Robust Online Motion Planning with Reachable Sets

by

Anirudha Majumdar

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

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author .... [illegible]
Department of Electrical Engineering and Computer Science
May 22, 2013

Certified by.................................... [illegible]
Russ Tedrake
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by.................................... [illegible]
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Theses

Robust Online Motion Planning with Reachable Sets

by

Anirudha Majumdar

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

In this thesis we consider the problem of generating motion plans for a nonlinear dynamical system that are guaranteed to succeed despite uncertainty in the environment, parametric model uncertainty, disturbances, and/or errors in state estimation. Furthermore, we consider the case where these plans must be generated online, because constraints such as obstacles in the environment may not be known until they are perceived (with a noisy sensor) at runtime. Previous work on feedback motion planning for nonlinear systems was limited to offline planning due to the computational cost of safety verification. Here we augment the traditional *trajectory library* approach by designing locally stabilizing controllers for each nominal trajectory in the library and providing guarantees on the resulting closed loop systems. We leverage *sums-of-squares programming* to design these locally stabilizing controllers by explicitly attempting to minimize the size of the worst case reachable set of the closed-loop system subjected to bounded disturbances and uncertainty. The reachable sets associated with each trajectory in the library can be thought of as "funnels" that the system is guaranteed to remain within. 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 our method on a simulation of a plane flying through a two dimensional forest of polygonal trees with parametric uncertainty and disturbances in the form of a bounded "cross-wind". We further validate our approach by carefully evaluating the guarantees on invariance provided by funnels on two challenging underactuated systems (the "Acrobot" and a small-sized airplane).

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

Acknowledgments

I would like to thank my advisor, Russ Tedrake, for giving me the opportunity to work at the Robot Locomotion Group first as an undergrad visiting research student and then as a PhD student. Russ's deep insights into difficult problems, quick flashes of intuition and positive outlook have been an inspiration for me through the last three years. It has been an incredible pleasure to work with Russ and I look forward to the years to come.

I would also like to thank Dan Koditschek, Dan Lee, and Vijay Kumar for being advisors and mentors during my undergraduate years at UPenn. 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 thrive, patiently listening to my half-baked ideas, and gently guiding me through the research path. 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 absolutely 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 different problems that need to be solved for a robot to work. 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 guided tour of the GRASP lab to a freshman who walked into his office with no appointment on a busy day.

I would like to thank the 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. In particular, I would like to thank Amir Ali Ahmadi for teaching me how to view problems through the lens of computational complexity theory (and for telling me how to prove Goldbach's conjecture - maybe I'll find time to do it one of these weekends), Mark Tobenkin for writing my first sums-of-squares program for me and teaching me a lot of math, and Ian Manchester, Zack Jackowski, Michael Levashov and Alec Shkolnik for answering my incessant annoying questions during my first weeks at the lab and teaching me about sums-of-squares programming. I have also had numerous enlightening conversations with the rest of the members of the lab including Mike Posa, Frank Permenter, Ram Vasudevan, Andy Barry, Tim Jenks, Scott Kuindersma, Joe Moore, Hongkai Dai, Robin Deits, John Roberts, Zico Kolter, Elena Glassman, Jacob Steinhardt, Kanako Miura and Jessy Grizzle. I would also like to thank Ned Burnell for working with me as a UROP.

Andy Barry, Tim Jenks and Zack Jackowski also deserve particular thanks for building the hardware platforms on which the experiments presented in the latter parts of the thesis were performed (Andy and Tim build the "Wingeron" airplane and Zack built the Acrobot). Without them, these experiments would have been impossible.

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.

Finally, I would like to thank my parents and sister for unconditional love over the years. Without them, none of this would be possible.

Contents

List of Figures

Chapter 1

Introduction

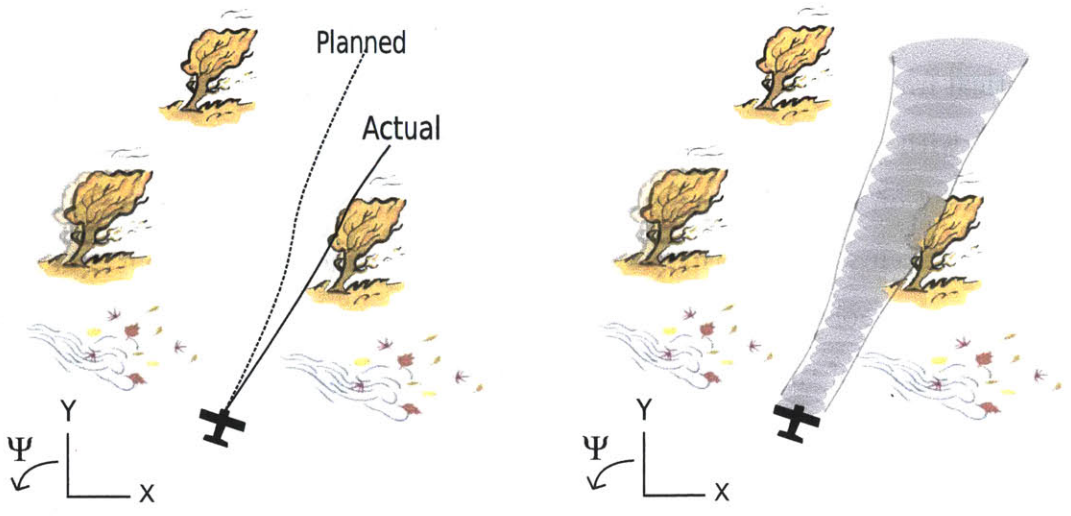
The ability to plan and execute dynamic motions under uncertainty is a critical skill that our robots must have in order to perform useful tasks in the real world. Whether the robot is an unmanned aerial vehicle (UAV) flying at high speeds through a cluttered environment in the presence of wind gusts, a legged robot traversing rough terrain, or a micro-air vehicle with noisy on-board sensing, the inability to take into account disturbances, model uncertainty and state uncertainty can have disastrous consequences.

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) [17], RRT* [16], and related trajectory library approaches [18] [9] [37] can handle large state-space dimensions and complex differential constraints. These algorithms have been successfully demonstrated on a wide variety of hardware platforms [33] [32]. However, a significant limitation 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. This issue is sketched in Figure 1-1(a), where a UAV attempting to fly through a forest with a heavy cross-wind gets blown off its planned nominal trajectory and crashes into a tree.

Recently, planning algorithms which explicitly take into account feedback control have been proposed. The LQR-Trees algorithm [39] creates a tree of locally stabilizing controllers which can take any initial condition in some bounded region in state space to the desired goal. The approach leverages *sums-of-squares programming* (SOS) [29] for computing regions of finite time invariance for the locally stabilizing controllers. 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.

In this thesis, we present a partial solution to this problem by combining trajectory libraries, feedback control, and tools from Lyapunov theory and algorithmic algebra in order to perform robust motion planning in the face of uncertainty. In particular, in the offline computation stage, we first design a finite library of open loop trajectories. For each trajectory in this library, we use *sums-of-squares programming* (SOS) 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). Once we have pre-computed such a *funnel library*, we can *sequentially compose* these robust motion plans online in order to operate in a provably safe manner. Given estimated positions of obstacles in some finite sensing horizon, we can choose a funnel from our library that does not intersect an obstacle. We can do this planning in a receding horizon fashion to achieve the desired task.

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



(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.

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 state uncertainty than the second one. Thus, it may actually 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.

1.1 Related Work

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 [18], autonomous ground vehicle navigation [32], and grasping [3] [8]. The Maneuver Automaton [9] 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 realtime motion planning with static and dynamic obstacles [10]. 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 [12]. More recently, tools from sub-modular sequence optimization have been leveraged in the optimization of the sequence and content of trajectories evaluated online [8].

The body of literature that deals with planning under uncertainty is also relevant to the work presented here [5] [30]. While these approaches generate motion plans that explicitly reason about the effect of uncertainty and disturbances on the behavior of the system, distributions over states ("belief states") are typically approximated as Gaussians for computational efficiency and the true belief state is not tracked. Thus, in general, one does not have robustness guarantees. The approach we take here is to assume that disturbances/uncertainty are *bounded* and provide explicit bounds on the reachable set to facilitate safe operation of the system.

Robust motion planning has also been a very active area of research in the robotics community. Early work focused on the purely kinematic problem of planning paths through configuration space with "tubes" of specified radii around them such that all paths in the tube remain collision-free [13]. Recent work has focused on reasoning more explicitly about the manner in which disturbances and uncertainties influence the dynamics of the robot, and is closer in spirit to the work presented here. In particular, [31] approaches the problem through dynamic programming on a model with disturbances by making use of the Maneuver Automaton framework mentioned earlier. However, the work does not take into account obstacles in the environment and does not provide or make use of any explicit guarantees on allowed deviations from the planned trajectories in the Maneuver Automaton. Another approach that is closely related to ours is Model Predictive Control with Tubes [26]. The idea is to solve the optimal control problem online with guaranteed "tubes" that the trajectories stay in. However, the method is limited to linear systems and convex constraints.

In [11], the authors design motion primitives for making a quadrotor perform an autonomous backflip. Reachable sets for the primitives are computed via a Hamilton-Jacobi-Bellman differential game formulation. However, a *predetermined* controller is employed for the reachability analysis instead of designing a controller that seeks to minimize the size of the reachable set (it is possible in principle to do this, but inconvenient in practice). More importantly, while their 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*.

The approach that is perhaps most closely related to our work is the recent work presented in [28]. 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 [28] are our focus on the *realtime* aspect of the problem and use of sums-of-squares programming as a way of computing reachable sets. In [28], 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.

A critical component of the work presented here is the computation of "funnels" for nonlinear systems via Lyapunov functions. This idea, along with the metaphor of a "funnel", was introduced to the robotics community in [6], where funnels were *sequentially composed* in order to produce dynamic behaviors in a robot. In recent years, sums-of-squares programming has emerged as a way of checking the Lyapunov function conditions associated with each funnel [29]. 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 semi-definite optimization program and is amenable to efficient computational algorithms such as interior point methods [29]. 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 designing controllers along time-indexed trajectories of a system that attempt to maximize the size of the set of initial conditions that are driven to a goal set [23]. In this thesis, we extend this approach to compute controllers that explicitly minimize the size of reachable sets around trajectories. 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 and state uncertainty.

An alternative approach to computing outer approximations of reachable sets is the one presented in [14]. The method relies on computing regions of finite time invariance using locally valid "barrier functions". Although the approach does not involve computing controllers that attempt to minimize the size of the reachable set, it is conceivable that the method could be extended to do so.

1.2 Contributions

This thesis makes three main contributions[1]. First, in Chapter 2 we provide a way of designing controllers using sums-of-squares programming that explicitly seek to minimize the effect that disturbances and uncertainties have on the system by minimizing the size of the reachable set ("funnel"). These controllers and corresponding reachable set guarantees can be generated for time-varying polynomial systems subjected to a broad class of uncertainties (bounded uncertainty in parameters entering polynomially in the dynamics). This is an extension of results presented in [23], where the control design approach seeks to maximize the size of the set of initial conditions that are guaranteed to be driven to some predefined goal set. The present work extends this approach to handle disturbances/uncertainty and provide guarantees on reachable sets rather than the set of initial conditions that are driven to the goal set.

Second, in Chapter 3 we show how a library of such funnels can be precomputed offline and composed together at runtime in a receding horizon manner while ensuring

[1]Preliminary versions of these results have appeared in [24, 23, 25]

that the resulting closed loop system is "safe" (i.e. avoids obstacles and switches between the planned sequence of funnels). This can be viewed as an extension of the LQR-Trees algorithm [39] for feedback motion planning, which was limited to offline planning due to the relatively large computational cost of computing the funnels. In contrast to LQR-Trees, our algorithm is suitable for real-time, online planning. We expect this framework to be useful in robotic tasks where the dynamics and perceptual system of the robot are difficult to model perfectly and for which the robot does not have access to the geometry of the environment until runtime. In Section 3.2, we demonstrate our method on a simulation of a plane flying through a two dimensional forest of polygonal trees with parametric uncertainty and disturbances in the form of a bounded "cross-wind".

Finally, in Chapter 4 we provide hardware validation of funnels computed using sums-of-squares programming on two challenging underactuated systems. To our knowledge, these experimental results constitute the first hardware validation of sums-of-squares programming based funnels.

Chapter 2

Computing Reachable Sets

A considerable amount of research effort in the motion planning community has focused on the design of trajectory libraries (see Section 1.1). Hence, here we assume that we are provided with a trajectory library consisting of a finite set of nominal feasible trajectories for the robot and concentrate our discussion on extending the techniques for the computation of controllers and associated regions of finite time invariance presented in [23] to compute reachable sets when there is uncertainty in the dynamics and state. Let

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

be the control system under consideration. Here, $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}^m$ is the control input and $w(t) \in \mathbb{R}^d$ is the disturbance/uncertainty term. We assume here that f and g are polynomials[1] in x and w. We further assume that $w(t)$ belongs to a bounded semialgebraic set $W = \{w \mid W_k(w) \geq 0, \forall k = 1, \dots K\}$.

Let $x_0(t) : [0, T] \mapsto \mathbb{R}^n$ be a nominal trajectory in our library that we want the system to follow and $u_0(t) : [0, T] \mapsto \mathbb{R}^m$ be the corresponding nominal open-loop control input. Defining new coordinates $\bar{x} = x - x_0(t)$ and $\bar{u} = u - u_0(t)$, we

[1]With the right change of coordinates, one can express the dynamics of most robotic systems as polynomials. For example, the dynamics of most rigid body systems can be transformed into polynomials by introducing new variables, s_i and c_i, for $\sin(\theta_i)$ and $\cos(\theta_i)$, and imposing the constraint that $s_i^2 + c_i^2 = 1$ (this equality constraint is easily imposed in the sums-of-squares programming framework). Another approach is to simply Taylor approximate the non-polynomial dynamics.

can rewrite the dynamics in these variables as $\dot{\bar{x}} = \dot{x} - \dot{x}_0(t)$. Then, given a set of initial conditions $F(0)$, we seek to design a controller that attempts to minimize the "size" of the time-varying reachable sets $B(t)$ (we will formalize what we mean by "size" soon). For a given controller, the reachable set $B(t)$ is the set of states that the system may be driven to at time t by *some* disturbance (i.e. some choice of $w(t) \in W$), given that the initial condition lay in the set $F(0)$. In general, we will not be able to compute reachable sets exactly. Rather, we will compute *outer approximations* of the reachable sets and design controllers to minimize the "size" of the outer approximation. Checking the following *invariance* condition for all $t \in [0, T]$ is sufficient for establishing the sets $F(t)$ as outer approximations of the reachable sets $B(t)$:

$$\bar{x}(0) \in F(0) \implies \bar{x}(t) \in F(t), \ \forall w : [0, T] \to W. \tag{2.1}$$

Our task will be to design time-varying controllers that minimize the size of the "funnel" described by the sets $F(t)$. We describe the funnel as a time-varying sub-level set of a function $V(\bar{x}, t)$:

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

This specification of the funnel allows us to use $\rho(t)$ as a natural surrogate for the "size" of the funnel at time t. We impose the following condition on $V(\bar{x}, t)$:

$$V(\bar{x}, t) = \rho(t) \implies \dot{V}(\bar{x}, t, w) < \dot{\rho}(t), \ \forall w(t) \in W \tag{2.2}$$

Letting $F(0) = \{\bar{x} \mid V(\bar{x}, 0) \leq \rho_0\}$, it is easy to see that this condition implies the invariance condition (2.1). Here, $\dot{V}(\bar{x}, t, w)$ is computed as:

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

In principle, we can parameterize our function $V(\bar{x}, t)$ as a polynomial in both t and x and check (2.2) $\forall t \in [0, T]$. However, as described in [40], this leads to expensive sums-of-squares programs. Instead, we can get large computational gains with little loss in accuracy by checking (2.2) at sample points in time $t_i \in [0,T], i = 1 \dots N$. As discussed in [40], for a fixed $V(\bar{x},t)$ and dynamics (and under mild conditions on both), increasing the density of the sample points eventually recovers (2.2) $\forall t \in [0,T]$. This allows us to *check* the answers we obtain from the sums-of-squares program below by sampling finely enough.

Thus, we parameterize $V(\bar{x},t)$ and the controller $\bar{u}$ by polynomials $V(\bar{x},t_i)$ and $\bar{u}(\bar{x},t_i)$ respectively at each sample point in time. Using $\sum_{i=1}^{N} \rho(t_i)$ as the cost function, we can write the following sums-of-squares (SOS) program:

$$\underset{\rho(t_i),L(\bar{x},t_i,w),V(\bar{x},t_i),\bar{u}(\bar{x},t_i),M_k(\bar{x},t_i,w)}{\textbf{minimize}} \quad \sum_{i=1}^{N} \rho(t_i) \tag{2.3}$$

subject to:

$$V(\bar{x},t_i) \text{ is SOS }, \ \forall i = 1 \dots N \tag{2.4}$$

$$-\dot{V}(\bar{x},t_i,w) + \dot{\rho}(t_i) + L(\bar{x},t_i,w)(V(\bar{x},t_i) - \rho(t_i)) \dots$$
$$-\sum_{k=1}^{K} M_k(\bar{x},t_i,w) W_k(w) \text{ is SOS }, \ \forall i = 1 \dots N \tag{2.5}$$

$$M_k(\bar{x},t_i,w) \text{ is SOS }, \ \forall i = 1 \dots N \tag{2.6}$$

$$\rho(t_i) \geq 0, \forall i = 2 \dots N \tag{2.7}$$

$$V(e,t_i) = V_{guess}(e,t_i), \ \forall i = 1 \dots N \tag{2.8}$$

Here, $L(\bar{x},t_i,w)$ and $M_k(\bar{x},t_i,w)$ are "multiplier" terms that help to enforce the invariance condition. It is easy to see that condition (2.5) is a sufficient condition for ensuring (2.2) at the sample points in time. This is because for all $w \in W$, we must have $\sum_k M_k(\bar{x},t_i,w)W_k(w) \geq 0$, since we have $M_k(\bar{x},t_i,w) \geq 0$ and $W_k(w) \geq 0$. Thus, when $V(\bar{x},t_i) = \rho(t_i)$, condition (2.5) implies[2] that $\dot{V}(\bar{x},t_i,w) < \dot{\rho}(t_i)$.

Condition (2.8) is a normalization constraint where e is the vector of all ones and $V_{guess}(\bar{x},t)$ is the candidate for $V(\bar{x},t)$ that is used to initialize the alternation scheme

[2]SOS decompositions obtained from numerical solvers generically provide proofs of polynomial *positivity* as opposed to mere non-negativity (see the discussion in [1, p.41]). This is why we claim a strict inequality here.

outlined below for solving the above optimization program. (If we do not impose a normalization constraint on $V(\bar{x}, t_i)$, $\sum_{i=1}^{N} \rho(t_i)$ can be made arbitrarily small simply by scaling the coefficients of $V(\bar{x}, t_i)$). We use a piecewise linear parameterization of $\rho(t)$ and can thus compute $\dot{\rho}(t_i) = \frac{\rho(t_{i+1}) - \rho(t_i)}{t_{i+1} - t_i}$. Similarly, we approximate $\frac{\partial V(\bar{x}, t_i)}{\partial t} \approx \frac{V(\bar{x}, t_{i+1}) - V(\bar{x}, t_i)}{t_{i+1} - t_i}$.

The above optimization program is not convex in general since it involves conditions that are bilinear in the decision variables. However, the conditions are linear in $L(\bar{x}, t_i, w)$, $\bar{u}(\bar{x}, t_i)$, $M_k(\bar{x}, t_i, w)$ for fixed $V(\bar{x}, t_i)$, $\rho(t_i)$, and are linear in $V(\bar{x}, t_i)$, $\rho(t_i)$, $M_k(\bar{x}, t_i, w)$ for fixed $L(\bar{x}, t_i, w)$, $\bar{u}(\bar{x}, t_i)$. Thus, in principle we could use a bilinear alternation scheme for solving this optimization program by alternating between the two sets of decision variables, $(L(\bar{x}, t_i, w)$, $\bar{u}(\bar{x}, t_i), M_k(\bar{x}, t_i, w))$ and $(V(\bar{x}, t_i), \rho(t_i), M_k(\bar{x}, t_i, w))$ and repeat until convergence in the following two steps: (1) Fix $(V(\bar{x}, t_i), \rho(t_i))$ and search for $(L(\bar{x}, t_i, w)$, $\bar{u}(\bar{x}, t_i), M_k(\bar{x}, t_i, w))$, and (2) Fix $(L(\bar{x}, t_i, w)$, $\bar{u}(\bar{x}, t_i))$ and search for $(V(\bar{x}, t_i), \rho(t_i), M_k(\bar{x}, t_i, w))$. However, in the first step of this alternation, we cannot optimize the cost function $\sum_{i=1}^{N} \rho(t_i)$ since we have to fix $\rho(t_i)$ (we *can* optimize the cost function in the second step). We could simply make the first step a *feasibility* problem (instead of optimizing a cost function), but this prevents us from searching for a controller that explicitly seeks to minimize the desired cost function since in the second step of the alternation, we do not search for a controller. We get around this issue by introducing an additional step in the alternation, in which we fix $L(\bar{x}, t_i, w)$ and $V(\bar{x}, t_i)$ and search for $\bar{u}(\bar{x}, t_i)$, $\rho(t_i)$ and $M_k(\bar{x}, t_i, w)$, while minimizing $\sum_{i=1}^{N} \rho(t_i)$. The steps in the alternation are summarized in Algorithm 1.

Each iteration of the alternations in Algorithm 1 is guaranteed to obtain an objective $\sum_{i=1}^{N} \rho^{\star}(t_i)$ that is at least as small as the previous one since a solution to the previous iteration is also valid for the current one. Hence, since the optimal cost is lower bounded (by zero), the iterations are guaranteed to converge.

Section 2.2 discusses how to initialize $V(\bar{x}, t_i)$ and $\rho(t_i)$ for Algorithm 1.

Algorithm 1 Robust Controller Design

```
Initialize $V(\bar{x}, t_i)$ and $\rho(t_i)$,   $\forall i = 1 \dots N$
$\rho_{prev}(t_i) = 0$,   $\forall i = 1 \dots N$.
converged = false;
while $\neg$converged do
    STEP 1 : Solve feasibility problem by searching for $L(\bar{x}, t_i, w)$, $\bar{u}(\bar{x}, t_i)$, $M_k(\bar{x}, t_i, w)$, and fixing $V(\bar{x}, t_i)$, $\rho(t_i)$.
    STEP 2 : Minimize $\sum_{i=1}^{N} \rho(t_i)$ by searching for $\bar{u}(\bar{x}, t_i)$, $\rho(t_i)$, $M_k(\bar{x}, t_i, w)$, and fixing $L(\bar{x}, t_i, w)$, $V(\bar{x}, t_i)$.
    STEP 3 : Minimize $\sum_{i=1}^{N} \rho(t_i)$ by searching for $V(\bar{x}, t_i)$, $\rho(t_i)$, $M_k(\bar{x}, t_i, w)$, and fixing $L(\bar{x}, t_i, w)$, $\bar{u}(\bar{x}, t_i)$.
    if $\frac{\sum_{i=1}^{N} \rho(t_i) - \sum_{i=1}^{N} \rho_{prev}(t_i)}{\sum_{i=1}^{N} \rho_{prev}(t_i)} < \epsilon$ then
        converged = true;
    end if
    $\rho_{prev}(t_i) = \rho(t_i)$,   $\forall i = 1 \dots N$.
end while
```

2.1 Incorporating Actuator Limits

Our method allows us to incorporate actuator limits into the control design procedure. Although we examine the single-input case in this section, this framework is very easily extended to handle multiple inputs.

Let the control law $u(x)$ be mapped through the following control saturation function:

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

where u_{max} and u_{min} are the maximum and minimum allowable inputs respectively. Then, a piecewise analysis of $\dot{V}(\bar{x}, t)$ can be used to check the Lyapunov conditions are satisfied even when the control input saturates. Defining:

$$\begin{aligned}
\dot{V}_{min}(\bar{x},t,w) =& \frac{\partial V(\bar{x},t)}{\partial \bar{x}}^T [f(\bar{x}+x_0(t),w)\dots \\
& + g(\bar{x}+x_0,w)u_{min} - \dot{x}_0(t)] + \frac{\partial V(\bar{x},t)}{\partial t} \qquad (2.9) \\
\dot{V}_{max}(\bar{x},t,w) =& \frac{\partial V(\bar{x},t)}{\partial \bar{x}}^T [f(\bar{x}+x_0(t),w)\dots \\
& + g(\bar{x}+x_0,w)u_{max} - \dot{x}_o(t)] + \frac{\partial V(\bar{x},t)}{\partial t} \qquad (2.10) \\
& \qquad (2.11)
\end{aligned}$$

we must check the following conditions:

$$u(\bar{x}) \leq u_{min} \implies \dot{V}_{min}(\bar{x},t) < \dot{\rho}(t) \qquad (2.12)$$

$$u(\bar{x}) \geq u_{max} \implies \dot{V}_{max}(\bar{x},t) < \dot{\rho}(t) \qquad (2.13)$$

$$u_{min} \leq u(\bar{x}) \leq u_{max} \implies \dot{V}(\bar{x},t) < \dot{\rho}(t) \qquad (2.14)$$

Algorithm 1 can be modified to enforce these conditions with extra multipliers in a manner identical to the one presented in [23]. This modification is relatively straightforward and we do not present it here.

2.2 Initializing $V(\bar{x},t_i)$ and $\rho(t_i)$

Obtaining an initial guess for $V(\bar{x},t_i)$ and $\rho(t_i)$ is an important part of Algorithm 1. In [39], 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:

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

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 $x_0(t)$. Q and R are positive-definite cost-matrices. The function:

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

is our initial Lyapunov candidate. $V_{guess}(\bar{x}, t_N) = \bar{x}^T S(0)\bar{x}$, along with a choice of ρ_0 can be used to determine the initial condition set, $F(0)$, of the funnel:

$$F(0) = \{\bar{x} \mid \bar{x} \in \mathbb{R}^n,\ \bar{x}^T S(0)\bar{x} \leq \rho_0\}.$$

We find that initializing $\rho(t_i)$ to an exponential function in time, $e^{\gamma t_i}$, works quite well in practice. We can tune γ to obtain feasible solutions. Intuitively, higher values of γ correspond to "larger" reachable sets and thus are more likely to be feasible.

We will demonstrate this control design procedure in Section 3.2 and Chapter 4.

Chapter 3

Funnel Libraries

The tools from Chapter 2 can be used to create libraries of funnels offline. Given a trajectory library, $\mathcal{T}$, consisting of finitely many trajectories $x_i(t)$, we can compute robust controllers $u_i(x,t)$ and associated reachable sets (funnels) for each trajectory in $\mathcal{T}$. However, there is an important issue that needs to be addressed when designing libraries of funnels and has an analogy in the traditional trajectory library approach. In particular, trajectories in a traditional trajectory library need to be designed in a way that allows them to be sequenced together. More formally, let $\mathcal{P}$ denote the projection operator that projects a state, x, onto the subspace formed by the non-cyclic dimensions of the system (i.e. the dimensions with respect to which the dynamics of the system are *not* invariant). Then, for two trajectories $x_i(t)$ and $x_j(t)$ to be executed one after another, we must have

$$\mathcal{P}(x_i(T_i)) = \mathcal{P}(x_j(0)).$$

Note that the cyclic coordinates do not pose a problem since one can simply "shift" trajectories around in these dimensions. This issue is discussed thoroughly in [9] and is addressed by having a *trim trajectory* of the system that other trajectories (*maneuvers*) start from and end at (of course, one may also have more than one *trim trajectory*).

In the case of *funnel* libraries, however, it is neither necessary nor sufficient for the nominal trajectories to line up in the non-cyclic coordinates. It is the *interface* between funnels that is important. Let $x_i(t)$ and $x_j(t)$ be two nominal trajectories in our library and $F_i(t) = \{x \mid x \in \mathbb{R}^n,\ V_i(\bar{x}, t) \leq \rho_i(t)\}$ and $F_j(t) = \{x \mid x \in \mathbb{R}^n,\ V_j(\bar{x}, t) \leq \rho_j(t)\}$ be the corresponding funnels. Further, we write $x = [x_c, x_{nc}]$, where x_c represent the cyclic dimensions and x_{nc} the non-cyclic ones. We say that F_i is *sequentially composable* with F_j if

$$\begin{aligned} &\mathcal{P}(F_i(T_i)) \subset \mathcal{P}(F_j(0)) \\ \iff &\forall x = [x_c, x_{nc}] \in F_i(T_i), \quad \exists x_{0,c} \quad s.t. \quad [x_{0,c}, x_{nc}] \in F_j(0). \end{aligned} \tag{3.1}$$

While (3.1) is a sufficient condition for two funnels to be executed one after another, the dependence of $x_{0,c}$ on x makes searching for $x_{0,c}$ a non-convex problem in general. Thus, we set $x_{0,c}$ to be the cyclic coordinates of $x_j(0)$, resulting in a stronger sufficient condition that can be checked easily via a sums-of-squares program:

$$\forall x = [x_c, x_{nc}] \in F_i(T_i), \quad [x_{0,c}, x_{nc}] \in F_j(0). \tag{3.2}$$

Intuitively, (3.2) corresponds to "shifting" the inlet of funnel F_j along the cyclic dimensions so it lines up with x_c. Note that not all pairs of funnels in the library will be sequentially composable in general. Thus, as we discuss in Section 3.1, we must be careful to ensure sequential composability when planning sequences of funnels online. We further note the possibility of computing *continuous families* of funnels around trajectories parameterized by "shifts" of a nominal funnel [25]. This can significantly increase the richness of the funnel library and the chance that a collision-free funnel can be found at runtime. One can also obtain continuously parameterized families of funnels by ensuring that *scalings* of a funnel also result in a valid funnel. We can visualize this as a continuously parameterized nested set of funnel.

3.1 Online Planning with Funnels

Having computed libraries of funnels in the offline pre-computation stage, we can proceed to use these primitives to perform robust motion planning online. 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 easily adapt the contents of this section to the former specification. We further assume that the robot is provided with polytopic regions in configuration space that obstacles are guaranteed to lie in and that the robot's sensors only provide this information up to a finite (but receding) spatial horizon. Our task is to sequentially compose funnels from our library in a way that avoids obstacles while moving forwards in the nominal direction. The finite horizon of the robot's sensors along with the computational power at our disposal determines how long the sequence of planned funnels can be at any given time.

The most important computation that needs to be performed at runtime is to check whether a given funnel intersects an obstacle. For the important case in which our Lyapunov functions are quadratic in x, this computation is a Quadratic Program (QP) and can be solved very efficiently (as evidenced by the success of larger scale QP formulations used in Model Predictive Control [7]). We denote $\bar{x} = x - x_0(t)$ as before, where $x_0(t)$ is the nominal trajectory. Let a particular obstacle be defined by half-plane constraints $A_j x \geq 0$ for $j = 1, ..., M$. Note that A_j will typically be sparse since it will contain zeros in places corresponding to non-configuration space variables (like velocities). Then, for $i = 1, \ldots, N$, we solve the following QP:

$$\begin{aligned} \underset{\bar{x}}{\text{minimize}} \quad & V(\bar{x}, t_i) \\ \text{subject to} \quad & A_j x \geq 0, \forall j \end{aligned} \tag{3.3}$$

Denoting the solution of (3.3) for a given t_i as $V^\star(\bar{x}^\star, t_i)$, the funnel does not intersect the obstacle if and only if $V^{\star}(\bar{x}^{\star}, t_i) > \rho(t_i), \forall t_i$. Multiple obstacles are handled by simply solving (3.3) for each obstacle. An important point that should be noted is that we do not require the obstacle regions to be convex. It is only required that they are represented as unions of convex sets. This allows us to handle situations where multiple polytopic regions overlap to form a non-convex region.

For higher order polynomial Lyapunov functions, one could check the following sums-of-squares conditions for all t_i:

$$\begin{aligned} & V(\bar{x}, t_i) - \rho(t_i) - \sum_j L_j(\bar{x}) A_j(\bar{x} + x_0(t_i)) \text{ is SOS} \\ & L_j(\bar{x}) \text{ is SOS }, \forall j = 1 \dots M \end{aligned} \tag{3.4}$$

However, these provide only sufficient conditions for non-collision. Thus, if the conditions in (3.4) are met, one is guaranteed that there is no intersection with the obstacle. The converse is not true in general. Further, depending on the state-space dimension of the robot, this optimization problem may be computationally expensive to solve online. Hence, for tasks in which online execution speed is crucial, one may need to restrict oneself to quadratic Lyapunov functions.

Algorithm 2 provides a sketch of the online planning loop. At every control cycle, the robot updates its state in the world along with the obstacle positions. It then checks to see if the sequence of funnels it is currently executing may lead to a collision with an obstacle (which should only be the case if the sensors report new obstacles). If so, it replans a sequence of funnels that can be executed from its current state and are collision-free. The $ReplanFunnels(x, \mathcal{O})$ subroutine assumes that funnel sequences that are *sequentially composable* have been ordered by preference during the precomputation stage. For example, for a navigation task, sequences may be ordered by how much progress the robot makes in some nominal direction. $ReplanFunnels(x, \mathcal{O})$ goes through funnel sequences and checks two things. First, it checks that its current state is contained in the first funnel in the sequence (after appropriately shifting the funnel in the cyclic dimensions). Second, it checks that the sequence leads to no collisions with obstacles. The algorithm returns the first sequence of funnels that satisfies both criteria. Finally, the online planing loop computes which funnel of the current plan it is in and applies the corresponding control input $u_i(x, t.internal)$.

Of course, several variations on Algorithm 2 are possible. 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 sequential composability and the moderately more significant cost of collision checking. In practice, it also may not be necessary to consider re-planning at the frequency of the control loop. Instead, longer sections of the plan may be executed before re-planning. Also, instead of choosing the most "preferred" collision-free sequence of funnels, one natural cost function which could guide the search is the minimum over t_i of $\frac{V^\star(\bar{x}^\star, t_i)}{\rho(t_i)}$. As before, $V^\star(\bar{x}^\star, t_i)$ is the solution of the QP (3.3). Since the 1-sublevel set of $\frac{V(\bar{x}, t_i)}{\rho(t_i)}$ corresponds to the funnel, maximizing this is a reasonable choice for choosing sequences of funnels.

Algorithm 2 Online Planning

```
1: Initialize current planned funnel sequence, $\mathcal{F} = \{F_1, F_2, \dots, F_n\}$
2: for $t = 0, \dots$ do
3:     $\mathcal{O} \Leftarrow$ Obstacles in sensor horizon
4:     $x \Leftarrow$ Current state of robot
5:     Collision $\Leftarrow$ Check if $\mathcal{F}$ collides with $\mathcal{O}$ by solving QPs (3.3)
6:     if Collision then
7:         $\mathcal{F} \Leftarrow ReplanFunnels(x, \mathcal{O})$
8:     end if
9:     $F.current \Leftarrow F_i \in \mathcal{F}$ such that $x \in F_i$
10:    $t.internal \Leftarrow$ Internal time of $F.current$
11:    Apply control $u_i(x, t.internal)$
12: end for
```

In order to initialize and replan the sequence of funnels $\mathcal{F}$, it is required that the current state be contained inside the first funnel in the sequence. Assuming perfect state estimates are available, this is easily checked. However, if perfect state information is not available, one needs to ensure that all possible states the system could be in lie inside the funnel. Assuming that measurement errors are bounded, one can use robust state estimation to provide worst-case bounds on the state estimate. For example, [22] provides a way of doing robust state estimation for polynomial discrete time systems via sums-of-squares programming. Given an outer approximation of the set of states the system could be in, one can check that the entire set is contained inside the funnel.

3.2 Example

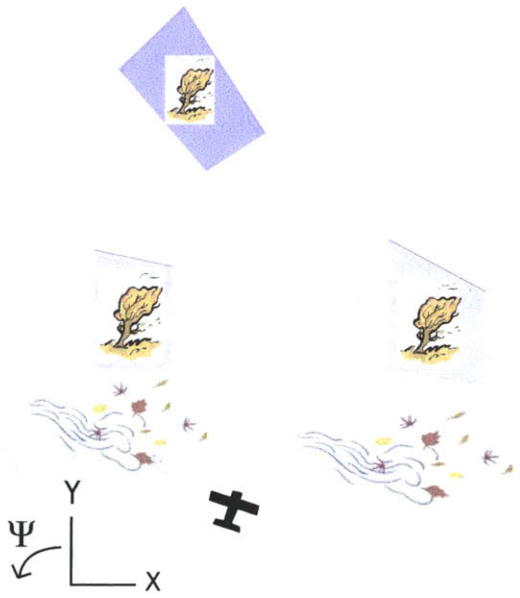


Figure 3-1: Visualization of the system showing the coordinate system, polygonal obstacles, and "cross-wind".

We demonstrate our approach on a model of an aircraft flying in two dimensions through a forest of polygonal trees. A pictorial depiction of the model is provided in Figure 3-1. The aircraft is constrained to move at a fixed forward speed and can control the second derivative of its yaw angle. We introduce uncertainty into the model by assuming that the speed of the plane is uncertain and time-varying and that there is a time-varying "cross-wind" whose magnitude is instantaneously bounded.

The full non-linear dynamics of the system are then given by:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ \dot{\psi} \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} -v(t)\cos\psi \\ v(t)\sin\psi \\ \dot{\psi} \\ u \end{bmatrix} + \begin{bmatrix} w(t) \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{3.5}$$

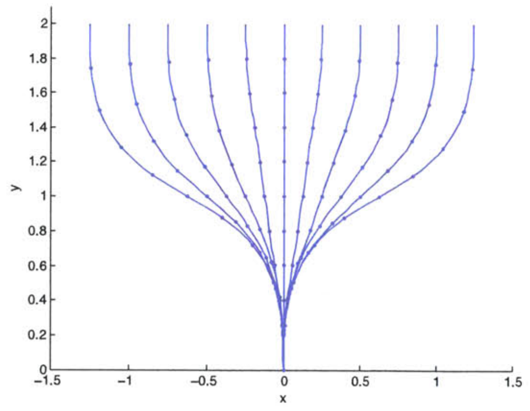
with the speed of the plane $v(t) \in [9.5, 10.5]\ m/s$ and cross-wind $w(t) \in [-0.3, 0.3]\ m/s$. The control input is bounded in the range $[-350, 350]$.

The plane's trajectory library, $\mathcal{T}$, consists of 11 trajectories and is shown in Figure 3-2(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 [4] by constraining $x_i(0)$ and $x_i(T_i)$ and locally minimizing a cost of the form:

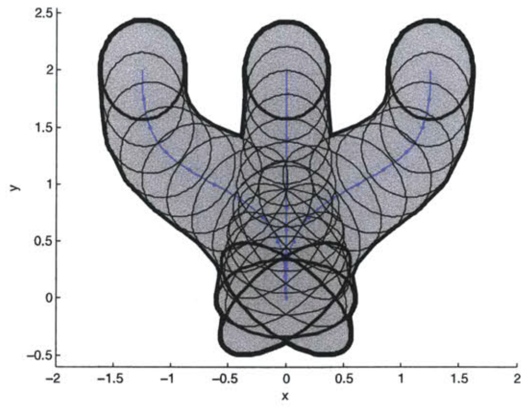
$$J = \int_0^{T_i} [1 + u_0(t)^T R(t) u_0(t)]\, dt.$$

Here, R is a positive-definite cost matrix. For each $x_i(t)$ in $\mathcal{T}$ we obtain controllers and funnels using the method described in Chapter 2. Similar to [39], we perform the verification on the time-varying nonlinear system by taking third-order Taylor-approximations of the dynamics about the nominal trajectories. For each trajectory, we use 11 sample points in time, t_i, for the verification. A 4.1 GHz PC with 16 GB RAM and 4 cores was used for the computations. The time taken for Step 1 of Algorithm 1 during one iteration of the alternation was approximately 10 seconds. Steps 2 and 3 take approximately 45-50 seconds each. Convergence is typically observed within 5 to 10 iterations of the algorithm. Three of the funnels in our library are shown in Figure 3-2(b). Note that the funnels have been projected down from the original four dimensional state space to the x-y plane for the sake of visualization.

Figure 3-3 demonstrates the use of the online planning algorithm in Section 3.1. The plane plans two funnels in advance while nominally attempting to fly in the y-direction and avoiding obstacles. The sensor range allows the plane to sense up to $5m$ ahead. The projection of the full sequence of funnels executed by the plane is shown



(a) Trajectory library consisting of 11 locally optimal trajectories.



(b) Three funnels in our funnel library projected onto the x-y plane.

Figure 3-2: Trajectory and funnel libraries for the plane.

in the figure. Figures 3-3(a) and 3-3(b) show the plane flying through the same forest with identical initial conditions. The only difference is that the cross-wind term is biased in different directions. In Figure 3-3(a), the cross-wind is primarily blowing towards the right, while in Figure 3-3(b), the cross-wind is biased towards the left. Of course, the planner is not aware of this difference, but ends up following different paths around the obstacles as it is buffeted around by the wind.

Finally, we demonstrate the utility of explicitly taking into account uncertainty in Figure 3-4. There are two obstacles in front of the plane. The two options available to the plane are to fly straight in between the obstacles or to bank right and attempt to go around them. If the motion planner didn't take uncertainty into account and simply chose to maximize the average distance to the obstacles, it would choose the trajectory that banks right and goes around the obstacles. However, taking the funnels into account leads to a different decision: going straight in between the obstacles is safer even though the distance to the obstacles is smaller. 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.

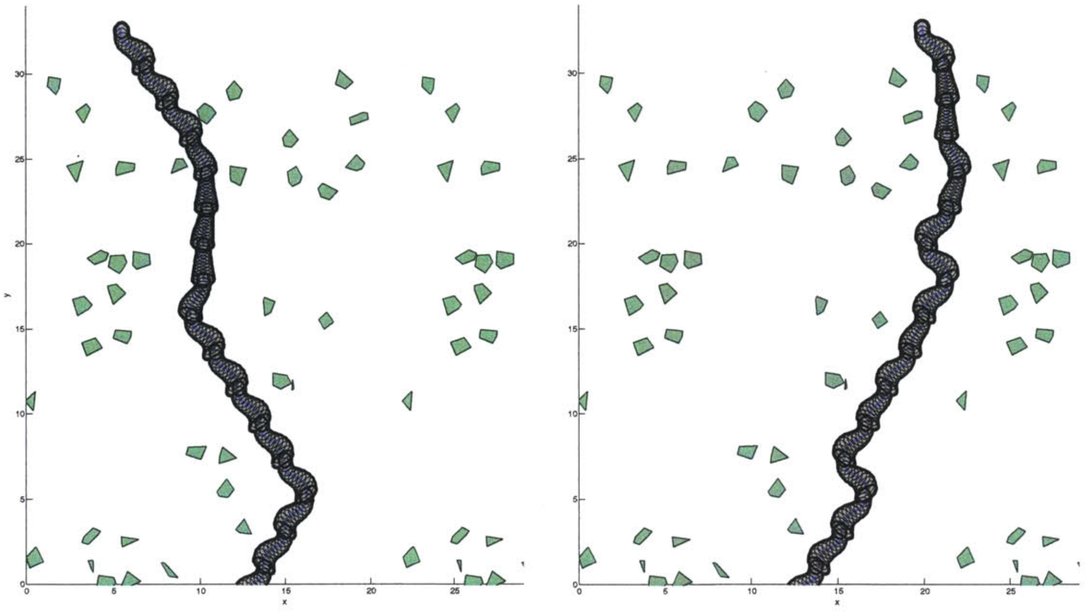

(a) Cross-wind biased towards the left. (b) Cross-wind biased to the right.

Figure 3-3: Robust online planning though a forest of polygonal obstacles. The two subfigures show the plane flying through the same forest, but with the cross-wind biased in different directions (the planner is not aware of this difference). The eventual paths through the forest are significantly different, but the plane navigates the forest safely in each case.

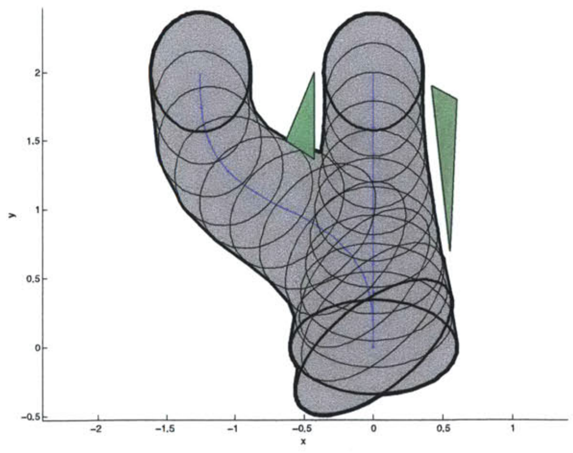


Figure 3-4: This figure shows the utility of explicitly taking uncertainty into account while planning. The intuitively more "risky" strategy of flying in between two closely spaced obstacles is guaranteed to be safe, while the path that avoids going in between obstacles is less robust to uncertainty and could lead to a collision.

Chapter 4

Hardware Validation

The method proposed in this thesis is premised on the assumption that the formal guarantees we compute in terms of funnels for challenging real-world robotic systems are valid on the true hardware platforms for which they are computed (and not just for the idealized model in simulation). This chapter aims to provide evidence that this is in fact possible by carefully evaluating the validity of sums-of-squares (SOS) based funnels on two very different challenging underactuated systems. As far as we know, the results presented here constitute the first experimental validation of sums-of-squares based funnels.

4.1 Acrobot

We first consider the "swing-up and balance" task on a severely torque limited underactuated double pendulum ("Acrobot") [35]. The hardware platform, shown in Figure 4-1, has no actuation at the "shoulder" joint θ_1 and is driven only at the "elbow" joint θ_2. A friction drive is used to drive the elbow joint. While this prevents the backlash one might experience with gears, it imposes severe torque limitations on the system. This is due to the fact that torques greater than 5 Nm cause the friction drive to slip. Thus, in order to obtain consistent performance, it is very important to obey this input limit. Encoders in the joints report joint angles to the controller at 200 Hz and finite differencing and a standard Luenberger observer [21] are used to

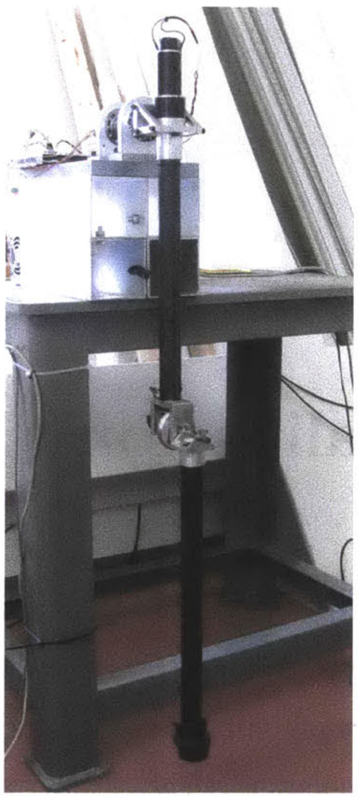
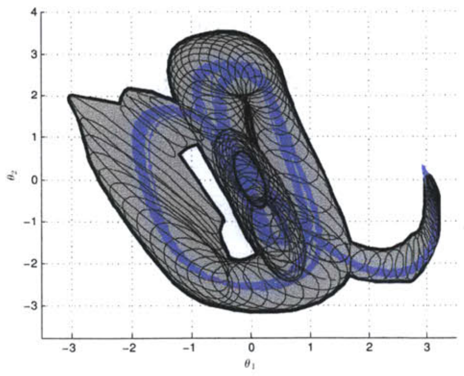


Figure 4-1: The "Acrobot" used for hardware experiments.

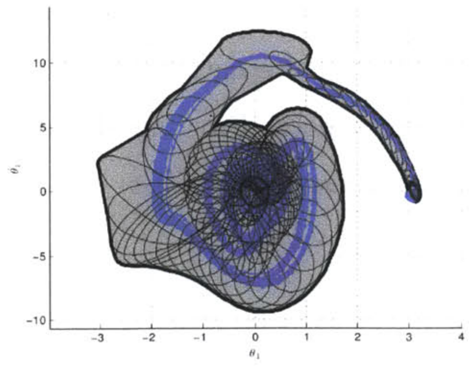
compute joint velocities.

The prediction error minimization method in MATLAB's System Identification Toolbox [19] was used to identify parameters of the model presented in [35]. We designed an open-loop motion plan for the swing-up task using direct collocation trajectory optimization [4] by constraining the initial and final states to $[0, 0, 0, 0]^T$ and $[\pi, 0, 0, 0]^T$ respectively. The dynamics were then Taylor expanded to degree 3 about the nominal trajectory in order to obtain a polynomial vector field.[1] We then designed a time-invariant nonlinear controller (cubic in the four dimensional state $x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^T$) using SOS programming. A linear time-varying controller was designed using the approach presented in Chapter 2 with the goal set given by the verified region of attraction for the time-invariant controller. We modified the

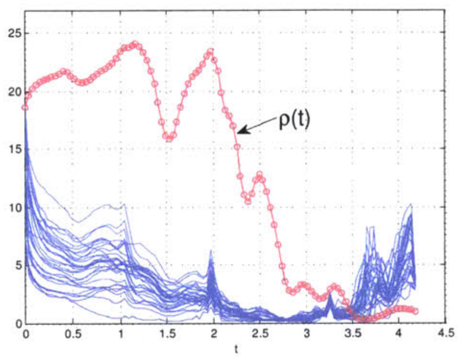
[1]Taylor expanding the dynamics is not strictly necessary since sums-of-squares programming can handle trigonometric as well as polynomial terms [27]. In practice, however, we find that the Taylor expanded dynamics lead to trajectories that are nearly identical to the original ones and thus we avoid the added overhead that comes with directly dealing with trigonometric terms.



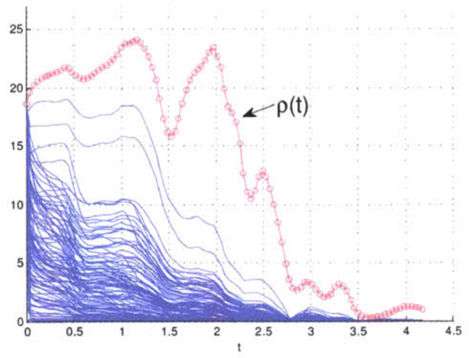
(a) $\theta_2 - \theta_1$ projection of experimental trajectories superimposed on funnel.



(b) $\dot{\theta}_1 - \theta_1$ projection of experimental trajectories superimposed on funnel.



(c) $V(\bar{x}, t)$ for 30 experimental trials



(d) $V(\bar{x}, t)$ for 100 simulated trials

Figure 4-2: Results from experimental trials on Acrobot.

approach presented in Chapter 2 in two ways. First, we do not incorporate any uncertainty in the dynamics. Second, we compute backwards reachable sets instead of reachable sets since the goal set here is fixed (and not the set of initial conditions). However, the computations are almost identical and we refer the reader to [23] for more details. 105 sample points in time, t_i, were used for the verification. For both the time-invariant balancing controller and the time-varying swing-up controller, we use Lyapunov functions, V, of degree 2. LQR controllers were used to initialize the sums-of-squares programs for both controllers.

We implemented our sums-of-squares programs using the YALMIP toolbox [20], and used SeDuMi [38] as our semidefinite optimization solver. A 4.1 GHz PC with 16 GB RAM and 4 cores was used for the computations. The time taken for Step 1 of Algorithm 1 during one iteration of the alternation was approximately 12 seconds.

Steps 2 and 3 took approximately 36 and 70 seconds (per iteration) respectively. 39 iterations of the alternation scheme were required for convergence, although we note that a better method for initializing $\rho(t_i)$ than the one presented in Section 2.2 is likely to decrease this number.

We validate the funnel for the controller obtained from SOS with 30 experimental trials of the Acrobot swinging up and balancing. The robot is started off from random initial conditions drawn from within the SOS verified funnel and the time-varying SOS controller is applied for the duration of the trajectory. At the end of the trajectory, the robot switches to the cubic time-invariant balancing controller. Figures 4-2(a) - 4-2(c) provide plots of this experimental validation. Plots 4-2(a) and 4-2(b) show the 30 trajectories superimposed on the funnel projected onto different subspaces of the 4-dimensional state space. Note that remaining inside the projected funnel is a necessary but not sufficient condition for remaining within the funnel in the full state space. Plot 4-2(c) shows the value of $V(\bar{x}, t)$ achieved during the different experimental trials ($V(\bar{x}, t) < \rho$ implies that the trajectory is inside the funnel at that time). The plot demonstrates that for most of the duration of the trajectory, the experimental trials lie within the verified funnel. However, violations are observed towards the end. This can be attributed to state estimation errors and model inaccuracies (particularly in capturing the slippage caused by the friction drive between the two links) and also to the fact that the Lyapunov function has a large gradient with respect to $\bar{x}$ towards the end. Thus, even though the trajectories deviate from the nominal trajectory only slightly in Euclidean distance (as plots 4-2(a) and 4-2(b) demonstrate), these deviations are enough to cause a large change in the value of $V(\bar{x}, t)$. We note that all 30 experimental trials resulted in the robot successfully swinging up and balancing. Figure 4-2(d) plots $V(\bar{x}, t)$ for 100 simulated experiments of the system started off from random initial conditions inside the funnel. All trajectories remain inside the funnel, suggesting that the violations observed in Figure 4-2(c) are in fact due to modeling and state estimation errors.

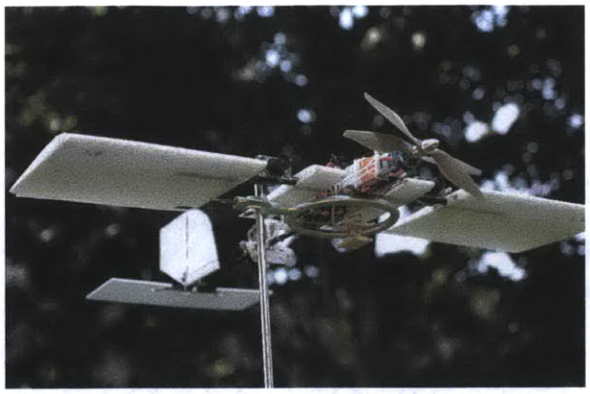


Figure 4-3: The "Wingeron" airplane used for hardware experiments.

4.2 Wingeron Airplane

Next, we consider a more challenging system with 12 state space dimensions and 5 control inputs. The "Wingeron" airplane has been described in [2] and is shown in Figure 4-3. The Wingeron does away with the traditional wing and aileron seen on most aircraft in favor of a wing that is completely actuated; in other words, the entire wing rotates to act as a control surface. This dramatically increases the range of possible roll rates the plane can execute. The states of the plane are the aircraft positions in a global coordinate frame, angles expressed in terms of Euler coordinates, and derivatives of these configuration space variables. The five control inputs are the throttle command, elevator angle, rudder angle, and left/right wingeron angles (the wingerons are actuated independently).

The prediction error minimization method in MATLAB's System Identification Toolbox is again used to identify parameters for an aerodynamic model taken from [34]. We designed an open-loop motion plan for a dynamic roll maneuver that lasts 0.32 seconds. A time-varying LQR controller is designed to stabilize this trajectory. A funnel is computed for this *fixed* controller using the methods presented in Chapter 2, and is depicted in Figure 4.2. The funnel computation takes approximately 15 minutes.

We performed experiments in a motion capture arena and again use finite differencing and a standard Luenberger observer to compute the derivative terms of the state. The plane is launched at approximately $7.5m/s$ from a launcher and flies into

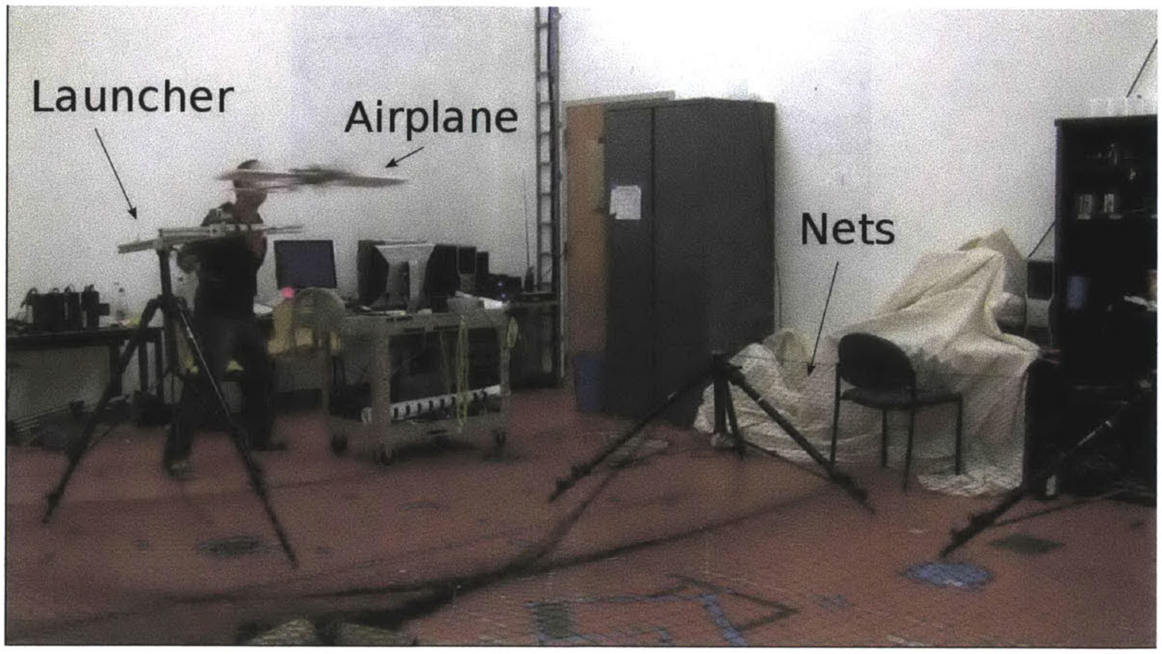
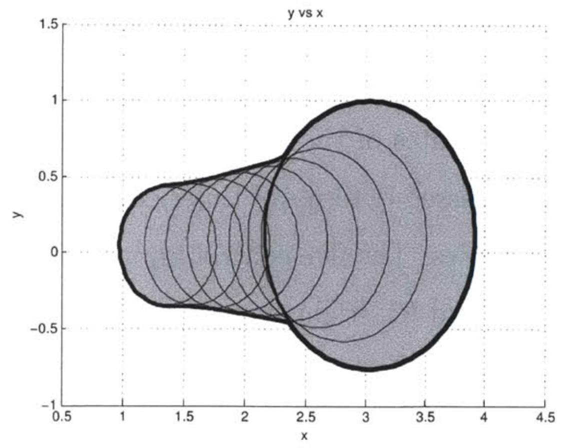
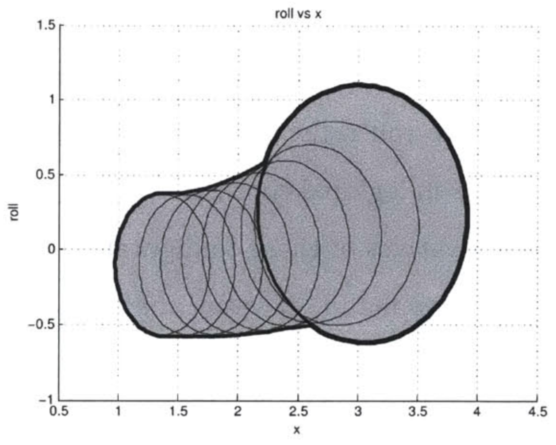


Figure 4-4: The experimental setup used for experiments with the Wingeron plane.



(a) Funnel slice in x-y subspace.



(b) Funnel slice in x-roll subspace.

Figure 4-5: Two dimensional slices of funnels computed for the Wingeron plane

a net once the trajectory is completed. The experimental setup is shown in Figure 4-4.

Figure 4-6 plots the value of the Lyapunov function V with time t for 5 different experimental trials on the hardware, each of which starts off inside the verified funnel. Here, the one sublevel set defines the boundary of the funnel. As the plot demonstrates, each of the 5 trajectories remains within the computed funnel for the

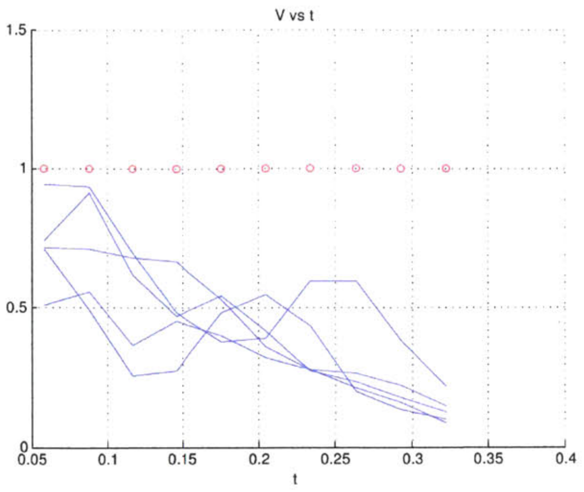


Figure 4-6: A plot showing V vs t for five experimental trials on the airplane. The one sublevel set (indicated in red) defines the boundary of the funnel. All trajectories stay within the computed funnel.

entire duration of the trajectory. While the number of trajectories considered here is relatively small, these results are promising and provide evidence for the claim that the online planning framework presented in this thesis can work on challenging real-world hardware platforms with complicated nonlinear dynamics.

Chapter 5

Conclusion and Future Work

5.1 Modifying funnels with data

As we saw in Chapter 4, while the funnels we obtain from sums-of-squares programming are largely faithful to the real hardware, there can still be some discrepancies. One natural way to address this is to use the robust verification methods presented in Chapter 2. However, in some cases, it may be difficult to explicitly parameterize the set of disturbances/uncertainty. In this setting, another possible approach is to use data obtained from real hardware to update a precomputed funnel. For example, suppose one observes that a particular trajectory violates the computed funnel. Then, we can incorporate this new piece of data by *re-solving* the sums-of-squares program (2.3) in Chapter 2 with the additional constraint that this trajectory remains within the new funnel. Implementing this data driven approach to adjusting funnels and proving generalization bounds on such a scheme are the subject of current work.

5.2 Stochastic Verification

Throughout this thesis, 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 over-conservative performance. In such situations, it is more natural to provide guarantees of reachability of a probabilistic nature. Recently, results from classical martingale theory have been combined with sums-of-squares programming in order to compute such probabilistic certificates of finite time invariance [36], i.e. provide upper bounds on the probability that a stochastic nonlinear system will leave a given region of state space. The results presented in [36] can be directly combined with the approach presented in this work to perform robust online planning on stochastic systems and will be the subject of future work.

5.3 Continuously Parameterized Families of Funnels

As discussed in Chapter 2, we are currently partially exploiting invariances in the dynamics by shifting trajectories (and corresponding funnels) that we want to execute next in the cyclic coordinates so they line up with the cyclic coordinates of the robot's current state. In our example from Section 3.2, this simply corresponds to translating and rotating funnels so the beginning of the next trajectory lines up with the current state's x,y and yaw. However, we could further exploit invariances in the dynamics by shifting funnels around locally to ensure that they don't intersect an obstacle while still maintaining the current state inside the funnel. One can then think of the nominal trajectories and funnels being *continuously parameterized* by shifts in the cyclic coordinates. Interestingly, it is also possible to use sums-of-squares programming to compute conservative funnels for cases in which one shifts the nominal trajectory in the *non-cyclic* coordinates [25]. Thus, one could potentially significantly improve the richness of the funnel library by pre-computing continuously parameterized funnel libraries instead of just a finite family. However, choosing the right "shift" to apply at runtime is generally a non-convex problem (since the free-space of the robot's environment is non-convex) and thus one can only hope to find "shifts" that are locally optimal.

Another way to obtain continuously parameterized funnels is to ensure that *scalings* of $\rho(t)$ result in sub-level sets of $V(\bar{x}, t)$ that are invariant. This is equivalent to computing outer approximations of reachable sets for scalings of the initial condition set. Specifically, the SOS program (2.3) can be modified to guarantee that for some $\epsilon > 0$, $\forall c \in [\epsilon, 1]$ the sub-level sets defined by $c\rho(t)$ are invariant. One can visualize this as a continuously parameterized nested set of funnels. At runtime, this allows us to choose from scalings of our funnels. This could potentially reduce the number of funnels we need in our library. However, in practice, it may be difficult to obtain controllers and funnels that make guarantees of this form, especially if the disturbance terms are large.

5.4 Sequence optimization for Large Funnel Libraries

For extremely large funnel libraries, it may be computationally difficult to search all the funnels while planning online. This is a problem that traditional trajectory libraries also face [8]. Advances in submodular sequence optimization were leveraged in [8] to address this issue. The approach involves limiting the set of trajectories that are evaluated online and optimizing the sequence in which trajectories are evaluated. Guarantees are provided on the sub-optimality of the resulting strategy. This technique could be adapted to work in our framework too and will be addressed in future work.

5.5 Designing Funnel Libraries

One issue that we have not addressed in this thesis is the choice of motion primitives in our library. While there has been considerable work on designing trajectory libraries (see Section 1.1), designing *funnel libraries* poses challenges that go beyond just choosing a good set of nominal trajectories. The effect of uncertainty and feedback must be taken into account while constructing the library. One interesting problem domain in which it may be possible to design funnel libraries in a principled way is the case where the statistics of obstacle positions are known *a priori* (but the actual positions are unknown). An example of such a scenario is the task considered in this thesis: autonomous UAV flight through a forest. It is known that the location of trees in a forest is well modeled by Poisson distributions [15]. Another example is legged robot locomotion on rough terrain, where the statistics of terrain variations are known beforehand. In such scenarios, it may be possible to design a randomized algorithm in the spirit of LQR-Trees [39] where one attempts to plan paths through particular realizations of the environment by sequencing funnels together and adding a funnel to the library every time a collision free sequence of funnels is not found in the existing library. Under certain assumptions on the distributions of obstacles (e.g. stationarity, ergodicity), it is conceivable that such a randomized algorithm may be probabilistically complete.

5.6 Conclusion

In this thesis, we have presented an approach for motion planning in *a priori* unknown environments with dynamic uncertainty in the form of bounded parametric model uncertainty, disturbances, and state errors. The method augments the traditional trajectory library approach by constructing stabilizing controllers around the nominal trajectories that explicitly attempt to minimize the size of the reachable set of the system subjected to disturbances and uncertainties. The precomputed set of reachable sets ("funnels") is then used to plan online by *sequentially composing* them together in a manner that ensures obstacles are avoided. By explicitly taking into account uncertainty and disturbances while making motion plans, we can evaluate trajectory sequences based on how susceptible they are to disturbances. We have demonstrated our approach on a simulation of a plane flying in two dimensions through a forest of polygonal obstacles. We have further validated the approach by providing hardware experiments on an Acrobot system and an airplane that evaluate the guarantees (in terms of funnels) computed via sums-of-squares programming. Future work will focus on generating funnel libraries automatically for environments with known obstacle distributions (e.g. forests) and extending our results to scenarios in which a stochastic description of uncertainty is more appropriate.

5.7 Funding

This work was supported by ONR MURI grant N00014-09-1-1051, the MIT Intelligence Initiative, and a Siebel Foundation Scholarship.

Bibliography

[1] A. A. Ahmadi. Non-monotonic Lyapunov Functions for Stability of Nonlinear and Switched Systems: Theory and Computation. Master's thesis, Massachusetts Institute of Technology, June 2008.

[2] Andrew J. Barry. Flying between obstacles with an autonomous knife-edge maneuver. Master's thesis, Massachusetts Institute of Technology, Sep 2012.

[3] 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.

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

[5] 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.

[6] 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.

[7] E. F. Camacho and Carlos Bordons. *Model Predictive Control.* Springer-Verlag, 2nd edition, 2004.

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

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

[10] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Real-Time Motion Planning for Agile Autonomous Vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, JanuaryFebruary 2002.

[11] J.H. Gillula, H. Huang, M.P. Vitus, and C.J. 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.

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

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

[14] A. Agung Julius and George J. Pappas. Trajectory based verification using local finite-time invariance. In *HSCC '09: Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control*, pages 223–236, Berlin, Heidelberg, 2009. Springer-Verlag.

[15] Karaman and Frazzoli. High-speed flight through an ergodic forest. In *IEEE Conference on Robotics and Automation*, 2012.

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

[17] J.J. Kuffner and S.M. 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.

[18] C. Liu and C.G. 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.

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

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

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

[22] C. Maier and F. Allgower. A set-valued filter for discrete time polynomial systems using sum of squares programming. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 223–228. IEEE, 2009.

[23] Anirudha Majumdar, Amir Ali Ahmadi, and Russ Tedrake. Control design along trajectories with sums of squares programming. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[24] Anirudha Majumdar and Russ Tedrake. Robust online motion planning with regions of finite time invariance. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, 2012.

[25] Anirudha Majumdar, Mark Tobenkin, and Russ Tedrake. Algebraic verification for parameterized motion planning libraries. In *Proceedings of the 2012 American Control Conference (ACC)*, 2012.

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

[27] A. Megretski. Positivity of trigonometric polynomials. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, Dec 2003.

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

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

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

[31] T. Schouwenaars, B. Mettler, E. Feron, and J.P. How. Robust motion planning using a maneuver automation with built-in uncertainties. In *American Control Conference, 2003. Proceedings of the 2003*, volume 3, pages 2211–2216. IEEE, 2003.

[32] Pierre Sermanet, Marco Scoffier, Chris Crudele, Urs Muller, and Yann LeCun. Learning maneuver dictionaries for ground robot planning. In *Proc. 39th International Symposium on Robotics (ISR08)*, 2008.

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

[34] Sobolic and Frantisek M. 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.

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

[36] Jacob Steinhardt and Russ Tedrake. Finite-time regional verification of stochastic nonlinear systems. In *Proceedings of Robotics: Science and Systems (RSS) 2011*, January 17 2011.

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

[38] Jos F. Sturm. Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625 – 653, 1999.

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

[40] Mark M. Tobenkin, Ian R. Manchester, and Russ 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.