initial configurations of the robot that are stabilized to the fixed point. As the plot illustrates, the controller is able to stabilize a very wide range of initial conditions. A video of simulations of the closed loop system started from different initial conditions is available online at http://youtu.be/lmAT556Ar5c.



(a) Nominal pose (fixed point) (b) Stabilized pose 1 (c) Stabilized pose 2

(d) Stabilized pose 3 (e) Stabilized pose 4 (f) Stabilized pose 5

Figure 9-6: Nominal position of the robot, i.e., the fixed point being stabilized (subplot (a)), and configurations of the robot that are stabilized by the controller designed using SDSOS programing (subplots (b)-(f)). A video of simulations of the controller started from different initial conditions is available online at http://youtu.be/lmAT556Ar5c.

Chapter 10

Discussion and Conclusion

In this thesis we have presented an approach for real-time motion planning in a priori unknown environments with dynamic uncertainty in the form of bounded parametric model uncertainty and external disturbances. The method augments the traditional trajectory library approach by constructing stabilizing controllers around the nominal trajectories in a library and computing outer approximations of reachable sets (funnels) for the resulting closed-loop controllers via sums-of-squares (SOS) programming. The pre-computed funnel library is then used to plan online by sequentially composing them together in a manner that ensures obstacles are avoided.

We have demonstrated our approach using extensive simulation experiments on a ground vehicle model. These experiments demonstrate that our approach can afford significant advantages over a trajectory-based approach. We also applied our approach to a quadrotor model and demonstrated how for certain classes of environments we can guarantee that the system will fly forever in a collision-free manner. We have also validated our approach using thorough hardware experiments on a small fixed-wing airplane flying through previously unseen cluttered environments at high speeds. To our knowledge, the resulting hardware demonstrations on a fixed-wing airplane constitute one of the first examples of provably safe and robust control for robotic systems with complex nonlinear dynamics that need to plan in realtime in environments with complex geometric constraints. It is also worth noting

that while one often associates robustness with very conservative behavior, our hardware experiments demonstrate that this need not be the case. In particular, the airplane performs some very aggressive maneuvers while still being robust.

Finally we discussed how DSOS and SDSOS programming, which are recently proposed alternatives to SOS programming, can be used to handle systems with 30-50 dimensional state spaces (well beyond what SOS programming can currently handle). The key idea behind DSOS/SDSOS programming is to replace the positive semidefiniteness constraint in SOS programming with stronger conditions based on diagonal dominance and scaled diagonal dominance. The resulting inner approximations of the SOS cone are referred to as the DSOS and SDSOS cones, which can be optimized over using linear programming (LP) and second-order cone programming (SOCP) respectively. The example applications we considered include a 30 state and 14 control input model of the ATLAS humanoid robot, thus demonstrating the promise of applying the funnel library approach to systems of this scale.

10.1 Challenges and extensions

10.1.1 Numerical difficulties

There are a number of challenges associated with our approach. One of the main difficulties in implementing our method is the relative immaturity of solvers for semidefinite programs (SDPs). While recently released software such as the MOSEK solver [76] have improved the speed with which solutions can be obtained, SDP solvers are still in their infancy as compared to solvers for Linear Programs (LPs). Thus, numerical issues (e.g., due to the scaling of problem data) inevitably arise in practice and must be dealt with in a relatively ad hoc manner (e.g., rescaling problem data or removing redundant decision variables). However, preprocessing of sums-of-squares (SOS) programs and SDPs is an active area of inquiry [83, 62, 91] and solver technology is bound to improve as SDPs are more widely adopted in practice.

10.1.2 Robots with complex geometries

Another challenge associated with our method has to do with implementing the approach on robots with complex kinematics/geometry. One option (as described in Chapter 6) is to project the funnel onto the configuration space (C-space) of the system and perform collision checking against the C-space representation of obstacles. However, computing C-space representations of obstacles is typically challenging for non-trivial geometries, especially since it needs to be done as obstacles are reported in real-time.

For the examples considered in this paper, the geometry of the robots were approximated relatively accurately by spheres. This allowed us to project the funnel onto the $x - y - z$ space (in contrast to the full configuration space of the system) and inflate this projection by the radius of the corresponding sphere. These inflated funnels were then used for collision-checking during real-time planning. For robots with complex geometries, inflating the funnel in $x - y - z$ space in this manner is not an entirely straightforward operation. However, this is potentially a more promising approach than performing collision-checking against C-space obstacles since the inflation can computed in the offline computation stage as it depends only on the geometry of the robot and not the obstacles.

10.1.3 Designing funnel libraries

There is an inherent tradeoff in our approach between the richness of the funnel library and the amount of computation that needs to be performed in real-time in order to be able to search through it. For extremely large funnel libraries, it may be computationally difficult to search through all the funnels while planning online. It is thus important to explore ways in which one could speed up this search. For example, one could exploit the observation that funnels in close proximity to a funnel that is in collision are also likely to be in collision and use this to choose the sequence in which to evaluate funnels for collisions. Further, given certain features of the environment one may be able to predict which funnels are likely to be in collision (without actually performing collision-checking) and evaluate these funnels first. These intuitions have been formalized and exploited using the theory of submodular

optimization in the context of trajectory libraries [33, 31]. The approach allows one to optimize the sequence in which trajectories are evaluated and should be generalizable to funnel libraries as well.

A closely related question is how to choose the funnels in the library in the first place. As we observed in Chapter 7.2 for the quadrotor system, one can derive relatively simple geometric conditions on the environment in order for us to be able to guarantee that the system will be able to navigate through it without collisions. If we know a priori that our environment will satisfy these geometric conditions, this provides a way to check if our funnel library is sufficiently rich. However, for real-world environments (e.g., forests) we may not be able to make such assumptions. Instead, we might have a *generative* (probabilistic) model of our environments and could use this to evaluate/design our funnel library. For instance, it is known that the locations of trees in a forest are modeled well by spatial Poisson processes [100]. In such settings, it may be possible to design a randomized algorithm for generating the funnel library where one samples different realizations of the environment, searches through the existing library to find a collision-free funnel for the sampled environment, and adds a new funnel to the library when such a funnel doesn't exist.

10.1.4 Probabilistic guarantees

Throughout this paper, we have assumed that all disturbances and uncertainty are bounded with probability one. In practice, this assumption may either not be fully valid or could lead to overly conservative performance. In such situations, it is more natural to provide guarantees of a probabilistic nature. Recently, results from classical super-martingale theory have been combined with sums-of-squares programming in order to compute such probabilistic certificates of finite time invariance [97], i.e. provide upper bounds on the probability that a stochastic nonlinear system will leave a given region of state space. Combining the techniques presented in [97] with the approach presented in this work to perform robust online planning on stochastic systems will be the subject of future work.

10.1.5 Reasoning about perception systems

In this thesis, we have focused most of our attention on reasoning about the real-time planning and control systems. In particular, we assumed that the robot is equipped with a perception system that reports (at runtime) regions in which obstacles are guaranteed to lie within. This may not always be a valid assumption; in practice, perceptual systems can often have false negatives and fail to report an obstacle in the environment. In such scenarios, it is unreasonable to expect to guarantee with probability one that the system will remain collision-free and we can only hope for probabilistic guarantees. One could envision modeling this perceptual uncertainty by considering an occupancy map with probabilities of occupancy associated with each voxel. Modeling and reasoning about such perceptual uncertainty remains an important open problem.

10.1.6 Real-time computation of funnels

Due to the computation time associated with funnels, the approach presented in this work had two phases: an offline phase for computing funnels and an online stage for real-time planning with funnels. As we observed in Chapter 9, however, it is possible to get large computational gains using DSOS and SDSOS programming as compared to SOS programming. While in Chapter 9 we highlighted the gains in terms of scalability for large-scale control systems, we also observed gains in running time for smaller-scale systems. This raises the interesting possibility of being able to dispense with the offline computation stage and compute a funnel and feedback controller *in real-time* based on the geometry of the environment. Currently, this would only be feasible for relatively low-dimensional systems but may be an exciting direction to pursue.

We believe that the work presented in this thesis has the potential to be deployed on real robots to make them operate safely in real-world environments. Our hope is that by building upon this work and pursuing the directions for future research presented above we can make this a reality.

Bibliography

[1] A.-A. Agha-Mohammadi, S. Chakravorty, and N. Amato. FIRM: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304, 2014.

[2] A. A. Ahmadi. *Algebraic Relaxations and Hardness Results in Polynomial Optimization and Lyapunov analysis*. PhD thesis, Massachusetts Institute of Technology, August 2011.

[3] A. A. Ahmadi and A. Majumdar. DSOS and SDSOS optimization: LP and SOCP-based alternatives to sum of squares optimization. In *48th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–5. IEEE, 2014.

[4] A. A. Ahmadi and A. Majumdar. DSOS and SDSOS optimization: More tractable alternatives to SOS optimization. *In preparation (http://aaa.lids.mit.edu/publications)*, 2016.

[5] A. A. Ahmadi, A. Majumdar, and R. Tedrake. Complexity of ten decision problems in continuous time dynamical systems. In *Proceedings of the 2013 American Control Conference (ACC)*, 2013.

[6] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical programming*, 95(1):3–51, 2003.

[7] D. Althoff, M. Althoff, and S. Scherer. Online safety verification of trajectories for un-

manned flight with offline computed robust invariant sets. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 3470–3477. IEEE, 2015.

[8] M. Althoff and J. Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

[9] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *IEEE Conference on Decision and Control (CDC)*, pages 4042–4048. IEEE, 2008.

[10] J. Anderson and A. Papachristodoulou. Dynamical system decomposition for efficient, sparse analysis. In *49th IEEE Conference on Decision and Control (CDC)*, pages 6565–6570. IEEE, 2010.

[11] J. Anderson and A. Papachristodoulou. A network decomposition approach for efficient sum of squares programming based analysis. In *American Control Conference (ACC)*, pages 4492–4497. IEEE, 2010.

[12] A. J. Barry. *High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo*. PhD thesis, Massachusetts Institute of Technology, Feb 2016.

[13] A. J. Barry, A. Majumdar, and R. Tedrake. Safety verification of reactive controllers for UAV flight in cluttered environments using barrier certificates. In *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[14] A. Ben-Tal and A. Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*, volume 2. SIAM, 2001.

[15] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner. Grasp planning in complex scenes. In *International Conference on Humanoid Robots*, pages 42–48. IEEE, 2007.

[16] J. V. D. Berg, S. Patil, and R. Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[17] J. T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM Advances in Design and Control. Society for Industrial and Applied Mathematics, 2001.

[18] L. Blackmore, H. Li, and B. Williams. A probabilistic approach to optimal robust path planning with obstacles. In *American Control Conference, 2006*, page 7 pp., june 2006.

[19] G. Blekherman, P. A. Parrilo, and R. R. Thomas. *Semidefinite optimization and convex algebraic geometry*, volume 13. SIAM, 2013.

[20] E. G. Boman, D. Chen, O. Parekh, and S. Toledo. On factor width and symmetric h-matrices. *Linear algebra and its applications*, 405:239–248, 2005.

[21] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[22] R. Brooks. Symbolic error analysis and robot planning. *The International Journal of Robotics Research*, 1(4):29–78, 1982.

[23] C. W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bulletin*, 37(4):97–108, 2003.

[24] A. Bry and N. Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Shanghai, China, 2011.

[25] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research*, 18(6):534–555, June 1999.

[26] A. Charnes and W. W. Cooper. Chance-constrained programming. *Management Science*, 6(1):pp. 73–79, 1959.

[27] M. Chen and C. Tomlin. Exact and efficient hamilton-jacobi reachability for decoupled systems. *arXiv preprint arXiv:1503.05933*, 2015.

[28] G. Chesi and D. Henrion. Guest editorial: Special issue on positive polynomials in control. *IEEE Transactions on Automatic Control*, 54(5):935–936, 2009.

[29] G. E. Collins and H. Hong. *Partial cylindrical algebraic decomposition for quantifier elimination*. Springer, 1998.

[30] E. Coumans et al. Bullet physics library. *bulletphysics. org*, 4(6), 2014.

[31] D. Dey. *Predicting Sets and Lists: Theory and Practice*. PhD thesis, Carnegie Mellon University, 2015.

[32] D. Dey, T. Liu, B. Sofman, and D. Bagnell. Efficient optimization of control libraries. Technical report, Technical Report (CMU-RI-TR-11-20), 2011.

[33] D. Dey, T. Y. Liu, M. Hebert, and A. J. Bagnell. Contextual sequence prediction with application to control library optimization. In Proceedings of Robotics Science and Systems, 2012.

[34] D. Dey, K. S. Shankar, S. Zeng, R. Mehta, M. T. Agcayazi, C. Eriksen, S. Daftry, M. Hebert, and J. A. Bagnell. Vision and learning for deliberative monocular cluttered flight. *arXiv preprint arXiv:1411.6326*, 2014.

[35] A. Domahidi and J. Jerez. FORCES Professional. Embotech GmbH (`http://embotech.com/FORCES-Pro`), July 2014.

[36] L. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[37] E. Frazzoli, M. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics*, 21(6):1077–1091, December 2005.

[38] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-Time Motion Planning for Agile Autonomous Vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, January–February 2002.

[39] S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *Automated Deduction–CADE-24*, pages 208–214. Springer, 2013.

[40] M. Ghasemi and M. Marshall. Lower bounds for polynomials using geometric programming. *SIAM Journal on Optimization*, 22(2):460–473, 2012.

[41] J. Gillula, H. Huang, M. Vitus, and C. Tomlin. Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1649–1654. IEEE, 2010.

[42] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.

[43] C. Green and A. Kelly. Toward optimal sampling in the space of paths. In *13th International Symposium of Robotics Research*, November 2007.

[44] L. Guibas, D. Hsu, H. Kurniawati, and E. Rehman. Bounded uncertainty roadmaps for path planning. In *Algorithmic Foundation of Robotics VIII*, pages 199–215. Springer, 2009.

[45] E. J. Hancock and A. Papachristodoulou. Structured sum of squares for networked systems analysis. In *Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 7236–7241. IEEE, 2011.

[46] D. Henrion and M. Korda. Convex computation of the region of attraction of polynomial control systems. *IEEE Transactions on Automatic Control*, 59(2):297–312, 2014.

[47] T. Horiuchi, M. Inoue, T. Konno, and Y. Namita. Real-time hybrid experimental system with actuator delay compensation and its application to a piping system with energy absorber. *Earthquake Engineering & Structural Dynamics*, 28(10):1121–1141, 1999.

[48] P. Jacobs and J. Canny. Robust motion planning for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2–7. IEEE, 1990.

[49] Z. Jarvis-Wloszek, R. Feeley, W. Tan, K. Sun, and A. Packard. Some controls applications of sum of squares programming. In *42nd IEEE Conference on Decision and Control*, volume 5, pages 4676 – 4681, December 2003.

[50] M. Junger, T. M. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-art*. Springer, 2009.

[51] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, January 1998.

[52] R. Kamyar, M. Peet, and Y. Peet. Solving large-scale robust stability problems by exploiting the parallel structure of polya's theorem. *IEEE Transactions on Automatic Control*, 58(8), 2012.

[53] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 30:846–894, June 2011.

[54] J. Kuffner and S. Lavalle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 995–1001, 2000.

[55] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake. Optimization-based locomotion planning, estimation, and control design for the Atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016.

[56] A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis: internal approximation. *Systems & control letters*, 41(3):201–211, 2000.

[57] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How. Motion planning for urban driving using RRT. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1681–1686. IEEE, 2008.

[58] J.-C. Latombe, A. Lazanas, and S. Shekhar. Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*, 52(1):1–47, 1991.

[59] H. X. Li and B. Williams. Generative planning for hybrid systems based on flow tubes. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 206–213, 2008.

[60] C. Liu and C. Atkeson. Standing balance control using a trajectory library. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3031–3036. IEEE, 2009.

[61] L. Ljung. System identification toolbox for use with matlab. 2007.

[62] J. Lofberg. Pre- and post-processing sum-of-squares programs in practice. *IEEE Transactions On Automatic Control*, 54(5):1007–, May 2009.

[63] T. Lozano-Perez, M. Mason, and R. Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):3–24, 1984.

[64] D. Luenberger. An introduction to observers. *IEEE Transactions on Automatic Control*, 16(6):596–602, 1971.

[65] A. Majumdar, A. A. Ahmadi, and R. Tedrake. Control design along trajectories with sums of squares programming. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4054–4061, 2013.

[66] A. Majumdar and R. Tedrake. Robust online motion planning with regions of finite time invariance. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, page 16, Cambridge, MA, June 2012.

[67] N. Malone, K. Manavi, J. Wood, and L. Tapia. Construction and use of roadmaps that incorporate workspace modeling errors. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1264–1271. IEEE, 2013.

[68] M. Mason. The mechanics of manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 544–548. IEEE, 1985.

[69] D. Mayne, M. Seron, and S. Rakovic. Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica*, 41(2):219–224, 2005.

[70] A. Megretski. Systems polynomial optimization tools (SPOT), available online: http://web.mit.edu/ameg/www. 2010.

[71] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Proceedings of the 12th International Symposium on Experimental Robotics (ISER 2010)*, 2010.

[72] P. Missiuro and N. Roy. Adapting probabilistic roadmaps to handle uncertain maps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1261–1267. IEEE, 2006.

[73] I. Mitchell, A. Bayen, and C. Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 50(7):947–957, July 2005.

[74] J. Moore. *Robust Post-Stall Perching with a Fixed-Wing UAV*. PhD thesis, Massachusetts Institute of Technology, September 2014.

[75] J. Moore, R. Cory, and R. Tedrake. Robust post-stall perching with a simple fixed-wing glider using lqr-trees. *Bioinspiration and Biomimetics*, 9(2):15, June 2014.

[76] A. Mosek. The mosek optimization software. *Online at http://www.mosek.com*, 54, 2010.

[77] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1988.

[78] J. L. Ny and G. Pappas. Sequential composition of robust controller specifications. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2012.

[79] F. Oldewurtel, C. N. Jones, and M. Morari. A tractable approximation of chance constrained stochastic mpc based on affine disturbance feedback. In *IEEE Conference on Decision and Control*, pages 4731–4736. IEEE, 2008.

[80] P. A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, May 18 2000.

[81] P. A. Parrilo. Exploiting algebraic structure in sum of squares programs. In *Positive polynomials in control*, pages 181–194. Springer, 2005.

[82] S. Patil, G. Kahn, M. Laskey, J. Schulman, K. Goldberg, and P. Abbeel. Scaling up gaussian belief space planning through covariance-free trajectory optimization and automatic differentiation. In *Algorithmic Foundations of Robotics XI*, pages 515–533. Springer, 2015.

[83] F. Permenter and P. Parrilo. Basis selection for SOS programs via facial reduction and polyhedral approximations. *Proceedings of the IEEE Conference on Decision and Control, December 2014.*

[84] R. Platt, L. Kaelbling, T. Lozano-Perez, and R. Tedrake. Non-gaussian belief space planning: Correctness and complexity. In *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[85] R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Proceedings of Robotics: Science and Systems*, 2010.

[86] S. Prajna. Barrier certificates for nonlinear model validation. *Automatica*, 42(1):117 – 126, 2006.

[87] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In R. Alur and G. Pappas, editors, *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 271–274. Springer Berlin / Heidelberg, 2004.

[88] S. Prentice and N. Roy. The Belief Roadmap: Efficient planning in linear POMDPs by factoring the covariance. In *Proceedings of the 12th International Symposium of Robotics Research (ISRR)*, 2007.

[89] B. Reznick. Some concrete aspects of Hilbert's 17th problem. In *Contemporary Mathematics*, volume 253, pages 251–272. American Mathematical Society, 2000.

[90] B. W. Satzinger, C. Lau, M. Byl, and K. Byl. Experimental results for dexterous quadruped locomotion planning with RoboSimian. In *Experimental Robotics*, pages 33–46. Springer, 2016.

[91] P. Seiler, Q. Zheng, and G. Balas. Simplification methods for sum-of-squares programs. *arXiv preprint arXiv:1303.0714*, 2013.

[92] P. Sermanet, M. Scoffier, C. Crudele, U. Muller, and Y. LeCun. Learning maneuver dictionaries for ground robot planning. In *Proc. 39th International Symposium on Robotics (ISR08)*, 2008.

[93] A. Shkolnik. *Sample-Based Motion Planning in High-Dimensional and Differentially-Constrained Systems*. PhD thesis, MIT, February 2010.

[94] R. Sipahi, T. Vyhlídal, S.-I. Niculescu, and P. Pepe. *Time Delay Systems: Methods, Applications and New Trends*, volume 423. Springer, 2012.

[95] F. M. Sobolic. Agile flight control techniques for a fixed-wing aircraft. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, June 2009.

[96] M. Spong. The swingup control problem for the acrobot. *IEEE Control Systems Magazine*, 15(1):49–55, February 1995.

[97] J. Steinhardt and R. Tedrake. Finite-time regional verification of stochastic nonlinear systems. *International Journal of Robotics Research*, 31(7):901–923, June 2012.

[98] B. Stevens and F. Lewis. *Aircraft Control and Simulation*. John Wiley & Sons, Inc., 1992.

[99] M. Stolle and C. Atkeson. Policies based on trajectory libraries. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*. IEEE, 2006.

[100] D. Stoyan and A. Penttinen. Recent applications of point process methods in forestry statistics. *Statistical Science*, pages 61–78, 2000.

[101] R. Tae, B. Dumitrescu, and L. Vandenberghe. Interior-point algorithms for sum-of-squares optimization of multidimensional trigonometric polynomials. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 3, 2007.

[102] R. Tedrake, I. R. Manchester, M. M. Tobenkin, and J. W. Roberts. LQR-Trees: Feedback motion planning via sums of squares verification. *International Journal of Robotics Research*, 29:1038–1052, July 2010.

[103] M. M. Tobenkin, I. R. Manchester, and R. Tedrake. Invariant funnels around trajectories using sum-of-squares programming. *Proceedings of the 18th IFAC World Congress, extended version available online: arXiv:1010.3013 [math.DS]*, 2011.

[104] N. D. Toit and J. Burdick. Probabilistic collision checking with chance constraints. *IEEE Transactions on Robotics*, 27(4):809–815, 2011.

[105] U. Topcu, A. Packard, and P. Seiler. Local stability analysis using simulations and sum-of-squares programming. *Automatica*, 44(10):2669 – 2675, 2008.

[106] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[107] M. Vitus and C. Tomlin. Closed-loop belief space planning for linear, gaussian systems. In *International Conference on Robotics and Automation (ICRA)*, pages 2152–2159. IEEE, 2011.

[108] M. Vitus and C. Tomlin. A hybrid method for chance constrained control in uncertain environments. In *Conference on Decision and Control (CDC)*, pages 2177–2182. IEEE, 2012.

[109] H. Yazarel and G. Pappas. Geometric programming relaxations for linear system reachability. In *Proceedings of the American Control Conference*, volume 1, pages 553–559. IEEE, 2004.

[110] Y. Zhang, M. Peet, and K. Gu. Reducing the complexity of the sum-of-squares test for stability of delayed linear systems. *IEEE Transactions on Automatic Control*, 56(1):229–234, 2011.