Control, Estimation, and Planning Algorithms for Aggressive Flight using Onboard Sensing

by

Adam Parker Bry

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2012

Author ....

Department of Aeronautics and Astronautics
February 2, 2012

Certified by ....

Nicholas Roy
Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by ....

Eytan H. Modiano
Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Control, Estimation, and Planning Algorithms for Aggressive Flight using Onboard Sensing

by

Adam Parker Bry

Submitted to the Department of Aeronautics and Astronautics
on February 2, 2012, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

This thesis is motivated by the problem of fixed-wing flight through obstacles using only on-board sensing. To that end, we propose novel algorithms in trajectory generation for fixed-wing vehicles, state estimation in unstructured 3D environments, and planning under uncertainty.

Aggressive flight through obstacles using on-board sensing involves nontrivial dynamics, spatially varying measurement properties, and obstacle constraints. To make the planning problem tractable, we restrict the motion plan to a nominal trajectory stabilized with an approximately linear estimator and controller. This restriction allows us to predict distributions over future states given a candidate nominal trajectory. Using these distributions to ensure a bounded probability of collision, the algorithm incrementally constructs a graph of trajectories through state space, while efficiently searching over candidate paths through the graph at each iteration. This process results in a search tree in belief space that provably converges to the optimal path. We analyze the algorithm theoretically and also provide simulation results demonstrating its utility for balancing information gathering to reduce uncertainty and finding low cost paths.

Our state estimation method is driven by an inertial measurement unit (IMU) and a planar laser range finder and is suitable for use in real-time on a fixed-wing micro air vehicle (MAV). The algorithm is capable of maintaining accurate state estimates during aggressive flight in unstructured 3D environments without the use of an external positioning system. The localization algorithm is based on an extension of the Gaussian Particle Filter. We partition the state according to measurement independence relationships and then calculate a pseudo-linear update which allows us to use 25x fewer particles than a naive implementation to achieve similar accuracy in the state estimate. Using a multi-step forward fitting method we are able to identify the noise parameters of the IMU leading to high quality predictions of the uncertainty associated with the process model. Our process and measurement models integrate naturally with an exponential coordinates representation of the attitude uncertainty. We demonstrate our algorithms experimentally on a fixed-wing vehicle flying in a challenging indoor environment.

The algorithm for generating the trajectories used in the planning process computes a transverse polynomial offset from a nominal Dubins path. The polynomial offset allows us to explicitly specify transverse derivatives in terms of linear equality constraints on the coefficients of the polynomial, and minimize transverse derivatives by using a Quadratic Program (QP) on the polynomial coefficients. This results in a computationally cheap method for generating paths with continuous heading, roll angle, and roll rate for the fixed-wing vehicle, which is fast enough to run in the inner loop of the RRBT.

Thesis Supervisor: Nicholas Roy
Title: Assistant Professor of Aeronautics and Astronautics

Acknowledgments

I would like to thank my advisor Nicholas Roy for giving me the freedom to shape this project while providing his guidance and sharp insight along the way. Thanks to the other members of the Robust Robotics Group and in particular thanks to Abraham Bachrach for being a constant sounding board for algorithmic ideas and software development, and for his direct contributions to the state estimation work in this thesis.

Thanks are due to the Franklin W. Olin Foundation and the entire Olin community for supporting my undergraduate education and setting me down the path that led this work.

My parents, Douglas Bry and Jill Parker, and sister, Leah Bry, are responsible for a foundation of support and a drive towards intellectual inquiry. Any strength in this work is reflective of that foundation and that drive.

Finally, I would to thank my friends, roommates, and the members of the MIT Cycling Team for giving me so much to live for outside of the lab.

Contents

Chapter 1

Introduction

Advances in system identification, control, and planning algorithms increasingly make it possible for autonomous flying vehicles to utilize the full scope of their natural dynamics. Quad-rotors capable of agile flight through tight obstacles, helicopters that perform extended aerobatic sequences, and fixed-wing vehicles that mimic the perching behavior of birds, have all been reported in the literature [14, 16, 42]. Simultaneously, advances in LIDAR and computer vision algorithms have made autonomous flight through obstacle-rich environments possible without the use of an external sensor system [4, 3]. Currently, there is growing interest in extending the highly dynamic maneuvers that have been demonstrated when accurate state information is always available to autonomous vehicles that operate in unstructured environments using only on-board sensors.

The central problem that motivates the work in this thesis is the autonomous flight of a small, unmanned, fixed-wing vehicle through an obstacle rich environment using only on-board sensors. This technology would represent a substantial step forward in the state of the art for autonomous systems and be a useful enhancement for the capabilities of existing micro air vehicles (MAVs) which are increasingly deployed in military, search and rescue, and security roles around the world. As we will see, the problem presents unique challenges from a control, state estimation, and planning perspective.

A key differentiator between this problem, and the closely related problem of

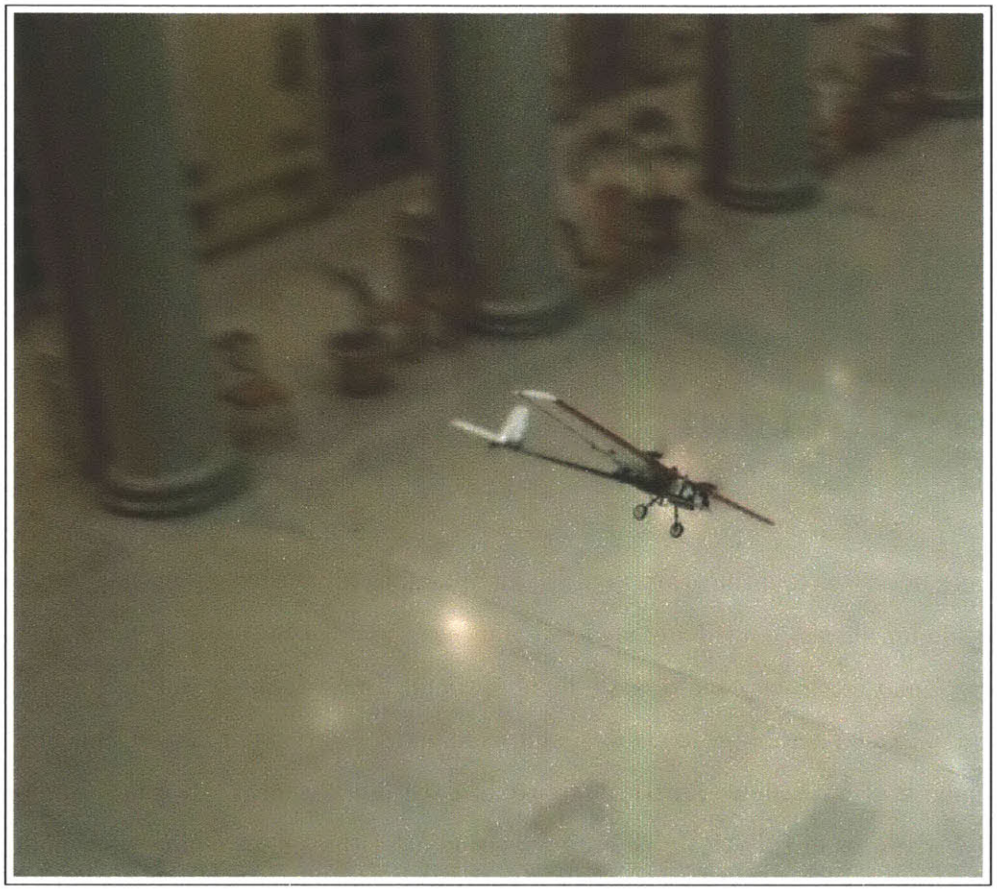


Figure 1-1: Fixed wing experimental platform flying indoors localizing using an on board laser range scanner and inertial measurement unit.

quad-rotor flight through obstacles using on-board sensors is the non-trivial vehicle dynamics. Because helicopters are capable of hovering in place, many of the same algorithms that have advanced the capabilities of ground robots in the last twenty years can be adapted for use in the hover regime [4]. Such is not the case for fixed-wing vehicles. While they share the payload and computation constraints with quad-rotors, a fixed-wing vehicle must maintain a forward velocity to lift the payload, and that velocity scales with the payload. Thus the more computation and sensing power on board an airplane, the faster it must fly for a fixed wing size. This constraint on the minimum velocity places a substantial design burden on anything that must run in real-time on the vehicle: namely the state estimation and feedback control algorithms. Further, most of the work with quad-rotors using laser range scanners (LIDARs) makes strong 2.5D assumptions about the environment. These assumptions are reasonable on a quad-rotor since the vehicles does not depart from the hover regime where the sensor is close to level in the x-y plane. For a fixed-wing vehicle, maneuvering requires departing the plane - at least in roll, and the tighter and more aggressive the maneuvering, the larger the magnitude of the departure.[1] True 3D localization using only a planar LIDAR is thus a primary challenge for our vehicle.

One could argue that state estimation is the most significant, direct challenge faced by a fixed-wing vehicle flying through obstacles using only on-board sensing. However, difficult state estimation translates directly into challenges in planning. When using on-board sensors, the ability of the vehicle to estimate its position depends on both the position itself and its velocity as obstacles and environmental features come in and out of view of the planar LIDAR. An aggressive bank angle may make localization in the horizontal dimensions difficult or impossible altogether, but allow the vehicle to get accurate height information as LIDAR beams hit the ground. Motion blur and other speed effects cause similar state-dependent localization problems for camera-based sensing if the vehicle executes maneuvers with high velocity or angular rates. Failure to reason intelligently about the state-dependent uncertainty during planning could lead to an unacceptable probability of crashing. A successful planner must have knowledge of what future state estimates may occur, in terms of both mean and uncertainty - the "belief space" of the vehicle. It must make plans that are appropriately cautious when the vehicle will be uncertain about its position while executing trajectories and maneuvers that allow the vehicle to gather information and improve its state estimate when advantageous in terms of overall path cost. By seeking low-cost or optimal paths, information gathering and caution around obstacles become tightly coupled problems. Much previous work in planning under uncertainty focuses solely on finding maximum information or minimum uncertainty paths [24]. In contrast, we are interested in gathering information and reducing uncertainty only

[1]2.5D means the environment is represented as an extrusion in three dimensions of a two-dimensional plan

if it allows the vehicle to reach the goal more efficiently: if the vehicle can fly a shorter trajectory by temporarily being very uncertain of its location, but an accurate state estimate will be recovered before flying close to an obstacle, that is desirable behavior.

To solve the planning problem we propose an asymptotically optimal sampling-based approach based on recent advances in sampling based motion planning [29]. Sampling-based approaches — most commonly variants of the Rapidly-exploring Random Tree (RRT) or Probabalistic Road Map (PRM)— are widely used in robotics: they are easy to implement, computationally efficient, and are often probabilistically complete in the sense that if a solution exists it will be found with probability 1 given enough samples. The algorithm we propose in this thesis, the Rapidly-exploring Random Belief Tree (RRBT) works by incrementally constructing a graph of feasible trajectories through state space and then "propagates" uncertainty along the edges of the graph by simulating state estimation and control along the paths. While powerful, this framework introduces additional constraints on the state estimation and control algorithms.

To efficiently propagate uncertainty, we use a Gaussian representation of the state distribution, such that state estimates are fully characterized by a mean and covariance matrix. Unfortunately, laser range measurements in three dimensions are generally far from Gaussian in their noise properties. However, the state estimation algorithm we propose extends a technique called the Gaussian Particle Filtering (GPF) to use particles (weighted samples in the state space) to capture the non-Gaussian, non-linear laser measurements as Gaussians. This not only makes for a computationally efficient state estimator, but also allows these pseudo-Gaussian measurements to be used efficiently during planning.

Sampling-based approaches work by sampling states and connecting them with feasible trajectories. The approach we use, an extension of the RRT* algorithm (a recent version of the RRT which asymptotically converges to the optimal solution) requires that these connections to sampled states be made exactly. Exact connections ensure that if a lower cost path is discovered to some point in state space, the lower cost alternative trajectory can be substituted in without disturbing the subsequent trajectory. However, generating exact connections is itself an algorithmic challenge. For our planning algorithm to be successful we must have a method of generating kinodynamically feasible trajectories between two arbitrary states, and this method must be computationally efficient since it will be used each time a state is sampled. We introduce a technique for generating paths based on a differentially flat model of coordinated-flight vehicle dynamics. Differential flatness implies system inputs and states are fully determined by a set of outputs and output derivatives. For the coordinated flight model, the output is the spatial trajectory. This model allows us to generate trajectories using transverse polynomial offsets from a nominal path which minimize the control effort required to follow that path.

Finally, since stochasticity factors prominently into the vehicle dynamics and sensing, we must use a feedback control law to stabilize the planned trajectory. In planning we propagate Gaussian uncertainty. To maintain a Gaussian, the dynamics along the path must be linear which is generally not the case for fixed-wing vehicles. However, the differentially flat vehicle model allows us to achieve approximately linear error dynamics, meaning the uncertainty prediction used in planning can more closely match what can be expected during path execution, as the RRBT planning algorithm uses closed-loop predictions of the state uncertainty.

Thus the central problem of fixed-wing flight through obstacles motivates algorithmic development in state estimation, planning, trajectory generation, and control. Furthermore, we can see that choices and design constraints in one algorithm can introduce choices and design constraints in the others. This thesis introduces algorithms for each of these pieces, which while all independently useful and potentially broadly applicable, are specifically motivated and designed to work as a unified system.

1.1 Contributions

This thesis makes contributions in trajectory generation (chapter 3), state estimation (chapter 4), and planning under uncertainty (chapter 5). We outline the contributions below.

1.1.1 Belief Space Planning for Dynamic Systems

The central and unifying contribution of this thesis is a novel algorithm that extends a class of recently proposed incremental sampling-based algorithms to handle state-dependent stochasticity in both dynamics and measurements [29]. The key idea is in leveraging the fact that the incremental sampling approach allows us to enumerate all possible paths through an environment to find paths that optimally trade off information gathering, avoiding obstacles, and quickly reaching the goal. To ensure the plan is suitable for systems with nontrivial dynamics, we evaluate paths with a probabilistic distribution over all possible trajectories that may be realized while following a path with a closed-loop controller. The algorithm proceeds by incrementally constructing a graph of feedback stabilized trajectories through state space, while efficiently searching over candidate paths through the graph as new samples are added. We provide a pruning technique that exploits specific properties of uncertainty propagation for eliminating possible paths and terminating search after each sample is added. In the limit, this process results in a tree in belief space (the space of probability distributions over states) that contains the optimal path in terms of minimum cost with a bounded probability of collision or "chance-constraint".

Both sampling-based algorithms and linear control and estimation schemes have been shown to scale well with dimensionality. Additionally, the algorithm has the powerful property offered by incremental sampling algorithms of quickly exploring the space and then provably converging to the optimal solution. This is particularly desirable for stochastic planning problems since they are computationally demanding and in a real-time setting the computational time available could vary widely.

1.1.2 State Estimation

The wide disparity between what is possible in terms of agile flight with an external positioning system and what has been demonstrated with onboard sensing suggests that state estimation from on board sensors is indeed a significant challenge in extending the capabilities of MAVs in real world environments. In addition to providing accurate estimates, the state estimation system must also accurately identify its uncertainty in order to enable planning with applicable partially observable motion planning algorithms.

This thesis presents a state estimation method that is suitable for use in real-time on a fixed wing MAV maneuvering through a cluttered environment. Our system leverages an inertial measurement unit (gyros and accelerometers) and a planar laser range finder in a filtering framework that provides the accuracy, robustness, and computational efficiency required to localize a MAV within a known 3D occupancy grid map.

Our process model is a based on an exponential-coordinates extended Kalman filter that is driven by inertial measurements. Unlike conventional system identification techniques that estimate model parameters from sensor data labeled with ground truth states, the model parameters of the IMU process model are estimated using an algorithm that does not require access to ground truth data. In order to efficiently project the nonlinear laser measurement update of the vehicle position back through the state estimate, we integrate the laser range-finder measurement as a pseudo measurement computed from a Gaussian Particle Filter state update in a lower dimensional measurement space. The use of the lower dimensional space drastically reduces the number of particles required, thereby enabling realtime performance in the face of the computational limitations of the flight computer. We demonstrate the effectiveness of our approach experimentally on a fixed wing vehicle being piloted in a challenging GPS-denied environment.

1.1.3 Trajectory Generation

The RRBT planning algorithm requires that we be able to make exact connections between sampled states. We develop an algorithm based on parameterizing a transverse polynomial offset from a nominal path which allows us to explicitly optimize and specify the transverse derivatives of that path at intermediate points along the trajectory and at specified start and end locations. For fixed-wing vehicles, these transverse derivatives map directly to heading (first derivative) roll angle (second derivative or curvature), roll rate (third derivative or derivative of curvature), and roll acceleration (fourth derivative, second derivative of curvature). Our method allows us to generate paths that are smooth up to the derivative of curvature (roll rate) and could be extended to higher derivatives if desired.

By explicitly specifying the transverse derivatives at points along a path, we can "correct" for discontinuities in the underlying path. The ability to match derivatives between segments allows us to use Dubins curves which are continuous only up to heading, but can be generated very quickly using simple geometric calculations, as the nominal paths.[2] We optimize the transverse offset to minimize quadratic cost on the offset and its derivatives which translates into approximate minimization of the control effort - namely roll rate and roll acceleration - necessary to follow the paths, Furthermore, since the transverse offset is represented as a polynomial, the quadratic minimization takes the form of a Quadratic Program (QP) with approximately linear constraints.

We present a solution method based on an initial joint optimization of polynomials for all the segments (lines and arcs) composing a Dubins path with linear constraints between the segments and at the start and end of the path. We then use correction expressions that capture the nonlinearaties in the transverse derivatives induced by the offset, to setup a second optimization for each segment individually to ensure the derivative smoothness of the final path. Both optimization steps take the form of a QP with linear constraints which can be solved in a single step using variable elimination. Thus kinodynamically feasible paths for the fixed wing vehicle representing exact connections in state space are achieved using single-step matrix inversions on matrices with dimension of the order to the polynomial. This computational efficiency makes the trajectory generation suitable for use in the RRBT algorithm.

[2]While we demonstrate the technique with Dubins curves, transverse polynomial paths could be used for any nominal path including concatenated line segments.

1.2 Thesis Overview

In chapter 2 we survey the literature and give an overview of the prior work that has been done in each of the areas we address algorithmically. Chapter 3 presents the trajectory generation and accompanying control algorithm used for planning. In chapter 4 we describe the state estimation algorithm and present results obtained from hand-flown flight experiments in a challenging indoor environment. Chapter 5 develops the RRBT algorithm, including theoretical proofs for the completeness and asymptotically optimal behavior of the algorithm under reasonable technical assumptions. We demonstrate the algorithm on example problems with single integrator and Dubins dynamics with polygonal obstacles, and we also provide simulation results for plans generated with transverse-polynomial paths and GPF measurements computed in a real three dimensional map of an urban environment. Finally, in 6 we conclude and discuss future work.

Chapter 2

Related Work

In this chapter we survey the literature relevant to the problems addressed in this thesis. The literature survey is organized in much the same way as the thesis itself. We begin by looking at work with similar high level goals: autonomous flight through obstacles. We then analyze related work in each of the algorithmic areas this thesis addresses: trajectory generation, state estimation, and finally motion planning, and specifically motion planning under uncertainty.

2.1 Flight Through Obstacles

The potential benefits of MAVs that can operate safely and reliable in and around obstacles has attracted considerable attention in the research community. The bulk of the work has been done with helicopters, either conventional variable pitch configurations or quad-rotor configurations, but more recently some work has been done with fixed-wing vehicles.

Bachrach, He, Prentice, and Roy present one example of a completely autonomous quad-rotor capable of operating in unknown and unstructured environments without the use of GPS [4]. Many of the design decisions in the algorithms presented in this thesis are motivated by this system. The core enabling technology is a high frequency scan matcher that uses successive laser scans from the Hokuyo LIDAR to estimate planar velocity which can be integrated over impressively long time scales to position. In conjunction with the on-board IMU, the scan matcher forms the backbone of the quad's state estimator, which can then be integrated with higher level full simultaneous localization and mapping (SLAM) solutions [54]. Other quad-rotor systems for flying in GPS-denied environments use very similar system architecture [52, 18].

While this framework is powerful and has resulted in impressive experimental demonstrations, it is based on flying close to the hover regime where motion planning can be done kinematically (motion plans are essentially constructed as simple line segments) and both estimation and planning are strongly dependent on 2.5D world assumptions. In contrast, our work is aimed at eliminating these assumptions with fully 3D state estimation and trajectory generation and planning that explicitly account for and utilize non-trivial fixed-wing dynamics.

Scherer et al. developed a helicopter system capable of flying at high speeds (10 m/s) around obtacles based on high accuracy differential GPS combined with a suite of on-board sensors and collision avoidance algorithms [50]. In contrast, we are interested in enabling this kind of behavior without the use of an external system such as GPS, and with a sensor payload more constrained by the physics of fixed-wing flight.

Much of the experimental work done with fixed-wing vehicles flying in close proximity to obstacles using on-board sensing is based on optical flow from visual sensors [9, 59]. Much of this work is inspired by the systems insects use to navigate [15]. The essentially idea is a direct mapping from sensory input to control actions. Optical flow is calculated by estimating pixel motion between camera frames, creating a vector field in image space. Areas of the image where the flow is large in magnitude represent potential close proximity to obstacles, and thus obstacle avoidance can be accomplished by steering away from areas of high optical flow. Such systems have been shown to be stable flying through corridor type environments.

While impressive in many respects, this kind of system is inherently limited. Because sensing is mapped directly to control output, no internal state estimate of the vehicle is maintained. This is highly computationally efficient for MAVs as small as

10 grams [59], however, the absence of an internal state estimate makes global, higher level reasoning difficult. For example it would be difficult to give such a system a command such as "go from point A to point B as quickly as possible while avoiding too high a probability of collision" and have certainty that it could accomplish this task. Furthermore, optical flow has a singularity directly in front of the vehicle — no matter how fast you fly directly at an obstacle, its position in the image frame does not change, which presents an obvious problem for many situations when flying around obstacles since the system is essentially blind to what is directly in front of it.

When highly accurate, off-board sensing systems are used for positioning, some extremely impressive, agile flight has been demonstrated. In particular Mellinger et al. designed a system capable of aggressively flying quad-rotors through narrow slits [43]. The system is based on a detailed vehicle dynamics model and the controller and trajectories are hand-tuned and iteratively refined through experiments. All of the work is done in a VICON motion capture system which provides real-time, millimeter-level-accuracy position and orientation of the vehicle at rates over 100Hz. In the fixed-wing regime, Cory et al. designed a system capable of mimicking the perching behavior of birds with small foam gliders [16]. Multiple control methodologies were shown to be successful in achieving perching behavior, but once again all of the work relied on a VICON motion capture system. Both of these systems serve as inspiration for our work in terms of what MAVs are dynamically capable of, but the reliance on on-board sensing necessarily differentiates the algorithms in this thesis.

2.2 Trajectory Generation

The general problem of "steering" a dynamics system from one state to another is the subject of a great deal of work in optimal control. For very simple linear systems the solution can be obtained as the solution of a linear system of equations [8]. In contrast, in the most general case for nonlinear systems, solutions can be computed via shooting methods or direcy collocation [57]. While powerful, these methods rely on decomposing a trajectory into a discrete "control tape" of actions which are played back to execute the trajectory, and the entire control tape is then optimized to satisfy the boundary conditions and minimize an objective function. In general, this results in a non-convex, nonlinear optimization problem with dimension equal to the number of control steps which can be enormous depending on the resolution required by the system. Tedrake uses similar techniques to drive a sampling-based motion planning system, but that work is specifically designed to maintain a sparse tree with feedback control laws, such that the number of trajectories that need to be computed is small (since they are computationally expensive). In controls, the asymptotically optimal properties of our planner rely on a dense population of state space with samples and trajectories. Thus, a more efficient means of computing trajectories is required.

The simplest technique in use for fixed-wing vehicle motion planning is Dubins curves [38]. While powerful, and extremely fast to compute, Dubins curves are only continuous in heading and have discontinuities in curvature, which corresponds to discontinuities in roll angle. For vehicles flying high above obstacles with a conservative radius used in Dubins arcs, this is not a problem as the controller can stabilize the vehicle back onto the path [33]. However, if we wish to use the full dynamic envelope, a different technique is required.

Spline curves — piecewise, independent polynomials in each dimension — are a potentially appealing choice. Mellinger et al. use "minimum-snap" spline curves for their quad-rotor vehicles flying in and around obstacles [41]. However, this technique is not directly applicable to planning for a fixed-wing vehicle.

Most importantly, splines do not respect curvature constraints and do not have constant velocity norm along a path. Pythagorean hodographs overcome these issues by maintaining a "perfect square" property of polynomials along a path such that $c(t) = \sqrt{a(t)^2 + b(t)^2}$ where a, b, and c are all polynomials [19]. While these curves may be arbitrarily smooth, the transverse derivatives, which are what we care about for control effort to follow the paths, are not explicitly specified in the path formulation.

Another alternative is to use clothoid-arc segments which stitch together Dubins curve segments (arcs and lines) with segments of continuously varying curvature [55].

While better than Dubins curves, this would result in paths where the vehicle either has zero roll rate or rolls at some pre-specified maximum roll rate corresponding to the derivative of curvature.

Our approach leverages the advantages of splines presented by Mellinger et al., but extends this functionality by defining a transverse offset from a nominal path that captures the basic shape of the desired trajectory, allowing for explicit control (both specification and optimization of) the transverse derivatives.

2.3 State Estimation

State estimation using Kalman filtering techniques has been extensively studied for vehicles flying outdoors where GPS is available. A relevant example of such a state estimation scheme developed by Kingston et al. [32] involves two Kalman filters where roll and pitch is determined by a filter driven by gyro measurements as inputs and where the accelerometer measurements are treated as a measurement of the gravity vector, assuming unaccelerated flight. A separate filter estimates position and yaw using GPS measurements.

This approach is representative of many IMU based estimators that assume zero acceleration and thus use the accelerometer reading as a direct measurement of attitude (many commercially available IMUs implement similar techniques on-board using a complementary filter). While this approach has practical appeal and has been successfully used on a number of MAVs, the zero acceleration assumption does not hold for general flight maneuvering and thus the accuracy of the state estimate degrades quickly during aggressive flight.

Van der Merwe et al. use a sigma-point unscented Kalman filter (UKF) for state estimation on an autonomous helicopter [56]. The filter utilizes another typical approach whereby the accelerometer and gyro measurements are directly integrated to obtain position and orientation and are thus treated as noise perturbed inputs to the filter. Our filter utilizes this scheme in our process model, however we use an EKF with exponential coordinates based attitude representation instead of the quaternions used by Van der Merwe et al.

Techniques to identify the noise parameters relevant for the Kalman filter emerged not long after the original filter, however the most powerful analytical techniques assume steady state behavior of a linear time invariant system and are thus unsuitable for the time varying system that results from linearizing a nonlinear system [40]. More recent work optimizes the likelihood of a ground-truth projection of the state over the noise parameters but thus requires the system be fitted with a sensor capable of providing ground-truth for training. [2]. Our algorithm does not require the use of additional sensors, or external ground truth.

Laser range finders combined with particle filter based localization is widely used in ground robotic systems [53]. While planar LIDARs are commonly used to estimate motion in the 2D plane, they have also proved useful for localization in 3D environments. Prior work in our group [5], as well as others [51, 22] leveraged a 2D laser rangefinder to perform SLAM from a quadrotor in GPS-denied environments. The systems employ 2D scan-matching algorithms to estimate the position and heading, and redirect a few of the beams in a laser scan to estimate the height. While the systems have demonstrated very good performance in a number of realistic environments, they must make relatively strong assumptions about the motion of the vehicle, and the shape of the environment. Namely, they require walls that are least locally vertical, and a mostly flat floor for height estimation. As a result, the algorithms do not extend to the aggressive flight regime targeted in this paper. Scherer et al. use laser rangefinders to build occupancy maps, and avoid obstacles while flying fast through obstacles [49], however they rely on accurate GPS measurements for state estimation, and do not focus on state estimation.

In addition to the laser based systems for GPS-denied flight, there has been a significant amount of research on vision based control of air vehicles. This includes both fixed wing vehicles [31], as well as larger scale helicopters [12, 30, 27]. While vision based approaches warrant further study, the authors do not address the challenge of agile flight. This is likely to be particularly challenging for vision sensors due to the induced motion blur, combined with the computational complexity of vision algorithms.

Recently, Hesch et al. [26] developed a system that is similar in spirit to ours to localize a laser scanner and INS for localizing people walking around in indoor environments. They make a number of simplifying assumptions such as zero velocity updates, that are not possible for a fixed-wing vehicle. Furthermore, they model the environment as a set of planar structures, which limits the types of environments in which their approach is applicable. Our system uses a general occupancy grid representation which provides much greater flexibility of environments.

2.4 Motion Planning Under Uncertainty

The general motion planning problem of trying to find a collision-free path from some starting state to some goal region has been extensively studied. In particular, sampling-based techniques have received much attention over the last 15 years. The Rapidly-exploring Random Tree (RRT) operates by growing a tree in state space, iteratively sampling new states and then "steering" the existing node in the tree that is closest to each new sample towards that sample. The RRT has many useful properties including probabilistic completeness and exponential decay of the probability of failure with the number of samples [37].

The Rapidly-exploring Random Graph (RRG) proposed by Karaman and Frazzoli is an extension of the RRT algorithm [29]. In addition to the "nearest" connection, new samples are also connected to every node within some ball. The result is a connected graph that not only rapidly explores the state space, but also is locally refined with each added sample. This continuing refinement ensures that in the limit of infinite samples, the RRG contains all possible paths through the environment that can be generated by the steering function used to connect samples. The RRT* algorithm exploits this property to converge to the optimal path by only keeping the edges in the graph that result in lower cost at the vertices with in the ball. While these algorithms have many powerful properties, they assume fully deterministic dynamics and are thus unsuitable for stochastic problems in their proposed form.

Partial observability issues pose severe challenges from a planning and control perspective. Computing globally optimal policies for partially observable systems is possible only for very narrow classes of systems. In the case of discrete states, actions, and observations, exact Partially Observable Markov Decision Process (POMDP) algorithms exist, but are computationally intractable for realistic problems [28]. Many approximate techniques have been proposed to adapt the discrete POMDP framework to motion planning, but they still scale poorly with the number of states, and the prospect of discretizing high-dimensional continuous dynamics is not promising [48].

For systems with linear dynamics, quadratic cost, and Gaussian noise properties (LQG), the optimal policy is obtained in terms of a Kalman filter to maintain a Gaussian state estimate, and a linear control law that operates on the mean of the state estimate [8]. While global LQG assumptions are not justified for the problems we are interested in, many UAVs operate with locally linear control laws about nominal trajectories, and the Kalman filter with various linearization schemes has proven successful for autonomous systems that localize using on-board sensors [4].

The Belief Road Map (BRM) [47] explicitly addresses observability issues by simulating measurements along candidate paths and then choosing the path with minimal uncertainty at the goal. However, the BRM assumes the mean of the system is fully controllable at each time step, meaning that while the path is being executed, the controller is always capable of driving the state estimate back to the desired path. This assumption is valid only for a vehicle flying slowly and conservatively such that dynamic constraints can be ignored. Platt et al. [46] assume maximum likelihood observations to facilitate trajectory optimization techniques and then replan when the actual path deviates past a threshold.

The notion of chance-constrained motion planning is not new. A method of allocating risk for fully observable systems is discussed by Ono et al. [44]. Evaluating a performance metric over a predicted closed-loop distribution for partially observable systems was described by van den Berg et al. [6] and also derived independently by the authors [10], while He et al. [25] use a similar technique with open-loop action sequences. The algorithm proposed by van den Berg et al., termed LQG-MP, picks the best trajectory in terms of minimum cost with a bounded probability of collision from an RRT. However, Karaman and Frazzoli show that the RRT in essence enumerates a finite number of paths, even in the limit of infinite samples [29]. Thus there is no guarantee that a "good" path will be found either in terms of cost or uncertainty properties. In fact, we provide an example problem where an RRT fails to find a solution that satisfies chance-constraints, even though one exists. In contrast, by searching over an underlying graph our approach is both complete and optimal as the graph is refined in the limit.

Search repair after the underlying graph changes is discussed by Koenig et al. [34], however that algorithm is used for deterministic queries, and our search techniques make direct use of covariance propagation properties. Censi et al. [13] and Gonzalez and Stentz [21] use search with a similar pruning technique, however both algorithms operate on static graphs and do not consider dynamic constraints. We extend the pruning strategy for dynamic systems (i.e., a non deterministic mean) and also use the pruning strategy in the context of an incremental algorithm to terminate search.

Chapter 3

Trajectory Generation and Control

In this chapter we develop the transverse-polynomial trajectory generation technique. It takes as input a start and end configuration generated by sampling in our planning algorithm and returns a dynamically feasible path between the configurations while optimizing for the control effort (primarily in roll) necessary to follow that path. The algorithm "steers" the vehicle between two points in state space by optimizing the coefficients of polynomials offset from a nominal path.

3.1 Coordinate Frames and Nomenclature

In this thesis we will primarily concern ourselves with only two coordinate frames. A static global frame, denoted by subscript, g, is a static east-north-pp (xyz) coordinate frame anchored at some origin. A body frame has its origin at the center of mass of the vehicle and is oriented x forward, y left, and z up (FLU). The transformation between the two frames is given by a vector, Δ, between the origins expressed in the global frame and an orientation given by R. The xyz components of any 3-vector quantity will be subscripted numerically as 1 for x, 2 for y, and 3 for z. This is depicted in figure 3-1.[1]

[1]The use of the ENU-FLU coordinate frame runs counter to conventional aeronautical analysis, however, it is in-line with more recent work done with quad-rotors operating indoors (which likely inherit z-up coordinate frames from ground robots) and the choice was made to make easier use of existing software libraries for mapping and visualization.

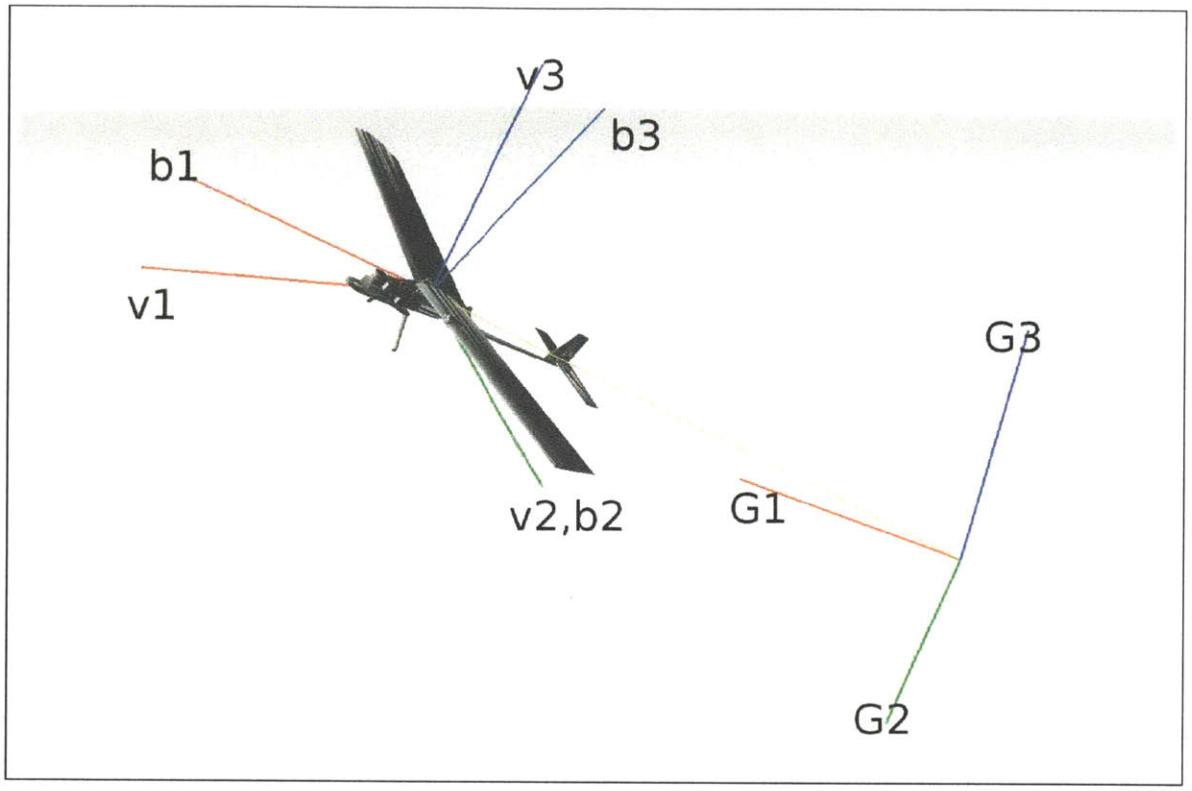


Figure 3-1: The velocity, body, and global frames depicted with XYZ axes in red, green, and blue respectively. The global frame ENU global frame is fixed to the ground with x pointing east). The body frame is fixed to the vehicle with x pointing out the nose. The velocity frame, assuming coordinated flight, shares the y-axis with the body frame and has the x-axis aligned with velocity. With positive angle of attack as shown in this picture, the x-axis points down relative to the body.

We will primarily represent acceleration, angular velocity, and linear velocity in body coordinates. However, in this chapter exceptions apply. For our coordinated flight model it is sometimes convenient to discuss velocity and acceleration in the global frame. In these cases we will refer to derivatives of position for clarity (i.e., $\dot{\Delta}$). Additionally, in this chapter we will use a velocity frame which shares the origin with the body frame, but has its x axis aligned with the body velocity, while the y and z axes still point left and up respectively as depicted in figure 3-1.

3.2 Coordinated Flight Model

The planning and control algorithms for the fixed wing vehicle are based on a coordinated flight model [23]. Coordinated flight is defined as a flight condition where the body velocity of the vehicle is contained within the longitudinal plane, $v_{b2} = v_{v2} = 0$ and hence $\dot{v}_{by} = 0$. The equations of motion for the coordinated flight model are expressed in the velocity frame which differs from the body frame by the angle of attack of the vehicle (assuming the vehicle is in a state of coordinated flight). The orientation of the velocity frame is given by the rotation matrix R_v in Forward-Left-Up (FLU) to East-North-Up (ENU) coordinates. The unit vector of the first column aligns with the velocity, $v_{v2} = v_{v3} = 0$, the unit vector of the second column points out the left wing, and the unit vector of the third column points up. We also have $\dot{\Delta} = R_v v_v$ and $\ddot{\Delta} = g + R_v a_v$. To maintain coordinated flight, the y and z components of velocity are constrained as:

$$\omega_{v2} = -(a_{v3} + g_{v3})/V \tag{3.1}$$

$$\omega_{v3} = g_{v2}/V, \tag{3.2}$$

where $g_v = R_v g$ and $V = ||v_v|| = v_{v1} = ||\dot{\Delta}||$.

A system is differentially flat if the inputs and states can be written as functions of the outputs and their derivatives. The coordinated flight model is flat with inputs $(\omega_{v1}, \dot{a}_{v1}, \dot{a}_{v3})$, states $(\Delta, \dot{\Delta}, R_v, a_{v1}, a_{v3})$, and output Δ, with the flat relationship expressed:

$$\begin{bmatrix} \dot{a}_{v1} \\ \omega_{v1} \\ \dot{a}_{v3} \end{bmatrix} = \begin{bmatrix} -\omega_{v2} a_{v3} \\ \omega_{v3} a_{v1}/a_{v3} \\ \omega_{v2} a_{v1} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1/a_{v3} & 0 \\ 0 & 0 & 1 \end{bmatrix} R^T \dddot{\Delta}. \tag{3.3}$$

Equation 3.3 is remarkably powerful. It tells us the necessary inputs for the simplified coordinated flight model to achieve an arbitrary third derivative on the trajectory of the airplane. To get intuition about how the differentially flat model, consider straight and level steady state flight with the world and velocity frames aligned for which we have,

$$\begin{bmatrix} \dot{a}_{v1} \\ \omega_1 \\ \dot{a}_{v3} \end{bmatrix} = \begin{bmatrix} \dddot{\Delta}_1 \\ -\dddot{\Delta}_2/g \\ \dddot{\Delta}_3 \end{bmatrix}, \tag{3.4}$$

yielding that the roll rate is proportional to the lateral (y direction) "jerk" of the path, the derivative of tangential acceleration is the forward "jerk" of the path, and the derivative of normal acceleration is the upward "jerk" of the path. Thus for a trajectory to have a continuous roll rate it must be smooth up to the 3rd derivative.

3.3 Trajectory Generation

Excepting takeoff and landing maneuvers, we restrict the motion of the vehicle to be in the plane at a constant altitude and fixed speed. For planning purposes we wish to be able to generate trajectories from an initial position, (Δ_1, Δ_2), yaw angle, ψ, roll angle, ϕ, and roll rate $\dot{\phi}$ to a final configuration.

Trajectory generation in the plane must primarily be concerned with roll dynamics as the pitch is effectively constrained to stay in the plane, rudder is constrained to maintain coordinated flight, and throttle is constrained to maintain constant speed. For the case of constant speed planar motion,

$$\phi = \tan^{-1}\left(\frac{V^2\kappa}{g}\right), \tag{3.5}$$

is a special case of equation 3.3, where κ is the curvature of the path (inverse radius, positive turning to the left or counter clockwise (CCW)).

The most commonly used approach for planar path planning for fixed wing vehicles is to use Dubins curves [38]. Dubins curves represent the optimal (shortest distance) path between two position and orientation, $(\Delta_1, \Delta_2, \psi)$, configurations respecting a minimum turning radius. The path between any two configurations will be made up three path segments consisting of either arcs (of the minimum turning radius) or straight lines, belonging to six possible "words", LSL, RSR, RSL, LSR, RLR, LRL, where L is a left turning arc, R is a right turning arc, and S is a straight line. The parameters of the segments (center, entry and exit angles for an arc, and start and end point for a line) can be directly determined from the geometry of the start and end configurations for each feasible word for a given configuration and then the shortest word selected. It is also possible to analytically determine which word will be shortest based on an algebraic partitioning of SE2, but for implementation simplicity we use the former method.

Since Dubins curves are composed of tangent lines and arcs, the curves are smooth in the sense that a vehicle following the path will have continuous yaw angle, but the curvature is discontinuous. We can see from equation 3.5 that discontinuity in curvature will translate to discontinuity in roll angle. This is clearly kinodynamically infeasible for a fixed wing vehicle. However, if the radius is chosen conservatively relative to the roll dynamics of the vehicle and the vehicle is not in close proximity to obstacles, Dubins curves provide a good approximation to kinodynamically feasible paths and a stabilizing controller can achieve reasonable tracking performance. Unfortunately, these conditions do not apply for the system we are interested in as we seek to make use of the full dynamic range of the vehicle while flying close to obstacles.

Differentiating equation 3.5, the roll rate is given by:

$$\dot{\phi} = \frac{V^3\dot{\kappa}}{g + \frac{(V^2\kappa)^2}{g}}. \tag{3.6}$$

The roll angle is a 2nd order system on aileron input [45]. For a truly kinodynamically feasible path we must have continuity up to the derivative of curvature. Clothoid arc paths are an extension of Dubins paths with clothoid arc segments stitching the lines and arc segments together to maintain continuous curvature (roll angle) [38]. However, the paths would still be discontinuous in roll rate. Further, this would require the selection of a roll rate (rate of change of curvature) at which all maneuvering would take place. As with the turning radius, this could be chosen conservatively at the expense of utilizing the full dynamic envelope. In addition, this kind of bang-bang control runs counter to how systems naturally behave. Minimum-snap trajectories, that is trajectories optimized to minimize quadratic cost on the 4th derivative and higher, have been shown to be successful for aggressive flight with quadrotors in highly constrained environments [41].

The roll angle and derivatives can be loosely approximated as,

$$\phi \approx \frac{V^2 \kappa}{g} \tag{3.7}$$

$$\dot{\phi} \approx \frac{V^3 \dot{\kappa}}{g} \tag{3.8}$$

$$\ddot{\phi} \approx \frac{V^4 \ddot{\kappa}}{g}. \tag{3.9}$$

While this is clearly an approximation as in general roll angles can be greater than 45^o, it gives the necessary intuition that minimizing the first and second derivatives of curvature for a path will translate to minimizing the roll rate and roll acceleration.

A quadrotor is capable of "rolling" about any axis in the body's horizontal plane and thus minimizing the snap or 4th derivative of independent splines in three dimensions (with a 4th spline for heading) will minimize the control effort to follow the path. This is exactly the method used by Mellinger et al. [41]. For a fixed wing vehicle, however, the picture is substantially more complicated. The axis along which roll acts rotates with the heading of the vehicle and the vehicle must turn with a finite radius, all while traveling at a constant or close to constant speed. This makes the independent axes spline method inapplicable.

The key idea of our approach is in parameterizing a lateral offset from a nominal path, and using this offset to optimize (minimize) the lateral dynamics quantities we care about.

3.3.1 Quadratic Optimization on the Derivatives of Polynomials

Let p_n denote the coefficients of a polynomial P of degree N such that

$$P(t) = p_n t^N + p_{n-1} t^{n-1} \ldots + p_0 \tag{3.10}$$

$$= \sum_{n=0}^{N} p_n t^n. \tag{3.11}$$

We are interested in optimizing the coefficients of P to minimize cost functions of the form

$$J = \int_0^T c_0 P(t)^2 + c_1 P'(t)^2 + c_2 P''(t)^2 + \ldots + c_N P^{(N)}(t)^2 dt, \tag{3.12}$$

which can be written in quadratic form as,

$$J = \bar{p}^T Q \bar{p} \tag{3.13}$$

where $\bar{p} \in \Re^{N+1}$ is the vector of polynomial coefficients and $Q = \sum_{r=0}^{N} c_r Q_r$.

Cost Matrix and Constraints

The square of a polynomial, P^2, can be written using a convolution sum

$$(P^2)_n = \sum_{j=0}^{N} p_j p_{n-j}. \tag{3.14}$$

The rth derivative is given by

$$P^{(r)}(t) = \sum_{n=r}^{N} \left(\prod_{m=0}^{r-1} (n-m) \right) p_n t^{n-r}. \tag{3.15}$$

Using equations 3.14 and 3.15 we can write the rth component of the cost as a quadratic function of the polynomial coefficients:

$$
\begin{aligned}
J_r &= \int_0^T P^{(r)}(t)^2 dt && (3.16)\\
&= \int_0^T \left(\sum_{n=r}^{N} \left(\prod_{m=0}^{r-1} (n-m) \right) p_n t^{n-r} \right)^2 dt && (3.17)\\
&= \int_0^T \sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m) \right) p_j t^{j-r} \left(\prod_{m=0}^{r-1} (n-j-m) \right) p_{n-j} t^{n-j-r} dt && (3.18)\\
&= \int_0^T \sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m)(n-j-m) \right) p_j p_{n-j} t^{n-2r} dt && (3.19)\\
&= \left(\sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m)(n-j-m) \right) p_j p_{n-j} \frac{t^{n-2r+1}}{(n-2r+1)} \right|_0^T && (3.20)\\
&= \sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m)(n-j-m) \right) p_j p_{n-j} \frac{T^{n-2r+1}}{(n-2r+1)} && (3.21)
\end{aligned}
$$

We find Q_r by taking the Hessian of this expression:

$$\frac{\partial J_r}{\partial p_i} = \frac{\partial}{\partial p_i} \sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m)(n-j-m) \right) p_j p_{n-j} \frac{\tau^{n-2r+1}}{(n-2r+1)} \tag{3.22}$$

$$= \sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m)(n-j-m) \right) \frac{\partial p_j p_{n-j}}{\partial p_i} \frac{\tau^{n-2r+1}}{(n-2r+1)} \tag{3.23}$$

$$= \sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m)(n-j-m) \right) \left(\frac{\partial p_j}{\partial p_i} p_{n-j} + \frac{\partial p_{n-j}}{\partial p_i} p_j \right) \frac{\tau^{n-2r+1}}{(n-2r+1)} \tag{3.24}$$

$$= \sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m)(n-j-m) \right) (\delta(i-j) p_{n-j} + \delta(n-j-i) p_j) \frac{\tau^{n-2r+1}}{(n-2r+1)} \tag{3.25}$$

$$= 2 \sum_{n=0}^{2N} \left(\prod_{m=0}^{r-1} (i-m)(n-i-m) \right) p_{n-i} \frac{\tau^{n-2r+1}}{(n-2r+1)} \tag{3.26}$$

$$\frac{\partial^2 J_r}{\partial p_i \partial p_l} = 2 \sum_{n=0}^{2N} \left(\prod_{m=0}^{r-1} (i-m)(n-i-m) \right) \frac{\partial p_{n-i}}{\partial p_l} \frac{\tau^{n-2r+1}}{(n-2r+1)} \tag{3.27}$$

$$= 2 \sum_{n=0}^{2N} \left(\prod_{m=0}^{r-1} (i-m)(n-i-m) \right) \delta(n-i-l) \frac{\tau^{n-2r+1}}{(n-2r+1)} \tag{3.28}$$

$$= 2 \left(\prod_{m=0}^{r-1} (i-m)(l-m) \right) \frac{\tau^{i+l-2r+1}}{(i+l-2r+1)} \tag{3.29}$$

$$Q_r^{il} = \begin{cases} 2 \left(\prod_{m=0}^{r-1} (i-m)(l-m) \right) \frac{\tau^{i+l-2r+1}}{(i+l-2r+1)} : & i \geq r \wedge l \geq r \\ 0 : & i < r \vee l < r \end{cases} \tag{3.30}$$

Similarly, constraints on the value of P and its derivatives can be written as linear functions of the coefficients:

$$A\bar{p} - b = 0. \tag{3.31}$$

The value of the rth derivative at τ is given by:

$$P^{(r)}(\tau) = \sum_{n=r}^{N} \left(\prod_{m=0}^{r-1} (n-m) \right) p_n \tau^{n-r}. \tag{3.32}$$

For a polynomial segment spanning from 0 to τ satisfying derivative constraints on both ends of the interval, A and b are constructed as:

$$A = \begin{bmatrix} A_0 \\ A_\tau \end{bmatrix}, b = \begin{bmatrix} b_0 \\ b_\tau \end{bmatrix} \tag{3.33}$$

$$A_{0_{rn}} = \begin{cases} \prod_{m=0}^{r-1}(r-m): & r = n \\ 0: & r \neq n \end{cases} \tag{3.34}$$

$$b_{0_r} = P^{(r)}(0) \tag{3.35}$$

$$A_{\tau_{rn}} = \begin{cases} \left(\prod_{m=0}^{r-1}(n-m)\right)\tau^{n-r}: & n \geq r \\ 0: & n < r \end{cases} \tag{3.36}$$

$$b_{\tau_r} = P^{(r)}(\tau). \tag{3.37}$$

Quadratic Program Formulation and Solution

We now have a quadratic program (QP) of the form:

$$\begin{aligned} \min_x \quad & x^T Q x \\ \text{s.t.} \quad & Ax - b = 0 \end{aligned}$$

where the decision variables are the polynomial coefficients, $x = \bar{p}$. Since the constraints are linear we can solve for x directly using the elimination approach [7]. The decision variables are partitioned (and rearranged if necessary) such that the first m (number of constraints) columns of A are linearly independent:

$$A = \begin{bmatrix} B & R \end{bmatrix} \tag{3.38}$$

$$x = \begin{bmatrix} x_B \\ x_R \end{bmatrix}. \tag{3.39}$$

This allows us to write x_B in terms of x_R using invertible B:

$$x_B = B^{-1}(b - Rx_R). \tag{3.40}$$

Partitioning Q and plugging in:

$$J = \begin{bmatrix} x_B^T & x_R^T \end{bmatrix} \begin{bmatrix} Q_{BB} & Q_{BR} \\ Q_{RB} & Q_{RR} \end{bmatrix} \begin{bmatrix} x_B \\ x_R \end{bmatrix} \tag{3.41}$$

$$= x_B^T Q_{BB} x_B + x_B^T Q_{BR} x_R + x_R^T Q_{RB} x_B + x_R^T Q_{RR} x_R \tag{3.42}$$

$$\begin{aligned} &= (b - Rx_R)^T B^{-T} Q_{BB} B^{-1}(b - Rx_R) + 2(b - Rx_R)^T B^{-T} Q_{BR} x_R \\ &\quad + x_R^T Q_{RR} x_R \end{aligned} \tag{3.43}$$

$$\begin{aligned} &= b^T B^{-T} Q_{BB} B^{-1} b - 2b^T B^{-T} Q_{BB} B^{-1} R x_R \\ &\quad + x_R^T R^T B^{-T} Q_{BB} B^{-1} R x_R + 2b^T B^{-T} Q_{BR} x_R \\ &\quad - 2x_R^T R^T B^{-T} Q_{BR} x_R + x_R^T Q_{RR} x_R \end{aligned} \tag{3.44}$$

$$\begin{aligned} &= b^T B^{-T} Q_{BB} B^{-1} b \\ &\quad + 2b^T B^{-T}(Q_{BR} - Q_{BB} B^{-1} R) x_R \\ &\quad + x_R^T (Q_{RR} + R^T B^{-T} Q_{BB} B^{-1} R - 2R^T B^{-T} Q_{BR}) x_R \end{aligned} \tag{3.45}$$

The $b^T B^{-T} Q_{BB} B^{-1} b = J_c$ term is the "penalty" in cost for satisfying the constraints and it does not factor into the location of the optimal solution x^*. Substituting $Q'_{RR} = Q_{RR} + R^T B^{-T} Q_{BB} B^{-1} R - 2R^T B^{-T} Q_{BR}$ and $f'_R = 2b^T B^{-T}(Q_{BR} - Q_{BB} B^{-1} R)$ we can rewrite as:

$$J = J_c + f_R'^T x_R + x_R^T Q'_{RR} x_R \tag{3.46}$$

$$\frac{\partial J}{\partial x_R} = f'_R + 2Q'_{RR} x_R \tag{3.47}$$

$$x_R^* = -\frac{1}{2} Q'^{-1}_{RR} f'_R \tag{3.48}$$

$$x_B^* = B^{-1}(b - Rx_R^*). \tag{3.49}$$

With $\bar{p}$ indexed as in equation 3.11, the first m columns of A will correspond to the low-order polynomial coefficients. The construction of A (equations 3.34 and 3.36) assures the linear independence of these columns. We restrict ourselves to considering problems where derivative constraints are only present sequentially starting with the value of the polynomial (i.e., we don't specify the $r+1$th derivative if the rth derivative is not specified). We can see from equation 3.48 that Q_{RR} must also be invertible (in fact positive-definite) for a unique solution to exist. By equation 3.30 this requires that their exists at least one non zero c_i for $0 \leq i \leq m$ meaning that we must have non zero cost on at least one derivative of lower or equal order to m to assure the convexity of the problem. We also note that m will be equal to the number of derivatives (including the 0th) specified at the beginning of the interval plus the number specified at the end.

We now have all the tools necessary to optimize for single polynomials according to quadratic cost on the derivatives and satisfying boundary constraints. Figure 3-2 shows the behavior of the optimization for different parameters.

3.3.2 Piecewise Polynomial Joint Optimization

Since we wish to "correct" for derivative discontinuities in Dubins curves it is necessary to jointly optimize multiple polynomial segments with specified derivative offsets between them. A piecewise polynomial path can be constructed as:

$$T(t) = \begin{cases} P_0(t) : & t \leq \tau_0 \\ {}_1P_1(t-\tau_0) : & \tau_0 < t \leq \tau_1 + \tau_0 \\ P_2(t-(\tau_0+\tau_1)) : & \tau_0 + \tau_1 t \leq \tau_2 + \tau_1 + \tau_0 \\ \vdots \end{cases} \tag{3.50}$$

where we define $T_k = \sum_{\kappa=0}^{k} \tau_\kappa$ to write

$$T(t) = P_k(t - \Gamma_{k-1}) : \qquad \Gamma_{k-1} < t \leq \Gamma_k. \tag{3.51}$$

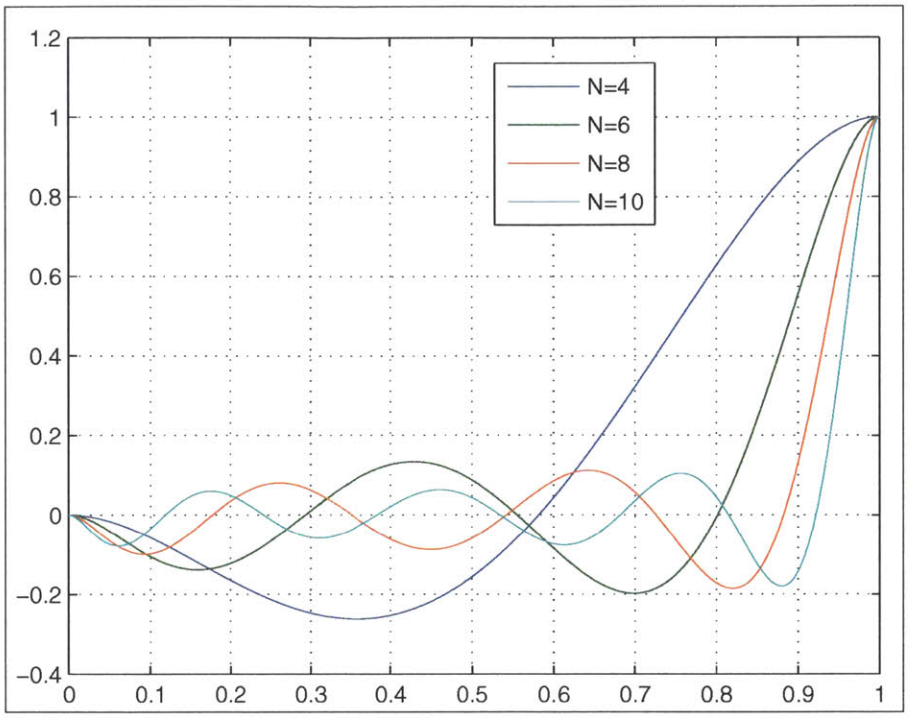


Figure 3-2: This figure shows polynomials with quadratic cost on the value of the polynomial for 0 constraints on the 0th and first derivatives at t=0 and 1 and 0 constraints at t=1 respectively. We can see that increasing the order of the polynomial better satisfies the cost function but also increases "ringing" in the solution.

We wish to enforce derivative continuity up to degree D. We allow the underlying path to have intrinsic transverse derivative discontinuities $\sigma_k^{(r)} = S_{k+1}^{(r)} - S_k^{(r)}$.

We can formulate an optimization over the polynomial coefficients by forming a vector

$$x = \left[\begin{array}{cccc} \bar{p}_0 & \bar{p}_1 & \dots & \bar{p}_K \end{array} \right] \tag{3.52}$$

The cost matrix is constructed as block diagonal on Q_k. The constraint matrix has the form:

$$A = \begin{bmatrix} A_0^0 & 0 & 0 & 0 & \dots & 0 & 0 \\ -A_\tau^0 & A_0^1 & 0 & 0 & \dots & 0 & 0 \\ 0 & -A_\tau^1 & A_0^2 & 0 & \dots & 0 & 0 \\ 0 & 0 & -A_\tau^2 & A_0^3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & -A_\tau^{K-1} & A_3^K \\ 0 & 0 & 0 & 0 & \dots & 0 & A_\tau^K \end{bmatrix}, \tag{3.53}$$

$$b = \begin{bmatrix} T^d(0) \\ \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{K-2} \\ T^d(\Gamma_K) \end{bmatrix}, \tag{3.54}$$

where $T^d(0)$ is a specified vector of derivative constraints at the beginning of the piecewise polynomial, and $T^d(\Gamma_K)$ is a specified vector of derivative constraints at the end. We have $|T^d(0)| = D_0, |T^d(\Gamma_K)| = D_{\Gamma_K}$, and in general $D_0 \neq D_{\Gamma_K} \neq D$. The total number of constraints $m = D_0 + D_{\Gamma_K} + (K-1)D$. To form the partition for elimination solving B must be full rank. To accomplish this we reindex and form B as the concatenation of the first $\lfloor m/K \rfloor$ columns of each block of columns corresponding to section k. The remaining $m - K\lfloor m/K \rfloor$ columns are taken as the next higher order column from arbitrary column blocks. The joint optimization can then be performed as above using the elimination approach.

For the purpose if achieving derivative continuity between Dubins curve segments all the elements of σ are set to 0 except for the second derivative which is set to the curvature difference between the preceding and following segments. $T^d(\Gamma_K)$ and $T^d(0)$ may be set arbitrarily, keeping in mind that the actual 2nd derivative will include the intrinsic curvature of the path segment, but practically the Dubins curve is constructed to a point (0th derivative) and heading (1st derivative) so there should only be a need to specify 2nd and higher order derivatives in the polynomial optimization (e.g. to match roll angle and rate boundary conditions). After the joint optimization is performed, the path is formed by offsetting (positive to the left) the polynomial from the original Dubins path, however, as we will see in the next section, a second step is necessary to obtain a smooth path.

3.3.3 Polar Coordinates Corrections

The polynomial offset from an arc in a Dubins curve is effectively a polynomial in polar coordinates, $R(\theta) = R_0 \pm P(\theta)$ (with the plus or minus depending on if the arc is CCW (-) or CW (+). To analyze the true polar coordinate derivatives we use the arc path ρ in the $\hat{r}$ direction (radial to the arc center) parameterized by θ which sweeps positive in the $\hat{\theta}$ direction. Note that the $\hat{\theta}$ may be CCW or CW positive depending on the intrinsic arc direction in contrast to conventional CCW only use. When used in a standard coordinate system $\hat{r}(\theta) = \cos\theta\hat{x} + \sin\theta\hat{y}$ and $\hat{\theta}(\theta) = -\sin\theta\hat{x} + \cos\theta\hat{y}$ for a CCW arc and $\hat{\theta}(\theta) = \sin\theta\hat{x} - \cos\theta\hat{y}$ for a CW arc.

Using the identities $\hat{r}'(\theta) = \hat{\theta}(\theta)$ and $\hat{\theta}'(\theta) = -\hat{r}(\theta)$ we obtain

$$\rho(\theta) = R(\theta)\hat{r}(\theta) \tag{3.55}$$

$$\rho'(\theta) = P'(\theta)\hat{r}(\theta) + R(\theta)\hat{\theta}(\theta) \tag{3.56}$$

$$\rho''(\theta) = (P''(\theta) - R(\theta))\hat{r}(\theta) + 2P'(\theta)\hat{\theta}(\theta) \tag{3.57}$$

$$\rho'''(\theta) = (P'''(\theta) - 3P'(\theta))\hat{r}(\theta) + (3P''(\theta) - R(\theta))\hat{\theta}(\theta), \tag{3.58}$$

for the first three derivatives.

To generate smooth paths we would like to parameterize in terms of the metric distance along the path as opposed to the angle swept out by the arc. To accomplish this we write θ as a function of the distance s to get the arc as $\rho(\theta(s))$ and take the derivatives with respect to the distance.

$$\frac{d\rho(\theta(s))}{ds} = \rho(\theta(s))'\theta'(s) \tag{3.59}$$
$$\frac{d^2\rho(\theta(s))}{ds^2} = \rho''(\theta(s))\theta'(s)^2 + \rho'(\theta(s))\theta''(s) \tag{3.60}$$
$$\frac{d^3\rho(\theta(s))}{ds^3} = \rho'''(\theta(s))\theta'(s)^3 + 3\rho''(\theta(s))\theta''(s)\theta'(s) + \rho'(\theta(s))\theta'''(s). \tag{3.61}$$

To enforce that s is the distance along the path we use the constraint:

$$||\rho'(\theta(s))|| = 1 = \theta'(s)\sqrt{P'(\theta(s))^2 + (R(\theta(s)))^2}, \tag{3.62}$$

which yields the relationship:

$$\theta'(s) = \frac{1}{\sqrt{P'(\theta(s))^2 + (R(\theta(s)))^2}}, \tag{3.63}$$

which can then be differentiated for use in the full path derivative expressions:

$$\theta' = \frac{1}{L} \tag{3.64}$$
$$\theta'' = -\frac{P'(P''+R)}{L^4} \tag{3.65}$$
$$\theta''' = -\frac{L^2(P''(P''+R) + P'(P'''+P')) - 4P'^2(P''+R)^2}{L^7}. \tag{3.66}$$

For convenience we define $L = \sqrt{P'^2 + R^2}$ and omit the argument θ for compactness in the expressions. Equations 3.56 - 3.58 and 3.64-3.66 can then be plugged into 3.59-3.61 to obtain expressions for the true path derivatives in the $\hat{r}$ direction:

$$\hat{r}^T\frac{d\rho(\theta(s))}{ds} = \frac{P'}{\sqrt{P'^2+R^2}} \tag{3.67}$$
$$\hat{r}^T\frac{d^2\rho(\theta)}{ds^2} = P''\left(\frac{1}{L^2} - \frac{P'^2}{L^4}\right) - \frac{R}{L^2} - \frac{P'^2R}{L^4} \tag{3.68}$$

$$\begin{aligned}\hat{r}^T\frac{d^3\rho(\theta(s))}{ds^3} = {} & \frac{P'''-3P'}{L^3} - 3\frac{P'(P''-R)^2}{L^5} \\ & - P'\left(\frac{P''^2 + P''R + P'P''' + P'^2}{L^5} - \frac{4P'^2(P''+R)^2}{L^7}\right),\end{aligned} \tag{3.69}$$

and similar expressions can be obtained in the $\hat{\theta}$ direction (or the numerical values substituted directly). To complete the optimization we also need the expressions solved for the first three polynomial derivatives:

$$P' = \frac{R\hat{r}^T \frac{d\rho(\theta(s))}{ds}}{\sqrt{1 - \left(\hat{r}^T \frac{d\rho(\theta(s))}{ds}\right)^2}} \tag{3.70}$$

$$P'' = \frac{\hat{r}^T \frac{d^2\rho(\theta)}{ds^2} + \frac{R}{L^2} + \frac{P'^2 R}{L^4}}{\frac{1}{L^2} - \frac{P'^2}{L^4}} \tag{3.71}$$

$$\begin{aligned} P''' = {} & L^3 \hat{r}^T \frac{d^3\rho(\theta(s))}{ds^3} + 3P' + 3\frac{P'(P''-R)^2}{L^2} \\ & + P'\left(\frac{P''^2 + P''R + P'P''' + P'^2}{L^2} - \frac{4P'^2(P''+R)^2}{L^4}\right). \end{aligned} \tag{3.72}$$

3.3.4 Line Segment Corrections

For the line segments in a Dubins path similar analysis applies. Using the same ρ notation with $\hat{r}$ perpendicular to the left of the line segment and $\hat{\theta}$ aligned with the line segment, and θ the linear distance along the segment, we have:

$$\rho(\theta) = P(\theta)\hat{r} + \theta\hat{\theta} \tag{3.73}$$

$$\rho'(\theta) = P'(\theta)\hat{r} + \hat{\theta} \tag{3.74}$$

$$\rho''(\theta) = P''(\theta)\hat{r} \tag{3.75}$$

$$\rho'''(\theta) = P'''(\theta)\hat{r}. \tag{3.76}$$

Following the same procedure as for arc segments we obtain:

$$\hat{r}^T \frac{d\rho(\theta(s))}{ds} = \frac{P'}{\sqrt{P'^2 + 1}} \tag{3.77}$$

$$\hat{r}^T \frac{d^2\rho(\theta)}{ds^2} = \frac{P''}{L^2} - \frac{P'^2 P''}{L^4} \tag{3.78}$$

$$\hat{r}^T \frac{d^3\rho(\theta(s))}{ds^3} = \frac{P'''}{L^3} - 3\frac{P'P''^2}{L^5} - P'\frac{L^2(P''^2 + P'P''') - 4P'^2 P''^2}{L^7}, \tag{3.79}$$

and solved for the polynomial derivatives:

$$P' = \frac{\hat{r}^T \frac{d\rho(\theta(s))}{ds}}{\sqrt{1 - \left(\hat{r}^T \frac{d\rho(\theta(s))}{ds}\right)^2}} \tag{3.80}$$

$$P'' = L^2 \hat{r}^T \frac{d^2\rho(\theta)}{ds^2} + \frac{P'^2 P''}{L^2} \tag{3.81}$$

$$P''' = L^3 \hat{r}^T \frac{d^3\rho(\theta(s))}{ds^3} + 3\frac{P'P''^2}{L^2} + P'\frac{L^2(P''^2 + P'P''') + 4P'^2P''^2}{L^4}. \tag{3.82}$$

3.3.5 Dubins-Polynomial Paths

The procedure for computing Dubins-Polynomial paths is depicted in algorithm 1. It is important to note that the polynomials used in equations 3.70-3.70 and 3.67-3.69 are in terms of the arc angle θ and thus the polynomials obtained must be rescaled by the arc radius as appropriate to be consistent in cost units between arc and line segments. Example curves for various configurations and cost settings are shown in figure 3-3. Figure 3-4 shows numerically integrated derivatives compared to the actual path confirming that the method returns paths with continuous 3rd derivatives and that our expressions for the derivatives are correct.

Algorithm 1 Dubins-Polynomial Path Generation

```
1: Input: $(x, y, \theta, \kappa, \dot\kappa, \ddot\kappa)_0, (x, y, \theta, \kappa, \dot\kappa, \ddot\kappa)_1$
2: Find Dubins curve for $(x, y, \theta)_0, (x, y, \theta)_1$
3: Perform piecewise polynomial optimization over Dubins curve with appropriate
   boundary conditions for $(\kappa, \dot\kappa, \ddot\kappa)_0, (\kappa, \dot\kappa, \ddot\kappa)_1$
4: Compute "true" transverse ($\hat{r}$) derivatives at each junction by averaging the pre-
   ceeding and following segments transverse derivatives as computed using equa-
   tions 3.67-3.69 (or 3.77-3.79 for a line)
5: Invert the transverse derivatives using equations 3.70-3.70 (3.80-3.80) to obtain
   polynomial boundary conditions for each segment
6: Optimize each segment individually single polynomial optimization
```

3.3.6 Path Limitations

To the extent that P' is small in the case of line segments and P' and P are small in the case of arc segments (since change in P changes the radius and thus introduces

Coriolis and other higher order polar effects), the polynomial optimization directly optimizes the curvature derivative (3rd polynomial derivative), and second curvature derivative (4th polynomial derivative). However, we can see from the correction equations in sections 3.3.3 and 3.3.4 that the approximation breaks down for P' and P far from 0. If the nominal path from which the offset is specified is chosen sensibly, this problem can be mitigated. The optimization also allows for cost on P' and P to penalize too much deviation from the nominal path and thus maintain the fidelity of the higher order derivatives. Further, even if the optimization is an approximation, the correction step ensures derivative continuity at segment junctions.

If the cost on curvature derivatives is high and the radius used in the Dubins paths too small, the optimization may occasionally yield arc segments with negative radius. When this occurs, the path must either be discarded or the cost on P increased until a positive radius is achieved. Potential locations of radius violations may be obtained using polynomial root finding algorithms.

3.4 Control

We now have the ability to produce paths with continuous roll rate optimized to minimize roll rate and roll acceleration. The 3rd derivative of the paths is immediately available through the same expressions used in the optimization. Thus, the open loop roll rate and change in normal acceleration necessary to follow the paths can be easily determined from equation 3.3. However, two key challenges remain. The first is in stabilizing the vehicle along a planned path and the second is in converting the differentially flat inputs, namely roll and change in normal acceleration, to the inputs available on the vehicle, aileron, elevator and rudder. The reader should note that the following section on trajectory stabilization does not contain novel research. We restate results from Hauser et al. [23] for clarity and to provide a thorough understanding of how the control system works.

3.4.1 Trajectory Stabilization

Differential flatness provides a direct method for stabilizing a desired path. Since the model allows us to command the third spatial derivative of the vehicle we can simply write the desired third derivative as the open loop derivative of the path plus feedback on deviation from the path and deviation derivatives (equation (8), [23]):

$$\Delta^{(3)} = \rho^{(3)} + k_2\ddot{e} + k_1\dot{e} + k_0 e, \tag{3.83}$$

where ρ is the path to be followed — a Dubins-Polynomial path in our case — and $e = \rho - \Delta$. The disadvantage of this approach is that it will use the tangential acceleration to correct axial errors along the path (i.e. the vehicle gets ahead or behind). Since we know the trajectory is collision-free, errors along the trajectory are not a concern, and changing speed to correct errors in the worst case could take the airplane out of an acceptable maneuvering envelope.

Following equation 11, [23] we have:

$$\begin{aligned} M\begin{bmatrix} s^{(3)} \\ \omega_{v1} \\ \dot{a}_{v3} \end{bmatrix} &= \begin{bmatrix} a_{v3}\omega_{v2} + \dot{a}_{v1} \\ a_{v1}\omega_{v3} \\ -a_{v1}\omega_{v2} \end{bmatrix} \\ &- R^T\left(3\rho''(s)\ddot{s}\dot{s} + \rho'''(s)\dot{s}^3 + k_2\ddot{e} + k_1\dot{e} + k_0 e\right), \end{aligned} \tag{3.84}$$

where,

$$M = \begin{bmatrix} \vdots & 0 & 0 \\ R^T\rho'(s) & a_{v3} & 0 \\ \vdots & 0 & -1 \end{bmatrix} \tag{3.85}$$

The key idea is replacing the change in tangential acceleration with the third derivative of the path parameter, s, while $\dot{s}$ and $\ddot{s}$, now become states in the controller. In our formulation s is the metric distance along the path so $\dot{s}$ can be initialized as the flight speed entering the path. The only restriction on this control law is that M be invertible which requires the fist column of $R_v \propto \dot{\Delta}$ can't be orthogonal to ρ'. Intuitively this makes sense since if the plane is flying perpendicular to the desired path, no control authority on the error dynamics is available through $\ddot{s}$. The singularity in M is also not of practical concern so long as the controller is initialized from reasonable conditions.

3.4.2 Differentially Flat Control Input Conversion

The trajectory stabilization algorithm provides us with ω_{v1} and $\dot{a}_{vt}$ and the remaining challenge is to translate these quantities into aileron, elevator, throttle, and rudder inputs such that coordinated flight is achieved at the desired settings.

Using traditional linear analysis for aircraft dynamics, roll rate can be shown to be a stable first order system with aileron input, and normal acceleration (effectively angle of attack or normal velocity) a stable second order system with elevator as input [45]. Thus one possibility would be to differentiate equation 3.3 to get flat inputs at the same system order as elevator and aileron. However, feedback for such an approach would require the third derivative of the actual system motion which is not directly available (derivative of acceleration). Further, the dynamics in pitch from elevator to normal acceleration and aileron to roll are fast relative to the derivatives of the path, and we can use the polynomial optimization to ensure this is always the case.

The approach we use for control is to integrate $\dot{a}_n$ and maintain a_n as controller state. Desired roll rate and normal acceleration are then translated algebraically into elevator, aileron, and rudder commands.

At steady state the sum of the pitching moments will be 0. Making typical aerodynamic assumptions, the pitching moment coefficient is given by

$$C_m = C_{m0} + C_{m\alpha}(\alpha - \alpha_0) + C_{mq}\frac{C_{\text{ref}}q}{2V} + C_{m\delta_e}\delta_e, \tag{3.86}$$

where α is the angle of attack, q is the pitching rate and δ_e is the elevator deflection. Using the relationship between angle of attack and normal acceleration and the normal acceleration due to pitch rate we get:

$$C_m = C_{m0} + C_{m\alpha}\alpha_0 + C_{m\alpha}C_{L0} + \left(\frac{mC_{m\alpha}}{\frac{1}{2}\rho S} + \frac{C_{mq}C_{\text{ref}}}{2}\right)\frac{a_n}{V^2} + C_{m\delta_e}\delta_e, \tag{3.87}$$

where ρ is the air density (not the path variable), and C_{L0} is the coefficient of lift at α_0. Setting equal to 0 and rearranging we have:

$$\delta_e = c_0\frac{a_n}{V^2} + c_1, \tag{3.88}$$

where c_1 and c_0 are constants. Using a similar approach for the rolling and yawing moments we have:

$$\delta_a = c_2\frac{p}{V} + c_3\frac{r}{V} \tag{3.89}$$

$$\delta_r = c_4\frac{p}{V} + c_5\frac{r}{V} \tag{3.90}$$

where p is the steady state roll rate, r is the steady state yaw rate, δ_a is the aileron deflection, and δ_r is the rudder deflection.

Equations 3.88, 3.89, and 3.90 give us the relationships we need. Coefficients c_0-c_5 can be empiracally fit from flight or simulation data and then used to map the desired normal acceleration to actualy control inputs.

3.5 Simulation Results

To validate our trajectory generation and control methodologies we used the physics engine from the open source CRRCsim flight simulator [1]. The simulator takes as input aerodynamic derivatives computed with AVL, an open source extended vortex lattice aerodynamic analysis program [17]. We constructed an AVL model of our fixed wing vehicle seen in figure 3-5.

We sampled 1000 random configurations and used them to generate Dubins paths, transverse polynomial paths enforcing 3rd order continuity, and transverse polynomial paths enforcing 4th order continuity. The nominal paths were then simulated with the full nonlinear model. The planning radius used in the Dubins curves was $R = 8$m with a desired flight velocity of 7m/s which places the vehicle close to it's dynamic limit. Polynomials of order $N = 10$ were used with cost of the zeroth-fourth derivatives, $c = \begin{bmatrix} 0.3 & 1 & 0 & 50 & 1 \end{bmatrix}$, and all other cost terms set to 0. The paths were sampled in a 40m square so they usually involve sequential turns without extended straight sequences. The samples were used to plan: Dubins paths, transverse polynomial paths up to 3rd order continuity, and transverse polynomial paths up to 4th order continuity. For computation time comparison we also include the full simulated model. Using a full model-based method to compute feasible trajectories would have computational cost on this order, as the full dynamics are simulated forward. The closed-loop RRT used on the Darpa Urban Challenge vehicle used this technique [36].

If the simulated vehicle deviated by more than 10m from the planned path, the case was marked as a failure and discarded. These samples were used to generate the results shown in table 3.1. We can see that for these conditions the pure Dubins curves fail more than 19% of the time. Additionally the average normed tracking error is more than double what it is for both of the transverse polynomial paths. Interestingly, fourth order continuity paths have slightly higher normed error with slightly lower failure rate. The failure cases of the transverse polynomial paths are likely due to rare cases of poor geometries leading to infeasible trajectories in the optimization which could be discarded with dynamic constraint checking. The higher normed error for the fourth order paths probably indicates that within the fidelity of our fit control mappings, the continuity in roll rate is not important. However, the optimization still minimizes roll acceleration effort, which clearly increases the feasibility of the paths compared to the fully discontinuous Dubins case.

The computation time of the transverse polynomial paths is dominated by the matrix inversion of the piecewise polynomial QP. While it is substantially slower than the Dubins calculation alone, it is still more than 20x faster than a single simulation of the full model, and to get an exact solution, multiple simulations would be required, as in the shooting method [57].

Figures 3-6 and 3-7 give some intuition for why the optimized paths are supe-

Path Type	Error Norm (m)	Fraction Failed	Average Computation Time (microseconds)
Dubins	1.60	0.194	20.0
3rd Order	0.67	0.006	495.7
4th Order	0.75	0.003	516.6
Full Model	-	-	10609.3

Table 3.1: This table shows path following error, fraction of simulations diverged, and computation time for a path for Dubins curves, transvers-polynomial curves enforcing continuity up to 3rd and 4th order, and a full simulation of the nonlinear model.

rior for tracking. Each depicts example tracking performance with the coordinated flight model for both Dubins curves and the same Dubins curve with an optimized polynomial offset. As we would expect, the coordinated flight model exhibits perfect tracking behavior on these paths.

Since the coordinated flight model tracks the paths exactly, the deviation must be introduced by errors in the control mappings from the coordinated flight model to the full nonlinear model (equations 3.88, 3.89, and 3.90). This is not surprising since we approximated the mapping using simple linear fitting, treating the nonlinear model as a black box. However, to get the best performance on an actual vehicle this is where it would make sense to spend the most effort. Ideally the fitting would take place on flight data exhibiting coordinated flight, so one possibility would be to iteratively fit and collect data under closed loop control until some threshold of tracking error is attained. We leave this polishing step as future work on the actual vehicle.

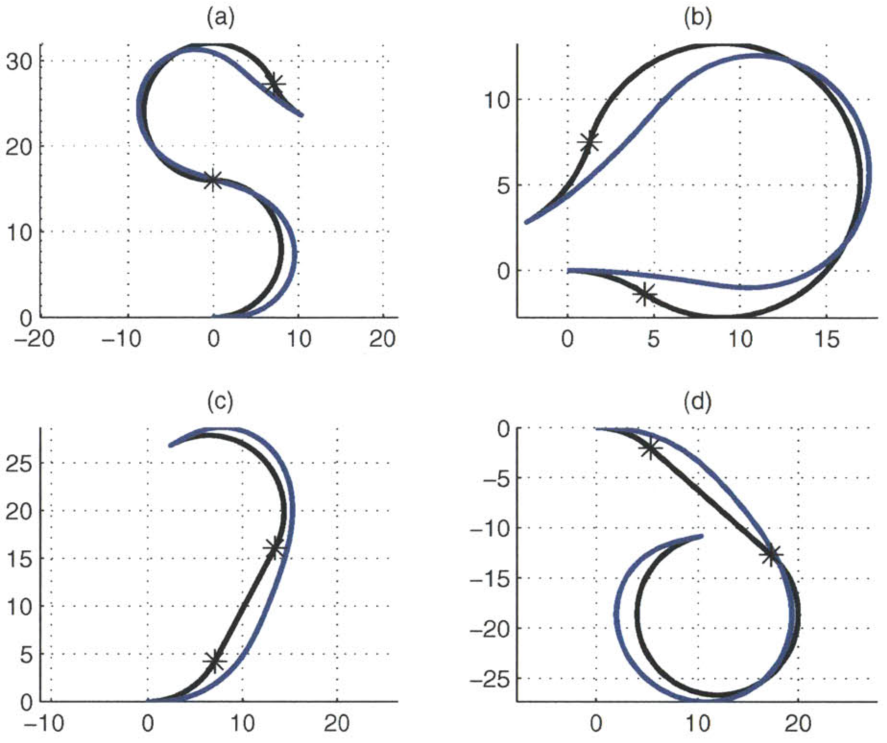


Figure 3-3: This figure shows Dubins-Polynomial paths for various configurations. The blue line in the transverse polynomial path and the black line is the Dubins path from which it is offset. For each of these examples, the curvature is constrained to be 0 at the start (0, 0, and facing left) and end of the path. We can see that generally the optimization yields paths that distribute the shifts in curvature around the Dubins segment junctions (denoted with black stars on the paths). Deviation from the nominal path is only lightly penalized, so we can see in (c) for example, the optimized path maintains deviation through the straight segment, and thus reducing the need for "sharpness" entering the second left turn. Conversely in (a) the optimization yields a path that smooths the slope between the initial left turn and the following right turn, thus reducing the roll action required. It is important to note that the optimization does not respect a hard turning radius constraint which we can see clearly in (b) where the entry and exit left turns are smoothed out at the expense of a sharpening of the right turn.

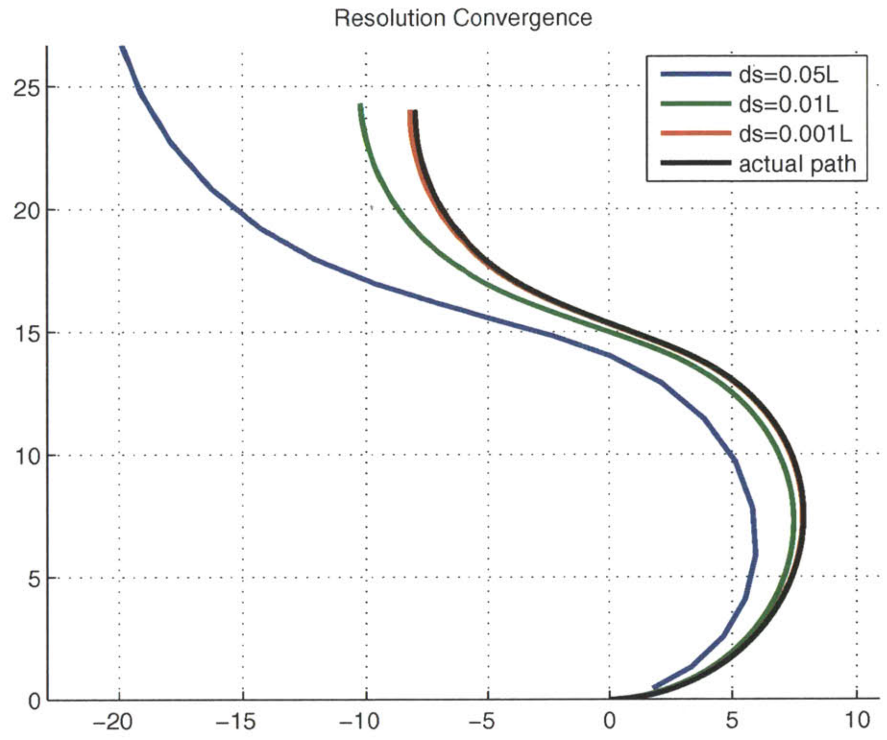


Figure 3-4: This figure shows convergence as the third path derivative, $\dddot{\Delta}$, is integrated numerically for finer step sizes ds as fraction of total path length L. Two things are important to note. If the third derivative expression was not incorrect, we would not be able to obtain the actual path by integrating it, and secondly, if the path were not continuous up to the third derivative, integration would not converge to the actual path.

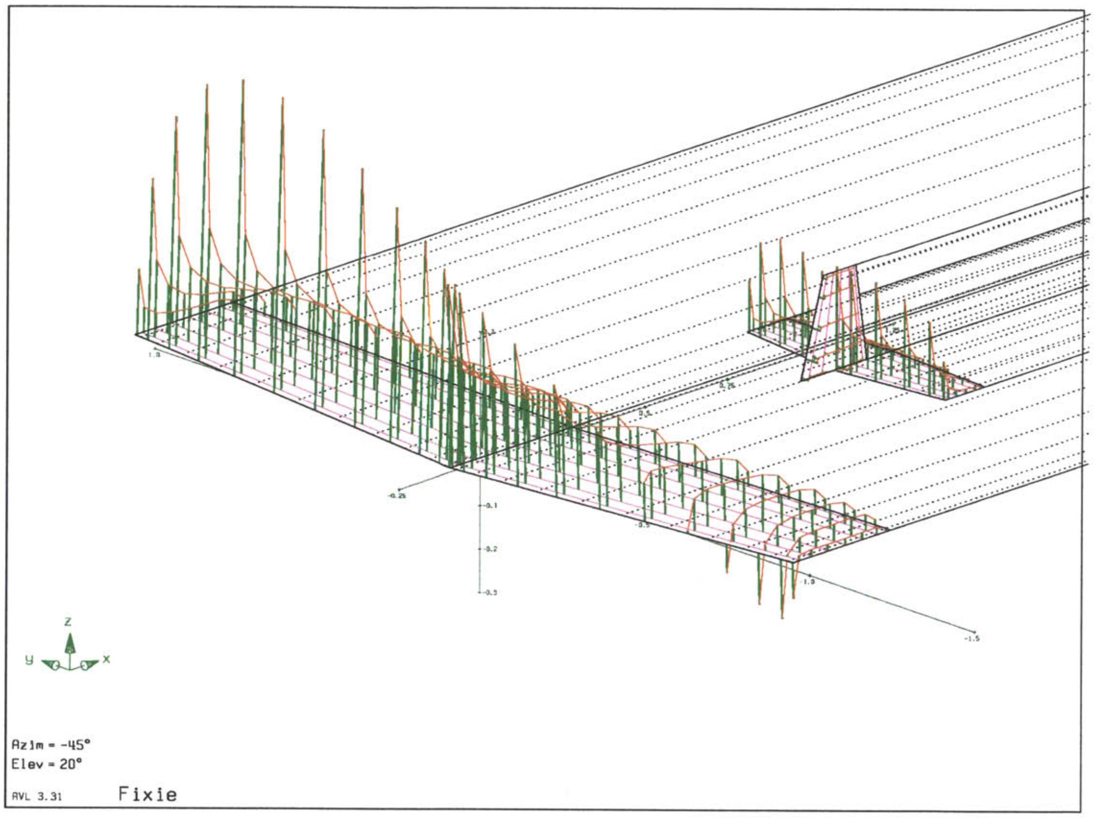


Figure 3-5: The AVL model used for aerodynamic analysis. The purple lines along the lifting surfaces show the vortex lattice discretization, while the orthogonal green lines show the lift distribution, and the dotted black lines show the shed trailing vortices. In this figure the vehicle is rolling to the right and pitching up which is why the right wing and tail are more heavily loaded.

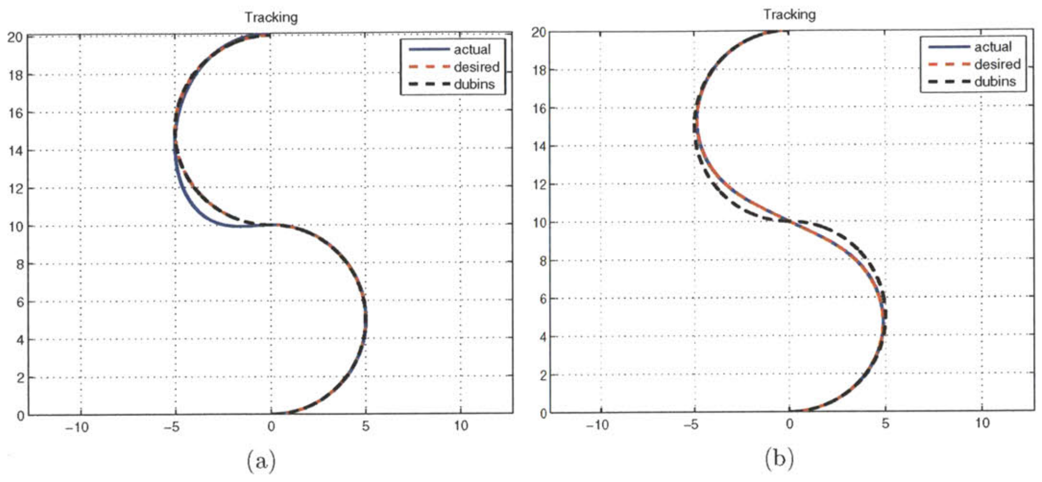


Figure 3-6: This figure shows tracking performance for the coordinated flight model for a Dubins curve (a) and a transverse polynomial path (b). We can see the actual and desired paths match for the transverse polynomial path as the optimization "anticipates" the discontinuity and symmetrically distributes the roll angle change into each arc segment. In contrast, when tracking the Dubins curve, the vehicle can not roll instantaneously and thus deviates from the path before stabilizing back onto it.

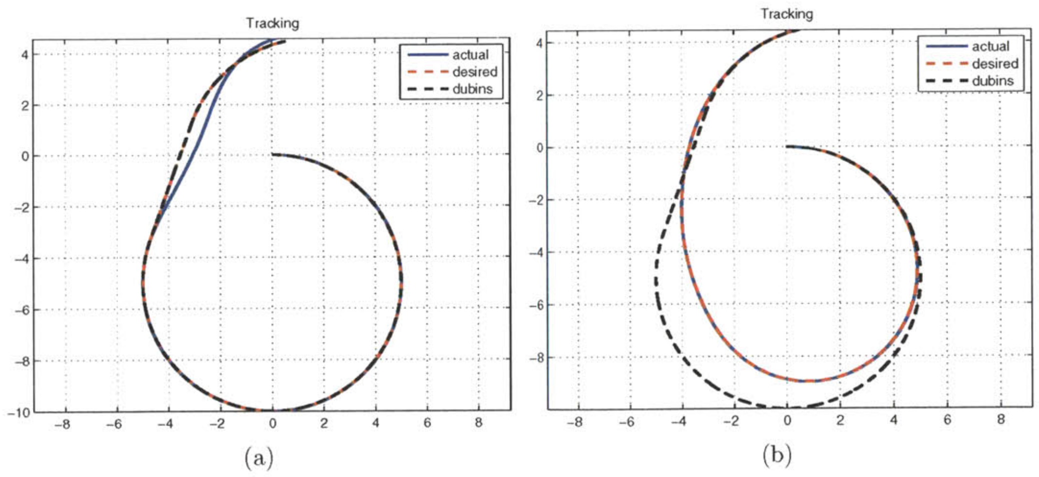


Figure 3-7: This figure shows another example with the pure Dubins curve (a) and the transverse polynomial path (b). We can see the optimization smoothly distributes curvature over the path, leading to perfect tracking performance, whereas with a pure Dubins curve the tracking deviates at the segment junction.

Chapter 4

State Estimation

To close the loop around the trajectory we must know the state of the vehicle. Since we're operating without the use of GPS, the state must be inferred by sensor readings and knowledge of the rigid-body dynamics. The state estimation algorithm is responsible for taking as input sensor readings from the IMU and laser range scanner and estimating the position and velocity of the vehicle to stabilize nominal trajectories. Our approach a Gaussian Particle Filter (GPF) based update step together with an EKF process model on the IMU to efficiently and compactly filter the sensor inputs. We also provide a method for estimating the noise parameters of the IMU without using ground truth data.

4.1 State Estimation Problem Statement

Assuming the MAV to be a rigid body and neglecting higher order effects such as propeller speed and time-varying airflow over the vehicle, the state of a MAV is given by its position and orientation and the associated linear and angular velocities. For control purposes it is convenient to represent the velocities in body coordinates.[1] Thus the goal of the filter is to estimate the quantities $\begin{bmatrix} \omega_b^T & v_b^T & R & \Delta^T \end{bmatrix}^T$ where $\omega_b = \begin{bmatrix} \omega_{b1} & \omega_{b2} & \omega_{b3} \end{bmatrix}^T$ is the angular velocity in body coordinates, $v_b = \begin{bmatrix} v_{b1} & v_{b2} & v_{b3} \end{bmatrix}^T$

[1]As discussed in chapter 3 the control is actually computed in the velocity frame which differs from the body frame by the angle of attack.

is the linear velocity in body coordinates, R is the rigid body orientation rotation matrix, and $\Delta = \begin{bmatrix} \Delta_1 & \Delta_2 & \Delta_3 \end{bmatrix}^T$ is the translation vector from the origin in global coordinates to the origin of the body frame, expressed in global coordinates.

We assume a set of IMU measurements consisting of 3-axis acceleration, 3-axis angular rate measurements, and planar laser range scans. Further, we assume we have access to a 3D map of the environment represented as an occupancy grid. In addition, for planning purposes, we require a covariance over the estimated quantities Σ_{vehicle} such that $\Sigma_{\text{vehicle}} \approx \Sigma_{\text{actual}}$ where Σ_{actual} is the true covariance.

4.2 IMU Process Model

Our state estimation algorithm uses an Extended Kalman Filter (EKF) to propagate a Gaussian distribution over predicted states. The EKF process model is based on a discrete time, nonlinear discrete transition function:

$$x_{t+1} = f(x_t, u_t, w_t) \tag{4.1}$$

where x_t is the system state vector, u_t is the input vector to the system, and w_t is a random disturbance drawn from a normal distribution $N(0, Q)$. The EKF tracks the state at time t as a Gaussian distribution with mean μ_t and covariance Σ_t. These first two moments are propagated forward according to:

$$\bar{\mu}_{t+1} = f(\mu_t, u_t, 0) \tag{4.2}$$

$$\bar{\Sigma}_{t+1} = A_t \Sigma_t A_t^T + W_t Q W_t^T \tag{4.3}$$

where $\bar{\mu}$ and $\bar{\Sigma}$ denote predicted quantities before a measurement update has occurred, and A_t and W_t are the appropriate partial derivatives f.

4.2.1 Exponential Coordinates Attitude Uncertainty

We track orientation uncertainty in perturbation rotations in the body frame. If the true orientation is given by the rotation matrix R, we can write $R = \hat{R}R(\chi)$ where $\hat{R}$ is the estimated orientation and $R(\chi) = e^{\chi^\wedge}$ is the error rotation matrix. $\chi \in \Re^3$ is the perturbation rotation about the body axes. We use the $^\wedge$ notation to denote the skew symmetric matrix formed as:

$$\chi^\wedge = \begin{bmatrix} 0 & -\chi_3 & \chi_2 \\ \chi_3 & 0 & -\chi_1 \\ -\chi_2 & \chi_1 & 0 \end{bmatrix} \tag{4.4}$$

Taking the exponential of a skew symmetric matrix returns a rotation matrix corresponding to a rotation of $|\chi|$ about the axis defined by χ where χ is referred to as the exponential coordinates of rotation.

In our expression for the true orientation, $R(\chi)$ post multiplies $\hat{R}$ which puts the perturbations in the body frame. Since the error is parameterized by χ and the covariance can be tracked in a 3×3 matrix Σ_χ. The covariance can be thought of as cones of uncertainty surrounding the body frame axes defined by the columns of $\hat{R}$. A sketch of this uncertainty is shown in figure 4-1 for the covariance (in degrees):

$$\Sigma_\chi = \begin{bmatrix} 3^2 & 0 & 0 \\ 0 & 5^2 & 0 \\ 0 & 0 & 15^2 \end{bmatrix} \tag{4.5}$$

This choice of coordinates for the filter error is desirable since fundamentally rigid body orientation, denoted mathematically as the special orthogonal group (SO3), has three degrees of freedom. While any three element representation is provably singular for some orientation, more commonly used parameterizations (i.e., quaternions or rotation matrices) will have constraints between the elements of the representation. Thus a linearized filter covariance over the parameters will not be full rank. Numerical errors would pose the constant threat of creating negative eigenvalues, and thus blowing up the estimator. Furthermore, an efficient linearized measurement update as is commonly used in Gaussian filters does not respect the constraints and thus does not map onto SO3. A renormalization scheme could be used after every update, but at any given time the representation can be arbitrarily poor [56].

As we will see, the attitude uncertainty representation is agnostic to the actual underlying orientation integration and tracking. Quaternions and rotation matrices are easy to update based on using χ in the estimator state vector μ.

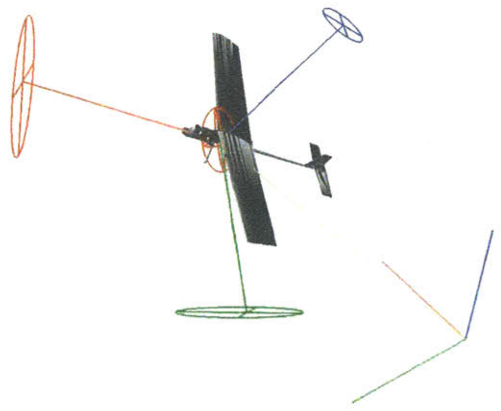


Figure 4-1: This figure shows the uncertainty representation in body axes. We see that high variance on the z axis perturbation maps into large motions for the x and y bases, all in the body frame.

4.3 Process Equations

The equations of motion for a rigid body are given by:

$$\begin{aligned}
\dot{\omega}_b &= J^{-1}(-\omega_b \times J\omega_b + \tau_b) && (4.6)\\
\dot{v}_b &= -\omega_b \times v_b + R^T\bar{g} + a_b && (4.7)\\
\dot{R} &= R\omega_b^{\wedge} && (4.8)\\
\dot{\Delta} &= Rv_b, && (4.9)
\end{aligned}$$

where τ_b is the torque applied to the body and a_b is the acceleration in body coordinates. Since the IMU returns noisy measurements of ω_b and a_b, we follow the commonly used technique of omitting ω_b from the state, neglecting equation 4.6, and treating the IMU measurements as inputs to the filter.

For the quantities used in equation 4.2 we have

$$x = \begin{bmatrix} v_b & \chi & \Delta \end{bmatrix} \qquad (4.10)$$

$$u = \begin{bmatrix} u_{\text{gyro}} & u_{\text{accel}} \end{bmatrix} \qquad (4.11)$$

$$w = \begin{bmatrix} w_{\text{gyro}} & w_{\text{accel}} \end{bmatrix} \qquad (4.12)$$

$$\qquad (4.13)$$

Combining this state representation with equations 4.7-4.9.

$$f_c(x_t, u_t, w_t) = \begin{bmatrix} \dot{v}_b \\ \dot{R} \\ \dot{\Delta} \end{bmatrix} \qquad (4.14)$$

$$= \begin{bmatrix} -\omega_b \times v_b + R^T\bar{g} + gu_{\text{accel}} \\ Ru_{\text{gyro}}^{\wedge} \\ Rv_b \end{bmatrix} \qquad (4.15)$$

Taking the appropriate partial derivatives we get:

$$\frac{\partial \dot{v}_b}{\partial x} = \begin{bmatrix} -\omega_b^\wedge & (R^T\bar{g})^\wedge & 0 \end{bmatrix} \tag{4.16}$$
$$\frac{\partial \dot{\chi}}{\partial x} = \begin{bmatrix} 0 & -\omega_b^\wedge & 0 \end{bmatrix} \tag{4.17}$$
$$\frac{\partial \dot{\Delta}}{\partial x} = \begin{bmatrix} R & -Rv_b^\wedge & 0 \end{bmatrix} \tag{4.18}$$

for a continuous dynamics linearization of:

$$A_c = \frac{\partial f}{\partial x} = \begin{bmatrix} -\omega_b^\wedge & (R^T\bar{g})^\wedge & 0 \\ 0 & -\omega_b^\wedge & 0 \\ R & -Rv_b^\wedge & 0 \end{bmatrix} \tag{4.19}$$

and for the input vector we have:

$$\frac{\partial \dot{v}_b}{\partial u} = \begin{bmatrix} v_b^\wedge & gI \end{bmatrix} \tag{4.20}$$
$$\frac{\partial \dot{\chi}}{\partial u} = \begin{bmatrix} I & 0 \end{bmatrix} \tag{4.21}$$
$$\frac{\partial \dot{\Delta}}{\partial u} = \begin{bmatrix} 0 & 0 \end{bmatrix} \tag{4.22}$$
$$W_c = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial \dot{v}_b}{\partial u} \\ \frac{\partial \dot{\chi}}{\partial u} \\ \frac{\partial \dot{\Delta}}{\partial u} \end{bmatrix}. \tag{4.23}$$

While more sophisticated approximations could be used, we construct the discrete quantities for the filter f, A_t, and W_t using Euler integration:

$$f(x_t, u_t, 0) = x_t + f_c(x_t, u_t, 0)dt \tag{4.24}$$
$$A_t = I + A_c dt \tag{4.25}$$
$$W_t = W_c dt. \tag{4.26}$$

We integrate the attitude separately as

$$R_{t+1} = R_t R(u_{\text{gyro}}^{\wedge}) \tag{4.27}$$

4.4 Identifying the Process Noise Parameters

The process noise covariance Q is a diagonal matrix populated as

$$Q = \begin{bmatrix} q_{\text{gyro}} I_3 & 0 \\ 0 & q_{\text{accel}} I_3 \end{bmatrix} \tag{4.28}$$

and q_{accel} and q_{gyro} are the parameters we wish to identify. Two issues lead to difficulties with the conventional approach to system identification techniques, where the vehicle trajectory is tracked using an external measurement system, and random deviations from the reference trajectory are modeled as process noise parameters. First, the way the noise projects onto the state changes with the time varying W_t matrix such that the Q matrix cannot be recovered in closed form simply by squaring the deviations. More importantly we cannot depend on the availability of ground truth measurements of the vehicle's deviation from a reference trajectory since it is generally very difficult to obtain realistic flight data for MAVs with labeled ground truth as size and weight restrictions prevent the use of highly accurate GPS systems, and only the smallest and lightest fixed wing vehicles (which are incapable of carrying a meaningful sensor payload) can fly in motion capture systems.

Nonetheless it is critical that the model parameters, and especially the process noise parameters, be accurate. For planning purposes we must be able to predict distributions over future states to guarantee safe trajectories. Within the context of state estimation and Monte-Carlo localization, as we describe in section 4.4.2, it is critical that an accurate covariance of the state estimate be maintained when sensor data is sparse or absent, such that the state estimate can be can be properly distributed to obtain measurements when they become available.

While we do not have access to ground truth with which to estimate the noise parameters, we can post-process data using a Kalman smoothing algorithm to obtain a state history $X = \begin{bmatrix} \hat{x}_0 & \hat{x}_1 & \dots & \hat{x}_T \end{bmatrix}$ with the error associated with each smoothed state estimate given by

$$\Gamma_t = E\left[(\hat{x}_t - x_t)(\hat{x}_t - x_t)^T\right] \tag{4.29}$$

The key idea in our approach is in projecting the process noise forward over multiple time steps such that the process noise dominates the error in the smoothed estimate, thus allowing us to treat the smoothed estimate as ground truth. Additionally, by projecting the noise forward over multiple steps, the parameters we identify will be suitable for use in planning algorithms that require open-loop type predictions [11] and the parameters will work with intermittent measurement functions. It is important to note that while the Kalman smoothing algorithm returns a covariance history as well as a state history, it may be arbitrarily far from the actual error Γ_t since the system is nonlinear and the smoother is run with the wrong noise parameters since the true parameters are unknown. However, as long as the system is observable over the smoothing window the error will be bounded.

Using the linearized dynamics from the EKF we can project the filter covariance forward N steps by repeatedly applying equations 4.3 4.3. Neglecting the error on the smoothed estimate, we obtain the expression:

$$E\left[(\hat{x}_{t+N} - \hat{x}_t)(\hat{x}_{t+N} - \hat{x}_t))^T\right] = \Sigma_{t,N} \tag{4.30}$$

$$= \sum_{i=0}^{N-1} G_{t+i,N} Q G_{t+i,N}^T \tag{4.31}$$

where $G_{t,N} = \prod_{j=t+1}^{t+N-1} A_j W_t$. This is an important quantity for our noise identification algorithm because as it maps the noise at each time step onto the state vector at time $t+N$. For identifying characteristics of the process noise, A_t must be neutrally stable and W_t must have full column rank. If A_t is highly unstable, the $\Sigma_{t,N}$ will be overly sensitive to the noise values w_i for small i, whereas if A_t is highly stable, $\Sigma_{t,N}$ will be dominated by larger values of i and thus the forward projection offers little benefit. However, many robotic systems, including our IMU dynamics, exhibit approximately neutrally stable behavior.

For neutrally stable systems, as N gets large we expect $\Sigma_t >> \Gamma_t$. We can then divide up the data set X to get $M = T/N$ samples from prediction distributions obtained by subtracting the state at time $t_{\text{end}} = iN + N - 1$ from the state at time $t_{\text{begin}} = iN$ for $i \in [0, M-1]$. This gives us M samples $y_i = x_{t_{\text{end}}} - x_{t_{\text{begin}}}$ drawn from distributions $N(0, \Sigma_{t_{\text{begin}},N}) = P(x_{t_{\text{end}}}|x_{t_{\text{begin}}})$. We have a joint likelihood function for our data given the parameters of Q as:

$$P(Y|x_0, Q) = \prod_{i=0}^{M-1} P(x_{iN+N-1}|x_{iN}, Q). \tag{4.32}$$

We would like to maximize this probability for which we use the log-likelihood function,

$$l(Y|x_0, Q) = -\frac{1}{2}\sum_{i=0}^{M-1} \log|\Sigma_i| + y_i^T \Sigma_i y_i. \tag{4.33}$$

From an intuitive standpoint we are optimizing for the q parameters that would produce the observed drift away from the smoothed estimate given by the samples y_i.

We setup and solve the optimization using standard nonlinear programming techniques. Specifically we use the interior point method implemented in Matlab to solve for the maximum likelihood values of q_{gyro} and q_{accel}.

4.4.1 Partioned GPF Measurement Update

While the EKF is effective for propagating the first two moments of the nonlinear dynamics through our IMU equations of motion, it is not well suited to integrating laser measurements from unstructured 3D environments. In contrast Monte-Carlo techniques are widely used in laser-based localization algorithms [53]. We use the Gaussian Particle Filter (GPF) to perform laser-based measurement updates [35].

In its standard form the GPF maintains a Gaussian distribution over the state space given a measurement history given by $P(x_t|z_{0:t}) = N(x_t; \mu_t, \Sigma_t)$. However, at each iteration of the filter, particles are used to incorporate nonlinear process and measurement models. To compute $P(x_{t+1}|z_{0:t})$ a set of samples $\{x_t^{(j)}\}_{j=1}^M$ is drawn from $N(\mu_t, \Sigma_t)$ and the samples are then propagated through the process model $f(x_t, u_t, w_t)$. To perform the measurement update the samples are weighted according to the measurement model $w_t^{(j)} = P(z_t|x_t^{(j)})$. The updated Gaussian at the end of an iteration of the filter is then obtained as the weighted mean and covariance of the samples

$$\mu_{t+1} = \frac{\sum_{j=1}^M w_t^{(j)} x_t^{(j)}}{w_t^{(j)}} \tag{4.34}$$

$$\Sigma_{t+1} = \frac{\sum_{j=1}^M w_t^{(j)} (x_t^{(j)} - \mu_{t+1})(x_t^{(j)} - \mu_{t+1})^T}{w_t^{(j)}}. \tag{4.35}$$

Assuming the underlying system is linear-Gaussian, the filter is shown to approximate the true distribution arbitrarily well with a large number of samples.

A straightforward implementation of the GPF for state estimation using a laser on a MAV is impractical and inefficient for two reasons:

1. IMU dynamics are well approximated by linearization as evidenced by the widespread use of EKFs in GPS-IMU state estimation, and thus using a particle process model adds significant computational burden while also introducing error through sampling.

2. The IMU filter maintains additional states to track velocity and IMU biases, however the laser measurements are only a function of the position and orientation, parameterized by Δ and χ in our formulation. In fact, most of the orientation information in the measurement exists in the plane of the laser contained in χ_z.

To address the first issue we only use the GPF to perform the measurement update, and instead of propagating samples through the measurement function, we sample directly from the prior distribution returned by the EKF after the process step, $N(\bar{\mu}, \bar{\Sigma})$. To address the second issue above we explicitly partition the state according to independence relationships in the measurement function. We perform a standard GPF measurement update on the partitioned state and use this to compute a pseudo measurement which is then used to update the full state.

The state is partitioned as,

$$x_t = \left[\begin{array}{cc} x_t^m & x_t^p \end{array} \right], \tag{4.36}$$

where $x_t^m \in \Re^k$ is the part of the state that affects the measurement, and $x_t^p \in \Re^{n-k}$ is independent from the measurement. More formally we assume our measurement function has the form

$$z_t = h(x_t^m, v_t), \tag{4.37}$$

permitting the independence factorization

$$P(z_t|x_t^p, x^m) = P(z|x^m). \tag{4.38}$$

We can similarly partition the covariance

$$\bar{\Sigma}_t = \left[\begin{array}{cc} \bar{\Sigma}_t^{(m^2)} & \bar{\Sigma}_t^{(mp)} \\ \bar{\Sigma}_t^{(pm)} & \bar{\Sigma}_t^{(p^2)} \end{array} \right]. \tag{4.39}$$

To perform the measurement update we draw samples $\{x_t^{m(j)}\}_{j=1}^M$ from $N(\bar{\mu}_t^m, \bar{\Sigma}_t^m)$. The samples are weighted with the measurement function in equation 4.38. From these weighted samples we can compute an update for $P(x_t^m|z_0 : z_t)$ using the conventional GPF weighted mean and covariance as in equations 4.34 and 4.35. The key idea is to then use the GPF update on the state variables that affect the measurement to propagate a Kalman update to the rest of the state.

To perform a Kalman measurement update we need to know the measurement value z_t, the covariance of the measurement R, and the observation matrix C. Firstly, we set C as a selector matrix for the measurement part of the state

$$C = \left[\begin{array}{cc} I_k & 0_{n-k} \end{array} \right]. \tag{4.40}$$

A measurement update on x^m would proceed as:

$$K^m = \bar{\Sigma}_t^m (C^m)^T (C^m \bar{\Sigma}_t (C^m)^T + R)^{-1} \tag{4.41}$$

$$\mu_t^m = \bar{\mu}_t^m + K^m (z_t - C^m \bar{\mu}_t^m) \tag{4.42}$$

$$\Sigma_t^m = (I - K^m C^m) \bar{\Sigma}_t^m \tag{4.43}$$

Plugging in the identity matrix for C^m, the above equations can be solved for R_t

$$\Sigma_t^m = \bar{\Sigma}_t^m - \bar{\Sigma}_t^m (C^m)^T (C^m \bar{\Sigma}_t (C^m)^T + R_t)^{-1} \bar{\Sigma}_t^m \tag{4.44}$$

$$R_t = (\bar{\Sigma}_t^{m^{-1}} - \bar{\Sigma}_t^{m^{-1}} \Sigma_t^m \bar{\Sigma}_t^{m^{-1}})^{-1} - \bar{\Sigma}_t^m \tag{4.45}$$

$$= (\Sigma_t^{m^{-1}} - \bar{\Sigma}_t^{m^{-1}})^{-1} \tag{4.46}$$

where we make use of the matrix inversion lemma between equations 4.45 and 4.46.

Using R_t we can now solve for the Kalman gain that would have produced the same change between our prior and posterior covariance using equation 4.41 and then recover the actual measurement that would have produced the same change in the mean of prior vs. posterior distributions:

$$z_t = K^{m^{-1}} (\mu_t^m - \bar{\mu}_t^m) + \bar{\mu}_t^m. \tag{4.47}$$

A Kalman gain for the entire state can then be computed using R_t and z_t, and a standard Kalman measurement update performed.

The posterior distribution quantities $\mu_t^{m^{-1}}$ and $\Sigma_t^{m^{-1}}$ are readily available from the GPF measurement update on the measurement part of the state vector. Naively one might use the Gaussian prior from which the samples were drawn to evaluate equations 4.46 and 4.47. However, the quantities we care about R_t and z_t are obviously sensitive to the difference between the prior and posterior mean and covariance. With a finite number of samples there will be some error between the distribution described by the sample set $\{x_t^{m(j)}\}_{j=1}^M$ and the Gaussian prior. This error will carry over to the weighted sample set which approximates the posterior. We can compensate by using the mean and covariance of the prior sample distribution instead of our analytic expressions for $\bar{\mu}_t^m$ and $\bar{\Sigma}_t^m$. In practice, this substitution makes an enormous difference, particularly with low numbers of particles (which is highly desirable in a real-time application).

Finally, we note that the solutions for R_t and z_t hinge on the invertibility of C^m which is a proxy for the invertibility of our measurement function h in equation 4.37 with respect to x_t^m. It can be difficult to know *a priori* if the measurement is well conditioned or invertible. If it is not (i.e., if the measurement doesn't actually contain information about some piece of x_t^m) then our R_t value may not be positive definite, leading to a filter divergence. Thus it is necessary in practice to perform an eigenvalue decomposition on R_t and set any negative eigenvalues to a large constant (implying no information gain along the corresponding eigenvector) and then reconstruct the matrix. This step also protects the algorithm from negative eigenvalues entering through sampling related errors.

4.4.2 Laser Localization

We generate 3D voxel maps of environments using orthogonal Hokuyo lidars mounted on a rolling tripod. A scan matching algorithm developed in prior work [5] runs using the horizontally mounted lidar while the vertical lidar sweeps out a 3D point cloud of the environment. The point cloud is then projected into a 3D occupancy map computed using OctoMap [58].

The likelihood evaluation proceeds according to standard techniques used in 2D localization. We blur the map using a Gaussian kernel around occupied cells. To perform particle measurement updates we project the scan from that time-step into the map and sum the log-likelihood of the reached cells before exponentiating to obtain a probability with which to weight the particles.

The exponential coordinates framework of the filter provides an easy framework for particle generation. To sample in orientation we sample $\{\chi_t^{m(j)}\}_{j=1}^M$ from $N(0, \bar{\Sigma}_{\chi t}^m)$ and then exponentiate and post multiply our predicted orientation quaternion $q_t^{m(j)} = \bar{q}_t q(\chi_t^{m(j)})$. An interesting question is the appropriate partitioning of the state vector for the updates described in section 4.4.1.

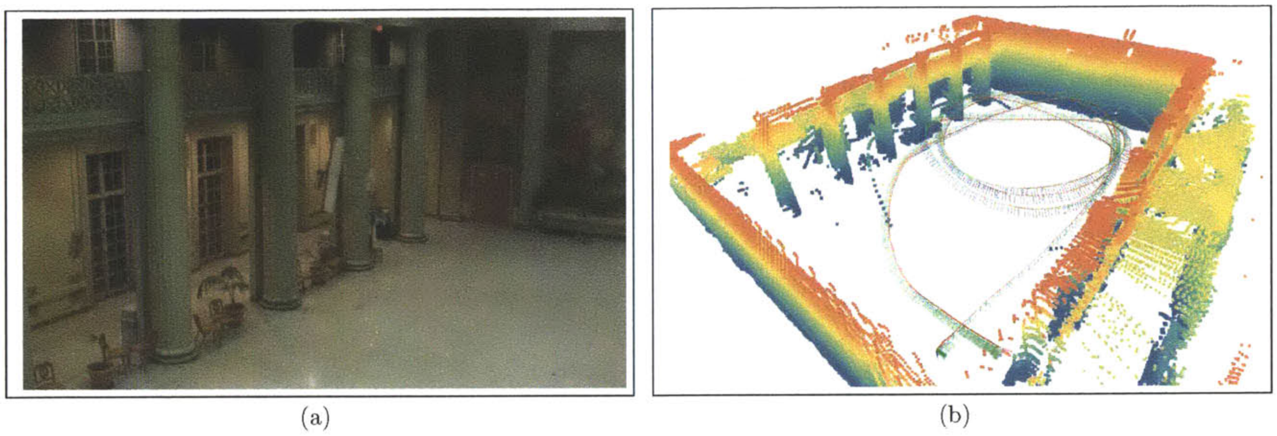


Figure 4-2: A picture of the indoor space (a) where we flew our fixed wing vehicle. The space is roughly 12 meters by 20 meters and our vehicle flies between 6 and 10 m/s, thus aggressive maneuvering and tight turning is required to stay airborn. The trajectory flown by the vehicle is shown by the red, green, and blue axes in (b). The quality of the state estimates is evident in the (height colored) point cloud rendered using the state estimates of our algorithm. The floor and ceiling were cropped for visual clarity.

The use of planar LIDARs to localize in the plane is ubiquitous, suggesting that when working in 3D, laser range scans should at least contain information about the xy plane and χ_z (orientation about the yaw axis of the vehicle). However, in general, a planar slice of a 3D environment may contain some information about the full orientation, but populating the 6D pose space parameterized by χ and Δ with particles may produce limited extra information relative to the computational cost incurred, especially because the direct formulation for our filter based on exponential coordinates, is capable of inferring attitude from accurate position (xyz) measurements. We investigate different choices for x^m in our experiments.

4.5 Experimental Results

Our experimental platform consists of a custom built fixed-wing vehicle carrying a payload of a Hokuyo UTM-30LX laser rangefinder, a Microstrain 3DM-GX3-25 IMU, and a 1.6GHz Intel Atom base flight computer. To identify the noise parameters of

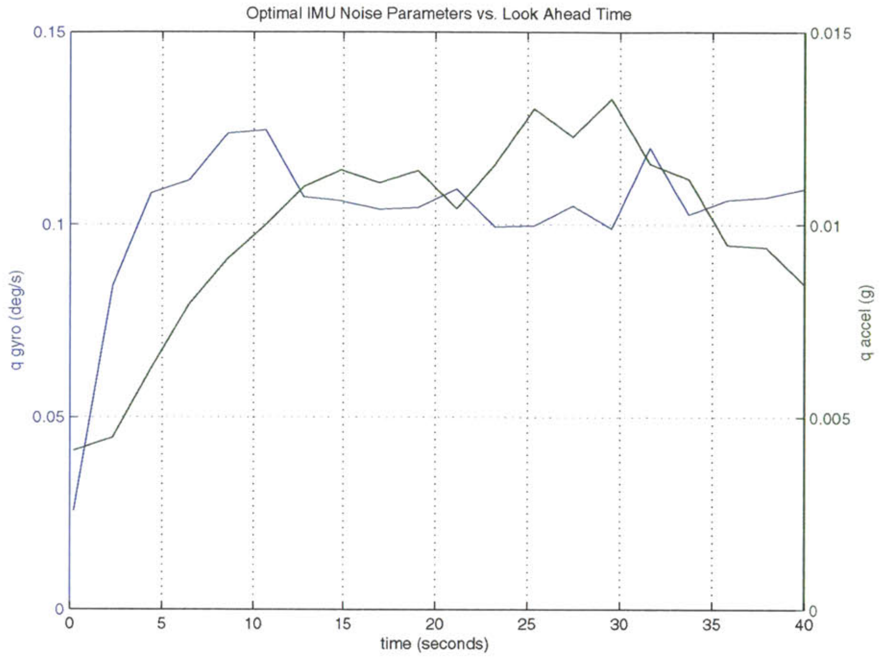


Figure 4-3: This figure shows values for q_{gyro} and q_{accel} obtained by optimizing equation 4.33 for different look ahead values of N. For small N the optimal noise parameters are dominated by the error in the smoothed estimates, Γ_t, but we see for large N consistent values are reached. It is interesting to note that as N increases fewer "samples" are available from a data set of fixed size, and thus more variance appears in the computed noise values, implying some optimal lookahead window to identify the parameters.

the IMU during realistic flying, we flew the vehicle outdoors with a low cost uBlox GPS unit. Optimized noise parameters as a function of the lookahead window N are depicted in figure 4-3. For small N the correlation and error in the smoothed estimate corrupts the optimized noise parameters, but as expected for larger N we can recover the true values. This is reflected in the convergence of the optimal values as N is increased. Using these optimized values to predict the uncertainty we obtain the results in figure 4-4 which shows predicted and empirical normed error as a function of look ahead. We get excellent agreement between the predicted values obtained from our model's covariance and the empirical values computed from lookahead in the data.

We conducted a number of flight tests in the indoor environment shown in Fig-

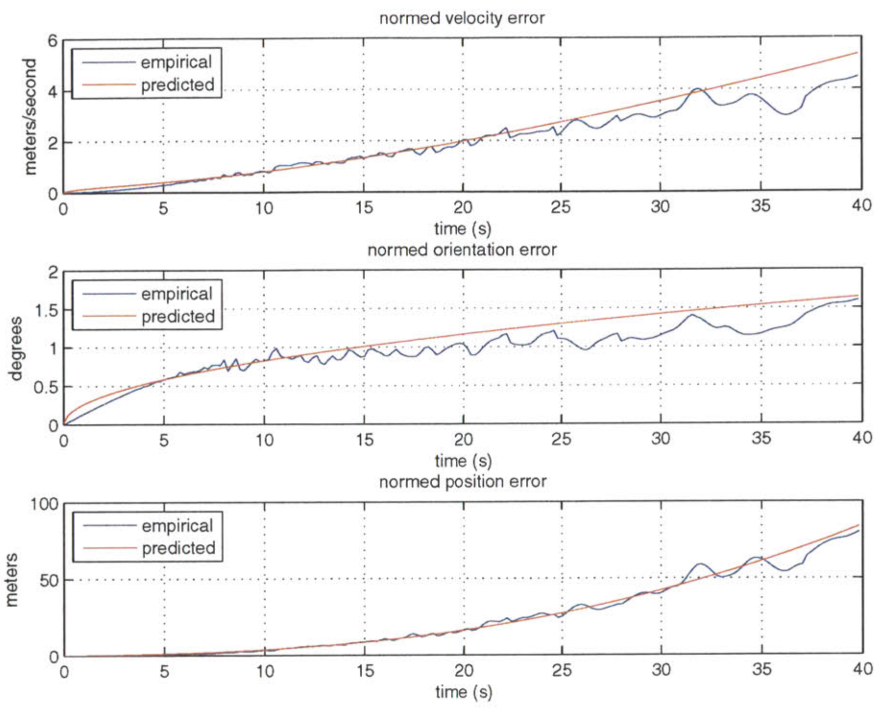


Figure 4-4: This figure shows the predicted normed error and the actual normed deviation from the smoothed estimates as a function of lookahead N. With the optimized values we can accurately predict uncertainty for both estimation and planning purposes.

ure 4-2(a). While we did not have access to any sort of ground truth state estimates, we were able to test our algorithms on real flight data. The accuracy of our state estimates were then validated qualitatively by looking at the accurate reconstruction of the 3D environment by reprojecting the laser points using our state estimates. One such 3D point cloud is shown in Figure 4-2(b). The fact that the point cloud matches the structure of the environment with the reprojected scans falling in close alignment on the structure (qualitatively) highlights the accuracy of our algorithm.

To quantify the error of the state estimator, we aggressively maneuvered the sensing payload in a high accuracy VICON motion capture studio. These ground truth state estimates allow us to evaluate the properties of our state estimation algorithm. Results for different number of particles and different state partitioning are summarized in figure 4-5. We can see that by not partitioning the state and performing

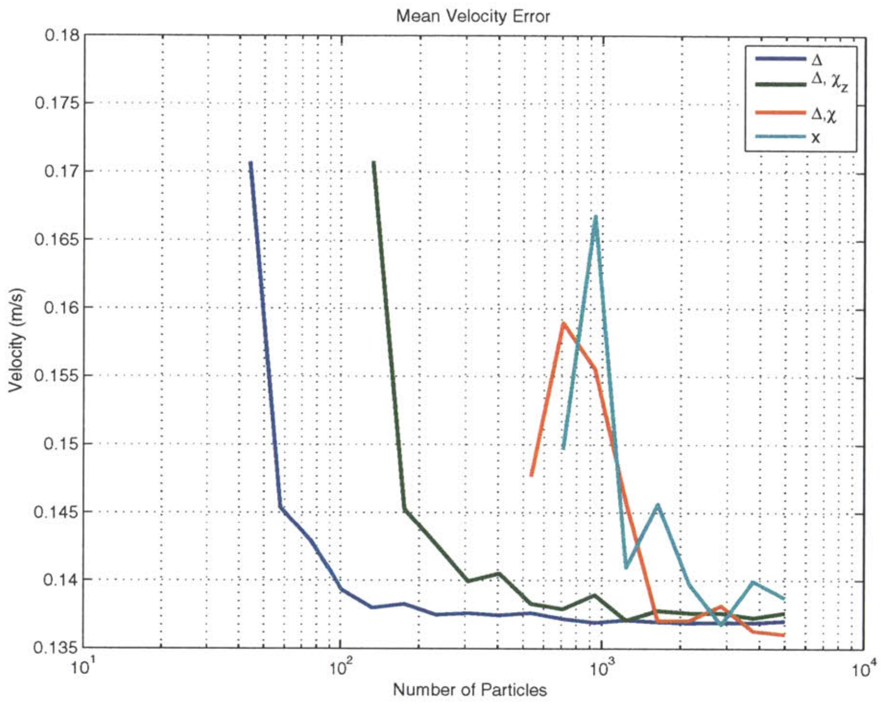


Figure 4-5: This figure shows the mean velocity error verses the number of particles used in the GPF for different state partitions in the measurement. As expected, the more states we use in the measurement function the more particles are required to obtain satisfactory estimates. In a naive implementation where the full state is used in the measurement and thus a standard GPF update performed, we require 2500 particles to get similar performance to a measurement update in Δ using only 100 particles. Thus our algorithm yields an effective 25x speedup.

standard GPF updates we incur significant computational cost in terms of number of particles to achieve the same level of accuracy. This makes sense given that we are using particles to capture the same correlations that are well captured analytically by the pseudo Kalman measurement update.

The experiments demonstrate the ability of our algorithm to maintain accurate state estimates in the face of fast motion. While a naive implementation of the GPF measurement update correctly estimates the state of the vehicle with a sufficient number of particles, the required number of particles is dramatically larger than for the partitioned state version. The naive GPF implementation would not be able to run in real time onboard the vehicle given the computation power available.

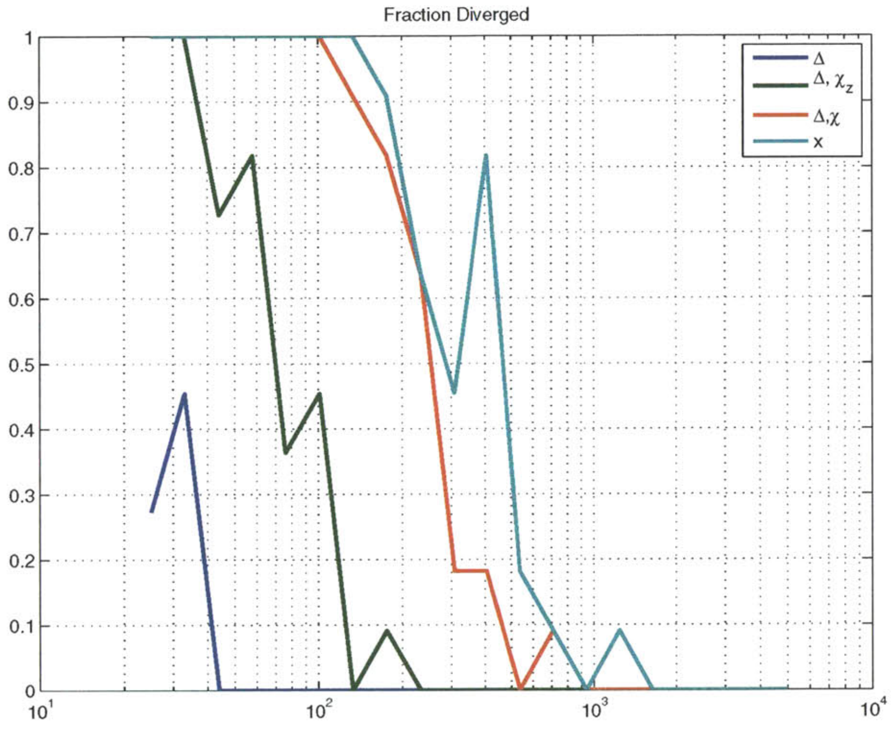


Figure 4-6: This figure shows the percentage of trials where the filter diverged as a function of the number of particles for different state partitions. Generally, the more of the state included in the measurement, the more particles are required to prevent divergence.

Chapter 5

Rapidly-exploring Random Belief Trees

We have seen in the previous chapter that it is possible to accurately estimate the vehicle state in a 3D environment. However, the quality of the localization solution can vary with the environment geometry. To ensure good performance for an autonomous system using these state estimates it is necessary to reason explicitly about uncertainty in the planning phase. In this chapter we introduce an algorithm for this purpose. We present the algorithm generally and then apply it to the fixed-wing planning problem in simulation.

5.1 Problem Formulation

The systems we are interested in are generally nonlinear and partially observable. The robot is given a discrete time description of its dynamics and sensors,

$$x_t = f(x_{t-1}, u_{t-1}, w'_t), \qquad w'_t \sim N(0, Q') \tag{5.1}$$

$$z_t = h(x_t, v'_t), \qquad v'_t \sim N(0, R'), \tag{5.2}$$

where $x_t \in \mathcal{X}$ is the state vector, $u_t \in \mathcal{U}$ is the input vector, w'_t is an random process disturbance, z_t is the measurement vector, and v'_t is a random component in the sensor readings. The state space $\mathcal{X}$ can be decomposed into $\mathcal{X}^{\text{free}}$ and $\mathcal{X}^{\text{obs}}$ where $\mathcal{X}^{\text{obs}}$ represents the states where the robot is in collision with obstacles.

Our approach to planning is built on an underlying method for finding dynamically feasible solutions and for stabilizing the system. Thus, we assume the availability of a CONNECT() function for finding a nominal trajectory and stabilizing controller between two states x^a and x^b such that,

$$(\check{X}^{a,b}, \check{U}^{a,b}, \check{K}^{a,b}) = \textsc{Connect}(x^a, x^b) \tag{5.3}$$

$$\check{X}^{a,b} = (\check{x}_0, \check{x}_1, \check{x}_2, \dots, \check{x}_{T_{a,b}}) \tag{5.4}$$

$$\check{U}^{a,b} = (\check{u}_0, \check{u}_1, \check{u}_2, \dots, \check{u}_{T_{a,b}}) \tag{5.5}$$

$$\check{x}_0 = x^a, x^b = f(\check{x}_{T_{a,b}}, \check{u}_{T_{a,b}}, 0) \tag{5.6}$$

$$\check{x}_t = f(\check{x}_{t-1}, \check{u}_{t-1}, 0) \; \forall t \in [1, T_{a,b}], \tag{5.7}$$

where the trajectory can be stabilized with an on-line state estimate $\hat{x}_t$ as in

$$\check{K}^{a,b} = (K_0, K_1, K_2, \dots, K_{T_{a,b}}), \tag{5.8}$$

$$x_t = f\left(x_t, \check{u}_{t-1} - K_t(\hat{x}_t - \check{x}_t), w_t\right). \tag{5.9}$$

The problem of computing such trajectories and controllers for various robotic vehicles has received an enormous amount of research attention and is beyond the scope of this paper. For general nonlinear systems, techniques such as shooting methods, or direct collocation [57] may be used. For flying vehicles, maneuver primitive approaches are appealing due to their relative computational efficiency [20]. For "Dubins" vehicle dynamics, the optimal trajectory is easily compute in closed form [38]. Stabilizing controllers may be designed, for example, using classical control theory or LQR design, depending on what the system dynamics demand.

The planning problem is specified with some uncertain knowledge of the robot's initial state given by the probability distribution

$$x_0 \sim N(\hat{x}_0, \Sigma_0), \tag{5.10}$$

and a goal region in the environment $x_{\text{goal}} \subset \mathcal{X}^{\text{free}}$ to which the robot wishes to travel.

The optimal path planning problem is then to minimize in expectation a stage cost function,

$$\underset{(\check{X},\check{U},\check{K})}{\operatorname{argmin}} E\left[\sum_{t=1}^{T} J(x_t)\right], \tag{5.11}$$

subject to

$$\check{x}_0 = \hat{x}_0, P(x_T \notin \mathcal{X}_{\text{goal}}) < \delta, \tag{5.12}$$

$$P(x_t \in \mathcal{X}_{\text{obs}}) < \delta, \forall t \in [0, T], \tag{5.13}$$

where $\delta < .5$ is a user specified threshold for how much risk to tolerate that defines the chance-constraint and $J : x \mapsto \mathcal{R}^+$ is the cost function. The expectation is with respect to the process and sensor noise w_t and v_t, and minimization is over concatenated paths returned by the CONNECT() function:

$$(\check{X}, \check{U}, \check{K}) = (\textsc{Connect}(x^0, x^1), \textsc{Connect}(x^1, x^2), \ldots, \textsc{Connect}(x^{l-1}, x^l)).$$

This formulation decouples the control design from the path planning optimization. Practically this makes sense for robots where the stabilizing controller is designed with dynamic considerations rather than the specific configuration of an operating environment.

5.2 Uncertainty Prediction

In order to evaluate a cost function and check the chance-constraint, we need a distribution over states that may be realized if we execute a given nominal trajectory. Taking appropriate partial derivatives of equations 5.1 and 5.2 we obtain the following time-varying linear system

$$\tilde{x}_t = A_t\tilde{x}_{t-1} + B_t\tilde{u}_{t-1} + w_t, \qquad w_t \sim N(0, Q_t) \tag{5.14}$$

$$\tilde{z}_t = C_t\tilde{x}_t + v_t, \qquad v_t \sim N(0, R_t), \tag{5.15}$$

where $\tilde{x}_t$, $\tilde{u}_t$, $\tilde{z}_t$ are now error quantities, representing the deviation from the nominal path such that $x_t = \check{x}_t + \tilde{x}_t$, $u_t = \check{u}_t + \tilde{u}_t$, and $z_t = \check{z}_t + \tilde{z}_t$.

During execution x_t will not be available to compute the control input. Instead we must use an estimate of x_t which we denote as $\hat{x}_t$. The covariance associated with the state estimate is given by Σ_t. To the extent that f and h are locally linear functions, the Kalman filter is the optimal estimator in the sense of minimum expected estimation error. The filter maintains a Gaussian state estimate, $x_t \sim N(\hat{x}_t, \Sigma_t)$ and operates recursively.[1] A process step first predicts the next state and associated covariance,

$$\bar{\tilde{x}}_t = A_t\hat{\tilde{x}}_{t-1} + B_t\tilde{u}_{t-1} \tag{5.16}$$

$$\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + Q_t, \tag{5.17}$$

and a measurement update then adjusts the prediction and incorporates the new information into the covariance,

$$S_t = C_t\bar{\Sigma}_tC_t^T + R_t \tag{5.18}$$

$$L_t = \bar{\Sigma}_tC_t^TS_t^{-1} \tag{5.19}$$

$$\hat{\tilde{x}}_t = \bar{\tilde{x}}_t + L_t(\tilde{z}_t - C_t\bar{\tilde{x}}_t) \tag{5.20}$$

$$\Sigma_t = \bar{\Sigma}_t - L_tC_t\bar{\Sigma}_t. \tag{5.21}$$

where L_t is the Kalman gain. While Σ_t captures the uncertainty that will be present on-line during path execution, it does not represent the full uncertainty from a planning perspective since the mean of the state estimate, $\hat{x}_t$, will not lie on the nominal

[1]We can easily convert between estimates of $\hat{x}_t$ and $\hat{\tilde{x}}_t$ by subtracting or adding the deterministic nominal value of $\check{x}_t$ appropriately.

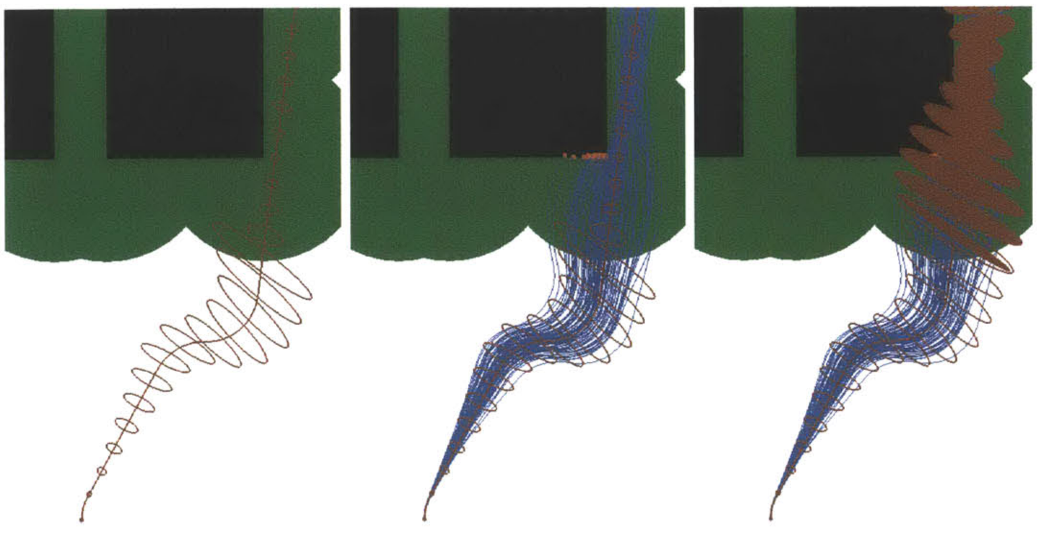

(a) (b) (c)

Figure 5-1: In this example a vehicle with Dubins dynamics (see section 5.5) is trying to traverse past the obstacles. The green areas show the regions where the vehicle can obtain range and bearing measurements from the corners of obstacles. (a) shows predicted covariance ellipses using only the Kalman filter which collapse as soon as measurements are received and then appear to pass safely past the obstacles. However, in (b) we see an ensemble of closed-loop trajectories for this system in blue that clearly do not match the expected Kalman filter distribution after measurements are received since it takes time for the controller to pull the robot back onto the nominal path. In (c) the solid ellipses show the closed-loop distribution which correctly predicts the collision.

trajectory as assumed in [47], [46], [13], [21] and others. Figure 5-1 illustrates visually why this is important for closed loop systems with nontrivial dynamics.

From a planning perspective we need to consider all possible $\hat{x}_t$ that could be realized during path execution. To do this we can use the Kalman filter equations, but treat the observations as random variables (since they have yet to be observed). We will track the distribution over possible state estimates as $P(\hat{\tilde{x}}) \sim N(\mu, \Lambda)$. Taking the appropriate expectations through the prediction step of the Kalman filter (equation

5.17) for the first moment of the distribution follows as:

$$\bar{\mu}_t = E[\bar{\tilde{x}}_t] = E[A_t\hat{\tilde{x}}_{t-1} + B_t\tilde{u}_{t-1}] \tag{5.22}$$

$$= E[A_t\hat{\tilde{x}}_{t-1} - B_tK_t\hat{\tilde{x}}_{t-1}] \tag{5.23}$$

$$= (A_t - B_tK_t)\mu_{t-1}. \tag{5.24}$$

For the update step (equations 5.20, 5.21) we have

$$\mu_t = E(\hat{\tilde{x}}_t) = E(\bar{\hat{\tilde{x}}}_t + L_t(\tilde{z}_t - C_t\bar{\hat{\tilde{x}}}_t)) = \bar{\mu}_t \tag{5.25}$$

$$\mu_t = (A_t - B_tK_t)\mu_{t-1}. \tag{5.26}$$

However, in general $\hat{\tilde{x}}_0 = \mu_0 = 0$ because the planning problem is specified with an initial state estimate from which the noise free trajectory is built. From equation 5.26 it is obvious that if $\mu_0 = 0$ then $\mu_t = 0\ \forall t$. Exploiting the fact that $\mu_t = 0$ and substituting A_K for $A_t - B_tK_t$, the expectations for the second moment follow as

$$\bar{\Lambda}_t = E[(A_K\hat{\tilde{x}}_{t-1})(A_K\hat{\tilde{x}}_{t-1})^T] \tag{5.27}$$

$$= A_K\Lambda_{t-1}A_K^T, \tag{5.28}$$

with the update step as

$$\Lambda_t = E[(\hat{\tilde{x}}_t\hat{\tilde{x}}_t^T)] \tag{5.29}$$

$$= E[(\bar{\hat{\tilde{x}}}_t + L_t(\tilde{z}_t - C_t\bar{\hat{\tilde{x}}}_t))(\bar{\hat{\tilde{x}}}_t + L_t(\tilde{z}_t - C_t\bar{\hat{\tilde{x}}}_t))^T] \tag{5.30}$$

$$= \bar{\Lambda}_t + L_tS_tL_t^T \tag{5.31}$$

$$= \bar{\Lambda}_t + L_tC_t\bar{\Sigma}_t \tag{5.32}$$

$$= (A_t - B_tK_t)\Lambda_{t-1}(A_t - B_tK_t)^T + L_tC_t\bar{\Sigma}_t. \tag{5.33}$$

Equation 5.33 gives an expression for propagating the distribution of possible state estimates that will be realized during execution. The $(A_t - B_tK_t)\Lambda_{t-1}(A_t - B_tK_t)^T$ term is contractive if K_t is a stabilizing controller for our system. The additive second term is equivalent to the uncertainty that is subtracted from the Kalman filter covariance during the measurement update. Intuitively, the uncertainty subtracted from the on-line state estimate when measurements are received, is added to uncertainty of the expected mean of the state estimate.

The distribution returned by the Kalman filter, $P(x_t|z_0, z_1, \ldots, z_t) = N(\hat{x}_t, \Sigma_t)$ can be viewed as $P(x|\hat{x})$ since $\hat{x}_t$ encodes the information from the measurements. Equation 5.33 gives an expression for updating a distribution $P(\hat{x}_t) = N(\check{x}_t, \Lambda_t)$. This provides a natural way to represent the joint belief as $P(x, \hat{x}) = P(x|\hat{x})P(\hat{x})$. Using common Gaussian manipulations we get

$$P(x_t, \hat{x}_t) = N\left(\begin{bmatrix} \check{x}_t \\ \check{x}_t \end{bmatrix}, \begin{bmatrix} \Lambda_t + \Sigma_t & \Lambda_t \\ \Lambda_t & \Lambda_t \end{bmatrix}\right). \tag{5.34}$$

The primary significance of this distribution is the marginal, $P(x_t) = N(\check{x}, \Lambda_t + \Sigma_t)$. For planning purposes this is what we care about. It describes the distribution over trajectories as the sum of the on-line state estimation error, Σ_t, and the uncertainty that arises from not having yet taken observations, Λ_t. This distribution is used to check the chance-constraint (5.13) and evaluate cost (5.11) for candidate paths.

5.3 Rapidly-exploring Random Belief Tree

The Rapidly-exploring Random Belief Tree (RRBT) interleaves graph construction and search over the graph to project a tree into belief space. The algorithm operates on a set of vertices, V, and edges, E, that define a graph in state space. Each vertex $v \in V$ has a state, $v.x$, and a set of associated beliefs nodes $v.N$. Each belief node $n \in v.N$ has a state estimate covariance $n.\Sigma$, a distribution over state estimates $n.\Lambda$, a cost $n.c$, and a parent belief $n.$parent. Belief nodes correspond to a unique path through the graph that could be followed to reach the vertex v, and the member variables of belief nodes ($n.\Sigma$, $n.\Lambda$, and $n.c$) are the properties that result from following that path. Each edge $e \in E$ contains the trajectory and control law to

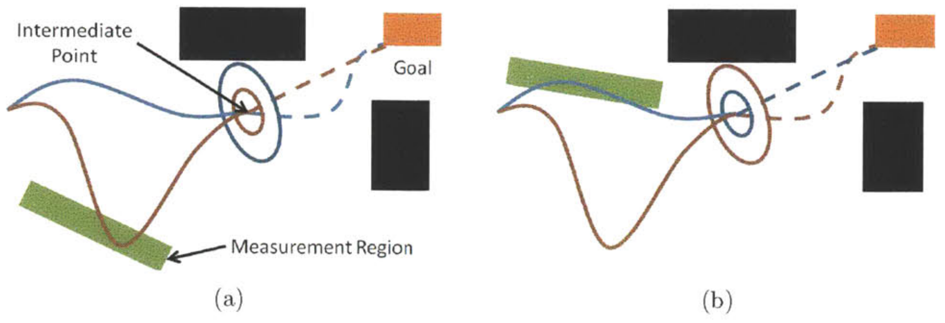


Figure 5-2: This figure shows two different paths that reach an intermediate point with different covariance. In (a) the red path dips down and gets measurements and thus has lower uncertainty at the intermediate point. This lower uncertainty gives it a lower cost to go, shown by the dotted line, since it can traverse closer to obstacles. Thus, at the intermediate point, neither the red or blue paths can be pruned. In contrast, in (b) the blue path receives measurements and thus has lower cost and uncertainty at the intermediate point. In this scenario it dominates the red path, and the red path would be pruned.

traverse between the associated vertices and is defined by the CONNECT() function. A search queue, Q, of belief nodes keeps track of paths that need updating at each iteration of the algorithm.

The covariance prediction (equations 5.21, 5.33), cost expectation evaluation (equation 5.11), and chance-constraint checking (equation 5.13) are implemented by a PROPAGATE(e, n_{start}) function that takes as arguments an edge and a belief node at the starting vertex for that edge, and returns a belief node at the ending vertex for that edge. If the chance-constraint is violated by the uncertainty obtained in propagating the covariances the function returns no belief.

Additionally, we require the following functions: SAMPLE() returns i.i.d. uniform samples from $\mathcal{X}^{\text{free}}$, NEAREST($V, v_{\text{new}}$) takes the current set of vertices as an argument and returns v_{nearest}, the vertex in V that minimizes some distance function to v_{new}, and NEAR(V, v_{new}) returns every vertex within some ball centered at v_{new} of radius $\rho \propto (\log(n)/n)^{(1/d)}$ where n is the number of state vertices and d is the state dimension. For a thorough discussion on the importance of the ball size see [29].

5.3.1 Comparing Partial Paths

At this point we observe that any graph of nominal trajectories through state space implies an infinite set of possible paths through that graph. Search algorithms like Dijkstra's algorithm and A* impose a total ordering on paths to each vertex in the graph based on cost, thus finding a single optimal path to each node. This total ordering is also the property that the RRT* algorithm exploits to "rewire" and maintain a tree in the state space. This works because the optimal cost to the goal from any vertex is not a function of the path taken to that vertex. However, as illustrated in figure 5-2, for our chance-constrained framework, this is generally not the case. As we demonstrate in section 5.4, we can impose a partial ordering of the form:

$$n_a < n_b \Leftrightarrow (n_a.\Sigma < n_b.\Sigma) \wedge (n_a.\Lambda < n_b.\Lambda) \wedge (n_a.c < n_b.c) \tag{5.35}$$

where n_a and n_b represent different partial paths to the same vertex, while being guaranteed not to prune an optimal path.

However, we still have a problem in that infinite loops in the graph may exist within this partial ordering. The covariance update equations make it possible for uncertainty to monotonically decrease while cost must monotonically increase. Physically, this may correspond to a robot circling in an information-rich part of the environment to improve the state estimate. We can rule out this infinite looping by introducing a parameter, ϵ, into the comparison which allows n_a to dominate n_b if the covariances associated with n_a are larger than those associated with n_b by a tolerance factor:

$$n_a \lesssim n_b \Leftrightarrow (n_a.\Sigma < (n_b.\Sigma + \epsilon I)) \wedge (n_a.\Lambda < (n_b.\Lambda + \epsilon I)) \wedge (n_a.c < n_b.c). \tag{5.36}$$

In practice ϵ can be set quite small, and provides a remarkably simple and efficient method for pruning useless paths.

The partially ordered sets of nodes at each vertex are maintained by an AppendBelief(v, n_{new}) function that takes as arguments a state vertex, and a new belief node. This function first checks to see if the new belief is dominated by any existing beliefs at v using equation 5.36. If it is dominated, the function returns failure. If it is not, the function then appends the new node and checks to see if it dominates any existing nodes using equation 5.35, pruning when necessary.

Algorithm 2 RRBT Algorithm

```
1: $n.\Sigma := \Sigma_0$; $n.\Lambda := 0$; $n.c := 0$; $n.\text{parent} := \text{NULL}$;
2: $v.x := x_{\text{init}}$; $v.N := \{n\}$;
3: $V := \{v\}$; $E := \{\}$
4: while $i < M$ do
5:   $x_{\text{rand}} := \text{SAMPLE}()$
6:   $v_{\text{nearest}} := \text{NEAREST}(V, x_{\text{rand}})$
7:   $e_{\text{nearest}} = \text{CONNECT}(v_{\text{nearest}}.x, x_{\text{rand}})$
8:   if $\exists v_{\text{nearest}}.n : \text{PROPAGATE}(e_{\text{nearest}}, n)$ then
9:     $V := V \cup v(x_{\text{rand}})$
10:    $E := E \cup e_{\text{nearest}}$
11:    $E := E \cup \text{CONNECT}(x_{\text{rand}}, v_{\text{nearest}}.x)$
12:    $Q := Q \cup v_{\text{nearest}}.N$
13:    $V_{\text{near}} := \text{NEAR}(V, v_{\text{rand}})$
14:    for all $v_{\text{near}} \in V_{\text{near}}$ do
15:      $E := E \cup \text{CONNECT}(v_{\text{near}}.x, x_{\text{rand}})$
16:      $E := E \cup \text{CONNECT}(x_{\text{rand}}, v_{\text{near}}.x)$
17:      $Q := Q \cup v_{\text{near}}.N$
18:    end for
19:    while $Q \neq \emptyset$ do
20:      $n := \text{POP}(Q)$
21:      for all $v_{\text{neighbor}}$ of $v(n)$ do
22:        $n_{\text{new}} := \text{PROPAGATE}(e_{\text{neighbor}}, n)$
23:        if $\text{APPENDBELIEF}(v_{\text{neighbor}}, n_{\text{new}})$ then
24:          $Q := Q \cup n_{\text{new}}$
25:        end if
26:      end for
27:    end while
28:  end if
29:  $i := i + 1$
30: end while
```

5.3.2 Algorithm Description

Algorithm 2 depicts the RRBT algorithm. The graph is initialized with a single vertex and single belief corresponding to the initial state estimate as specified in equation

5.10 on lines 1-3. This belief will form the root of the belief tree.

At each iteration of the main loop, the state graph is updated by sampling a new state and then adding edges to the nearest and near vertices as in the RRG algorithm. Whenever an existing vertex has an outgoing edge added, all the belief nodes at that vertex are added to the queue. It should be noted that the new vertex is only added to the graph (along with the appropriate edges) if the chance-constraint can be satisfied by propagating an existing belief at the nearest vertex to the new sampled vertex as shown by the check on line 8. This is analogous to "collision-free" checks in a standard RRT.

After all the edges have been added, the queue is exhaustively searched using uniform cost search from lines 19-27, using the the pruning criteria discussed above and implemented by APPENDBELIEF(). The choice of uniform cost search is important because it guarantees that within an iteration of the algorithm (adding a new sample), no new belief will be appended at a state vertex and then pruned. This is a direct consequence of the partial ordering in equation 5.35 including cost, and the fact that with uniform cost search and a positive cost function, the cost of nodes being examined must monotonically increase.

5.4 Convergence Analysis

In this section we show, given some reasonable assumptions about the environment and the system dynamics, that the RRBT algorithm converges to the optimal path in the limit of infinite samples. We begin by stating necessary assumptions.

Assumption 1. *Let* $e_1 =$ CONNECT*(x^a, x^c), $e_2 =$* CONNECT*(x^a, x^b), and* $e^3 =$CONNECT*(x^b, x^c). If $x^b \in e_1$, then the concatenation $[e_2, e_3]$ must be equal to e_1 and as a consequence have equal expected cost for any initial belief.*

This assumption states that the CONNECT() function must be consistent for intermediate points and that the cost function must also be consistent. It further states that our CONNECT() function must correctly and consistently interpolate the

LQG properties. This is necessary since our algorithm relies on refining through infinite sampling which implies that samples will be infinitely close together. Since for most robots the discrete dynamics equations will be derived from a continuous system description, this implies that we must be able to compute a "partial" step by rediscretizing the continuous system with the appropriate time step.

Assumption 2. *There exists a ball of radius $\gamma \in \mathbb{R}_+$ at every point $x \in \mathcal{X}$ such that (i) for all $x' \in \mathcal{X}^\gamma$, $\int_{\mathcal{X}_{obs}} P(x')dx' < \delta$, where $P(x')$ is a reachable belief at x', and (ii) $x \in \mathcal{X}^\gamma$*

This assumption is a stochastic-chance-constrained parallel to assumption 14 in [29]. It states that the obstacles in the environment are spaced such that it is possible to move the mean of a distribution within some ball and not violate the chance-constraint. This is necessary to give the graph a finite sample volume to converge in.

Assumption 3. *The structure of $\mathcal{X}_{obs}$ and is such that if $x^a \sim N(\hat{x}, \Sigma^a)$, $x^b \sim N(\hat{x}, \Sigma^b)$, and $\Sigma_a < \Sigma_b$ then $P(x^a \in \mathcal{X}^{obs}) \leq P(x^b \in \mathcal{X}^{obs})$ for all $\hat{x} \in \mathcal{X}_{free}$*

This assumption simply states that decreasing the covariance can't increase the probability of collision at a given state estimate. This may be violated for very sparse environments with small obstacles and large uncertainty, but is practically very reasonable.

Assumption 4. *The cost function is convex in the sense that if $x^a \sim N(\hat{x}, \Sigma^a)$, $x^b \sim N(\hat{x}, \Sigma^b)$, and $\Sigma_a < \Sigma_b$ then $E[J(x^a)] \leq E[J(x^b)]$ for all $\hat{x} \in \mathcal{X}_{free}$.*

While this is a restrictive assumption, we note that it includes a uniform cost function over the state, resulting in shortest path behavior. Additionally, the environmental obstacles need not be convex since the cost function is decoupled from the obstacle constraints. Further, even if the actual cost function is not globally convex, it may still be locally convex along the optimal path and the above assumption can still be met for Σ^a and Σ^b below a certain threshold.

Assumption 5. *The partial derivatives that lead to equations 5.14 and 5.15 are exact.*

This is the most restrictive assumption. It states that our system must be perfectly locally linear and further, that the LQG properties (R, Q, A, B, and C) must be the same during the planning phase and execution phase. While this is certainly not generally true, for many systems this is a reasonable approximation, and it is justified since we are using a feedback control law to stay close to the nominal trajectory. This is also more realistic than assuming maximum likelihood observations. Instead we are assuming that we can predict the properties of the measurements, without assuming we know the actual values of the measurements.

Lemma 1. *Let $\mathcal{P}^{cc}$ denote the set of all finite length paths through $\mathcal{X}$ such that for every $x_t \in p$ for every $p \in \mathcal{P}^{cc}$ $P(x_t \in \mathcal{X}^{obs}) < \delta$. Let $\mathcal{P}^{cc}_{V_i,E_i}$ denote a similar set contained in the graph of the RRBT algorithm at iteration i.* $\lim_{i\to\infty} \mathcal{P}^{cc}_{V_i,E_i} = \mathcal{P}^{cc}$.

Proof. (Sketch) This follows from assumptions 1 and 2 along with results presented in [29]. The idea is that if the obstacles in the environment are spaced such that there is some reachable belief that will permit a distribution to be shifted within a ball, then in the limit of infinite samples, there will be an infinitely dense connected graph in the ball. Since this property is assumed to hold for all x, the environment will be covered by an infinitely dense connected graph. □

Lemma 1 states that in the limit of infinite samples, the underlying graph built by the RRBT algorithm contains all finite length paths that respect the chance-constraint. We must therefore show that the search tree of beliefs that we maintain on top of this graph contains all possible paths in the graph that *could* be optimal. Our pruning strategy exploits the fact that LQG belief propagation is invariant with respect to inequality in initial beliefs. To demonstrate this, we use the general Binomial Matrix Inversion Lemma,

$$(A+B)^{-1}=A^{-1}-A^{-1}B(B+BA^{-1}B)^{-1}BA^{-1}. \qquad (5.37)$$

We make use of the general Lemma as it relates to positive definite covariance manipulations with the following Lemma.

Lemma 2. *For two covariance matrices A and B, there exists another symmetric positive definite matrix C such that $A^{-1} = (A+B)^{-1} + C$.*

Proof. This follows immediately from equation 5.37 and the observation that if A and B are symmetric-positive-definite, then the quantity $A^{-1}B(B+BA^{-1}B)^{-1}BA^{-1}$ must also be symmetric positive-definite. $\square$

This property extends to the covariance of the Kalman filter with the following Theorem.

Theorem 1. *For two covariance matrices, Σ_0^1 and Σ_0^2, where there exists some positive-definite matrix D_0 such that $\Sigma_{x_0}^1 + D_0 = \Sigma_{x_0}^2$, there will always be another positive definite matrix D_t such that $\Sigma_{x_t}^1 + D_t = \Sigma_{x_t}^2 \forall t \in [0, \infty)$.*

Proof. We begin by noting that the Kalman filter relies upon a two step recursion. Thus if the property holds through each step of the recursion it holds for all $t < \infty$.

For the process step we have

$$\begin{aligned} \bar{\Sigma}_t^2 &= A\Sigma_{t-1}^2 A^T + Q = A(\Sigma_{t-1}^1 + D)A^T + Q \\ &= A\Sigma_{t-1}^1 A^T + ADA^T + Q \\ \bar{\Sigma}_t^1 &= A\Sigma_{t-1}^1 A^T + Q \\ \bar{\Sigma}_t^2 - \bar{\Sigma}_t^1 &= ADA^T = D'. \end{aligned}$$

For the measurement update we turn to the information form of the Kalman update,

$$\Sigma_t^2 = (\bar{\Sigma}_t^{2^{-1}} + R^1)^{-1} = ((\bar{\Sigma}_t^1 + D')^{-1} + R^1)^{-1}.$$

by Lemma 2 we can write

$$\Sigma_{x_t}^2 = (\bar{\Sigma}_{x_t}^{1^{-1}} - D'' + R^1)^{-1},$$

and again

$$\Sigma_{x_t}^2 = (\Sigma_{x_t}^{1^{-1}} + R^1)^{-1} + D'''.$$

Thus we have $\Sigma_{x_t}^2 = \Sigma_{x_t}^1 + D'''$ where D''' is positive-definite. □

The following Theorem states that a similar property holds for the mean uncertainty.

Theorem 2. *For two state covariance matrices, Σ_0^1 and Σ_0^2, where there exists some positive-definite matrix D_0 such that $\Sigma_t^1 + D_0 = \Sigma_t^2$, and two corresponding mean covariance matrices, Λ_0^1 and Λ_0^2, with the same property $\Lambda_0^1 + E_0 = \Lambda_0^2$, there will be some positive-definite matrix E_t such that $\Lambda_t^1 + E_t = \Lambda_t^2$ always holds.*

The proof follows in a similar manner to Theorem 1, by plugging into the belief propagation equations and applying Lemma 2.

Theorem 3. *For two beliefs n_a and n_b at the same state, let p_a and p_b be the nominal trajectories to the beliefs, and let p_a^g and p_a^g be the optimal nominal trajectories from n_a and n_b to the goal. If $n_a \lesssim n_b$ then $E\left[\sum_{p_a^g} J(x_t)\right] + n_a.c \leq E\left[\sum_{p_b^g} J(x_t)\right] + n_b.c + c_\epsilon$, and $\lim_{\epsilon \to 0} c_\epsilon = 0$.*

Proof. This result follows directly from assumptions 3 and 4 combined with Theorems 1 and 2. If $n_a \lesssim n_b$, then the optimal cost to go for n_a must be within some constant factor of that for n_b. Since the limit yields $n_a < n_b$ the constant factor has to approach 0. The accumulated cost is strictly less than that of n_b and thus the total cost is also strictly less than that for n_b. □

Theorems 1 through 3 prove that the pruning strategy is conservative in that it only removes suboptimal paths; since by Theorem 1 the original graph contains all paths achievable with the CONNECT() function, in the limit of infinite samples and $\epsilon \to 0$, the belief tree will contain the optimal path.

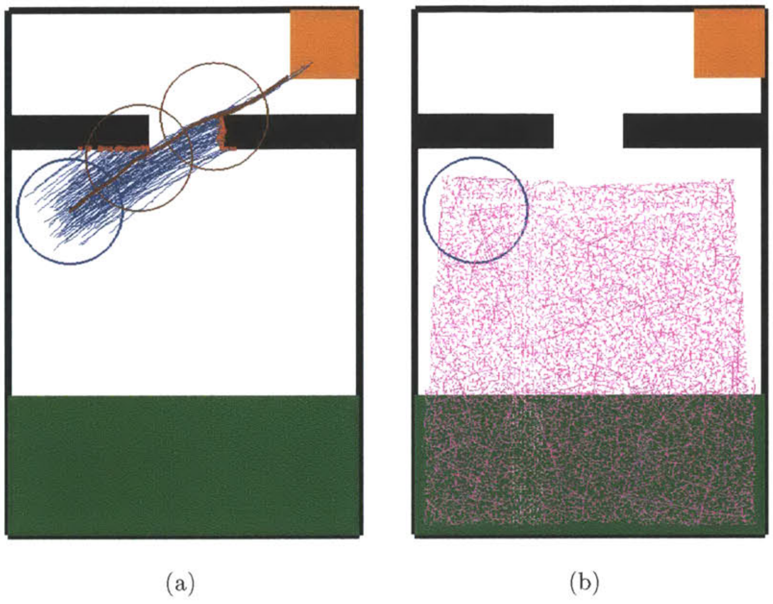


Figure 5-3: In this environment the robot can only receive measurements in the bottom green region. Ignoring uncertainty and moving straight to the orange goal region (a) results in a high probability of collision, while growing an RRT while checking the chance-constraint (b) will not find a solution. The purple lines in (b) depict nominal paths through the environment none of which reach the upper region since any sample drawn from the that region will be connected to a path that hasn't visited the green region, and thus cannot safely pass the obstacles.

5.5 Experimental Results

We first implemented the algorithm on a 2D system with dynamics:

$$x_t = x_{t-1} + u_{t-1} + w_t), \qquad w_t \sim N(0, 0.01I) \tag{5.38}$$

$$z_t = x_t + v'_t, \qquad v_t \sim N(0, R), \tag{5.39}$$

where $R = \infty I$ or $R = 0.01I$ depending on the location in the environment. Figure 5-3 shows a specific configuration of an environment that forces trade-offs between information gathering and finding short paths, where the robot can only receive measurements in a small region away from the goal. In this scenario, moving straight to the goal will not satisfy the chance-constraint.

To evaluate the probability of collision on each step, we used the conservative

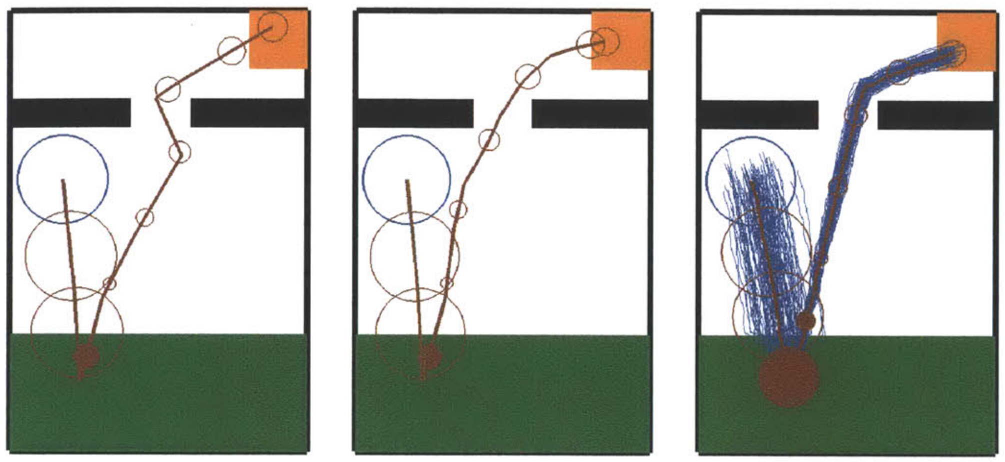

(a) (b) (c)

Figure 5-4: (a) shows the RRBT algorithm after 100 iterations, (b) after 500 iterations, and (c) after 10000 iterations. The algorithm quickly finds a feasible solution that goes down to the information region to localize and then pass safely between the obstacles. As more samples are added, this path is refined. Note that the solution is slightly conservative in that the path goes far enough into the green region to ensure the probability mass within the chance-constraint actually receives measurements.

approximation of checking the ellipse defined by the covariance matrix and a desired chance bound for collisions with obstacles. This is computationally faster than integrating the distribution over $\mathcal{X}^{\text{obs}}$, and since the problem formulation states that the specification is an upper bound, this is a reasonable approximation to make. For more aggressive (but still conservative) approaches see [44].

For the example in figure 5-3, simply growing an RRT and checking the chance-constraint along the paths, as proposed in [6], fails to find a feasible solution since the Voronoi-bias will prevent expansion of the paths that have passed through the measurement region. In contrast, the RRBT algorithm, not only find a feasible path, it refines towards the optimal path as shown in figure 5-4. The cost and runtime statistics averaged over 20 runs in this environment are shown in figure 5-5. As our theoretical predict, we can see the cost converging as a function of the number of samples. Additionally, the computational complexity per iteration is sub-linear.

It is important to note that the algorithm is dependent on being able to predict the properties of the measurements that will be received. Since, for the problems we

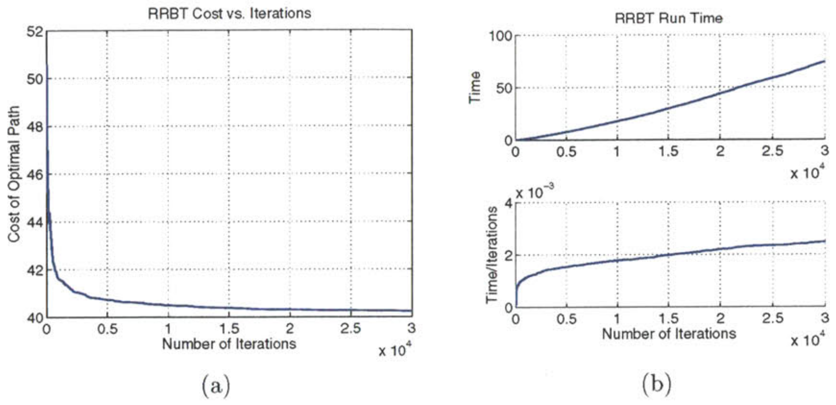


Figure 5-5: This figure shows the cost of the best path in the tree (a), the run time (b - top), and the run time per iteration (b - bottom) as a function of the number of samples for the environment in figure 5-4, averaged over 20 runs.

are interested in, the measurements are a function of state, the actual measurement properties may vary from the predicted covariance. In our implementation we handle this by only predicting a measurement if only probability mass below the chance-constrained level is outside of a measurement region. This can be seen in figure 5-4 where the solution goes far enough into the measurement region to ensure that every state element inside the covariance ellipse receives measurements.

In addition to the 2D system we also tested the algorithm in a domain with Dubins vehicle dynamics and range and bearing beacon measurements from the corners of obstacles which serves as an approximate model for a fixed-wing vehicle that uses a corner detector with a LIDAR. The continuous Dubins vehicle dynamics are described by:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} V\cos(\theta) \\ V\sin(\theta) \\ \omega \end{bmatrix}, \tag{5.40}$$

where,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \theta \end{bmatrix}, u = \begin{bmatrix} V \\ \omega \end{bmatrix}. \tag{5.41}$$

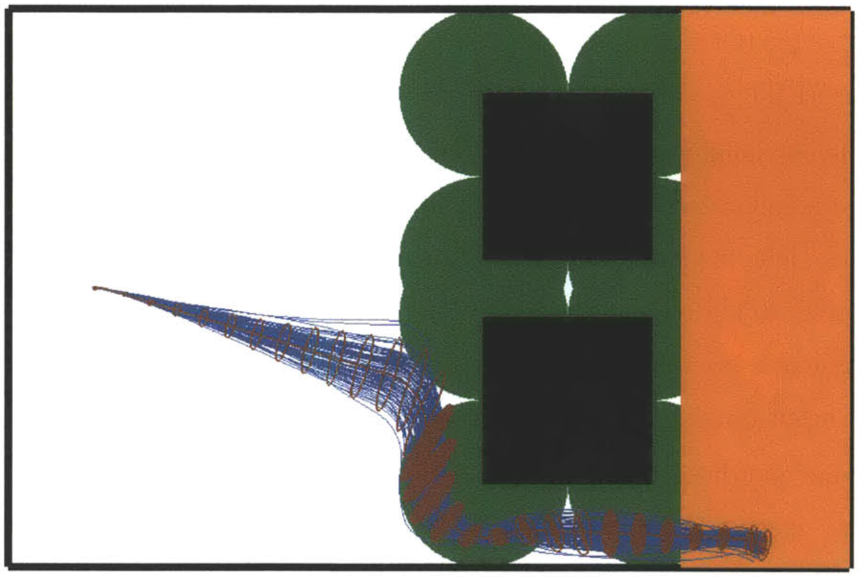


Figure 5-6: This figure shows the algorithm running in a more complicated environment with Dubins dynamics and beacon measurements. We see that the algorithm finds a path that turns parallel to the obstacles so as to localize and stabilize onto the nominal trajectory before going past the obstacles to the goal.

We refer the reader to [54] for details on discretizing and implementing this model, and for the specifics of the measurement model. A path returned by the RRBT algorithm for a sample environment is shown in figure 6. This is a challenging environment where uncertainty accumulates as the vehicle heads towards obstacles. When the vehicle reaches the obstacles it receives measurements and uncertainty collapses. However, as shown in figure 5-1, proceeding directly past the obstacles is not possible since it takes time for the actual path to stabilize down onto the nominal path. Instead, the algorithm returns a solution that turns parallel to the obstacles, gets measurements, stabilizes onto the nominal path, and then safely goes to the goal.

5.5.1 Fixed-wing Vehicle Simulation

We also tested the RRBT algorithm in simulation using the trajectory generation methodology described in chapter 3 as the CONNECT() function and the state estimation approach described in chapter 4 together with the closed loop dynamics in the PROPAGATE() function. In this section we provide the technical details necessary to apply the RRBT to the fixed-wing system.

While the polynomial optimization minimizes roll input, it does not directly guarantee satisfying roll rate constraints. This guarantee could be obtained by introducing control points along the path and enforcing inequality constraints, however, satisfying such constraints would require an iterative QP solution instead of the direct matrix inversion approach for equality constraints. Instead we simply discard trajectories that violate input constraints as if a collision had occurred.

We assume coordinated flight along paths which exist in the XY plane. This leaves a state space $x \in \mathbb{R}^5$ comprised of XY position, heading, roll, and roll rate: $x = \begin{bmatrix} \Delta_1 & \Delta_2 & \psi & \phi & \dot{\phi} \end{bmatrix}$. While the transverse-polynomial paths would allow us to sample in this space and then make exact connections, such a strategy would be inefficient, as we would sample states with opposing roll, roll rate, and heading such that the connected path had either immediately infeasible or awkward approaches. Instead we sample in the reduced space of planar position and heading, SE2, represented as $\mathbb{R}^3$, and leave the end boundary condition floating in the polynomial optimization. This sampling strategy results in smoother paths and fewer discarded samples.

To query collisions we use the same octomap used in GPF localization [58]. In the CONNECT() function this is done by querying cells the trajectory passes through. In the PROPAGATE() function this is done (conservatively) by ray-tracing from the mean of the distribution to the corners of an XYZ aligned box defined by the diagonal elements of the covariance matrix scaled by the desired risk factor - typically 2σ.

The propagate function must implement equations 5.33 along with 5.17. However, we have a slight difficulty in that the states and inputs used in the filter and controller are different. The control law presented in chapter 3 has position, Δ, velocity in the global frame, $\dot{\Delta}$, normal acceleration in the body frame, a_{v1}, and the path parameter derivatives $s, \dot{s}, \ddot{s}$, as states, with the derivative of tangential acceleration, $\dot{a}_{v1}$, roll rate, $\dot{\phi}$, and the third derivative of the path parameter $\dddot{s}$, as inputs. The state estimator has velocity in the body frame v_b, position, Δ, and orientation, R, as states. One possibility would be to augment the state for planning purposes by taking the union of the state variables for each, and augment both the estimator dynamics and the mean (Λ) dynamics equations with the appropriate jacobians.

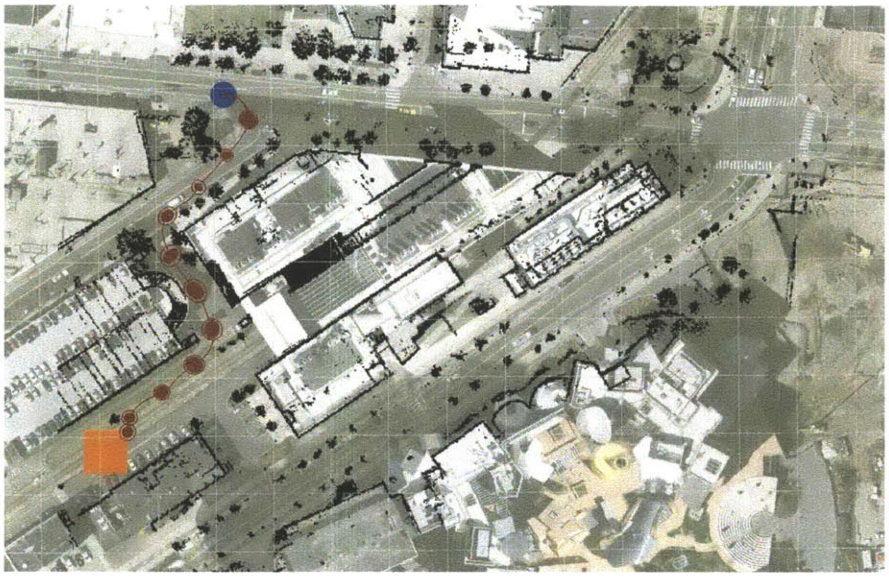
However, the differentially flat control law has the powerful property that the error dynamics are third-order on position, and linear. Thus we can express the Λ dynamics in terms of a triple integrator while using the EKF and GPF equations exactly as described in chapter 4. One possibility for computing the GPF measurement updates in the propagate function would be to simulate laser measurements and then run them through the GPF pipeline just as is done on-line. However, while the GPF is efficient, simulating laser measurements and then sampling and weighting particles is not fast enough to run in the planning inner loop. Instead, we discretize the state space in roll, heading, x and y, and compute a measurement function map of position covariance matrices using the GPF in an off-line step, which is then used during planning.

Using this framework we computed motion plans in an octomap built with data from the MIT Darpa Urban Challenge vehicle driving through Cambridge around the Stata Center on MIT's campus [39]. The octomap and associated motion plans for both the RRBT and LQG-MP algorithms are shown in figure 5-7 and 5-9 respectively. We can see that all the paths are appropriately cautious based on the uncertainty around obstacles. Additionally, we can see that the family of paths that passes most directly to the goal (of which the optimal path is a member) is highly constrained by both the dynamic turning radius and the uncertainty. In multiple trials, the RRBT does not always find this path within 50000 iterations, however, the LQG-MP algorithm shown in figure 5-9 rarely finds it.

Each algorithm was run in the same environment at the same start and end locations for 50000 iterations in 20 different trials. Plots showing the fraction of feasible paths obtained as a function of both the iteration number and run time are shown in figure 5-10. In figure 5-11 we show the cost and number of partial paths (belief nodes) as a function of iteration number and number of state vertices respectively. As we would expect, the RRBT outperforms the RRT in terms of cost.

The reason for this is that the RRT (LQG-MP) algorithm can't optimize for either cost or uncertainty. Once a path to a given sample is established, it will never be replaced. By contrast, the RRBT algorithm continuously replaces paths through search and thus can reduce cost and uncertainty to all points in state space as more samples are added. In this environment, that translates to finding the most direct, but also more aggressive, path to the goal more often.

For the RRT the number of state vertices is equivalent to the number of belief nodes since the underlying structure is a tree. The variance in the RRT across runs comes from the number of successful samples. The more interesting feature is the fact that the number of partial paths for the RRBT scales linearly with the number of vertices, even for the very intricate empirical belief space caused by the laser measurement function as compared to simpler measurement functions shown in previous examples. This would support the hypothesis that the set of dominance relationships setup in the belief space remains constant even as the space is more densely populated. Exploring this property analytically is a potentially interesting piece of future work.


(a)

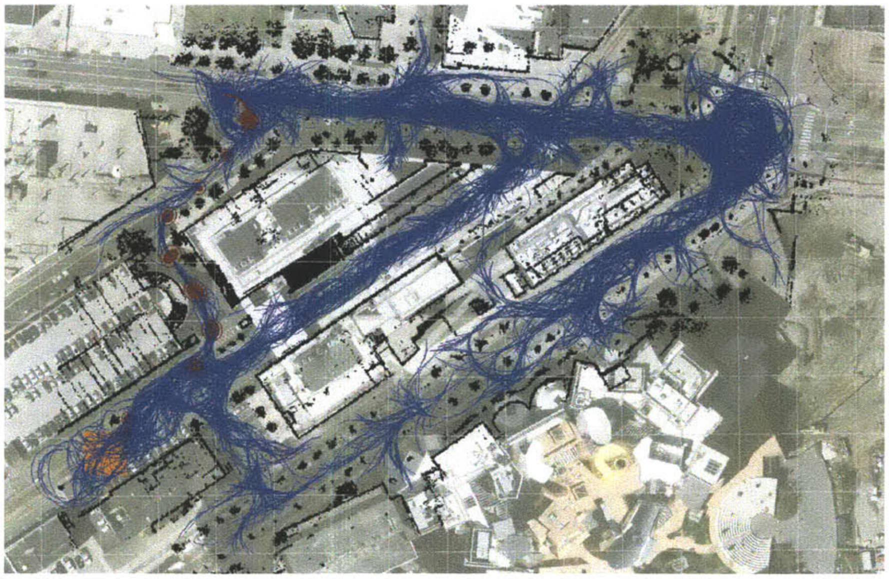

(b)

Figure 5-7: Example motion plans through the urban canyon computed using the RRBT. The octomap is overlaid on aerial imagery in black. The planning takes place at 3 meters in height, and the map is cropped between 2 and 4 meters for clarity in visualization. The vehicle starts in upper left corner with uncertainty given by the blue circle. In this run the RRBT converges to the path shown in maroon (a). In (b) we see the family of partial paths generated by the algorithm.

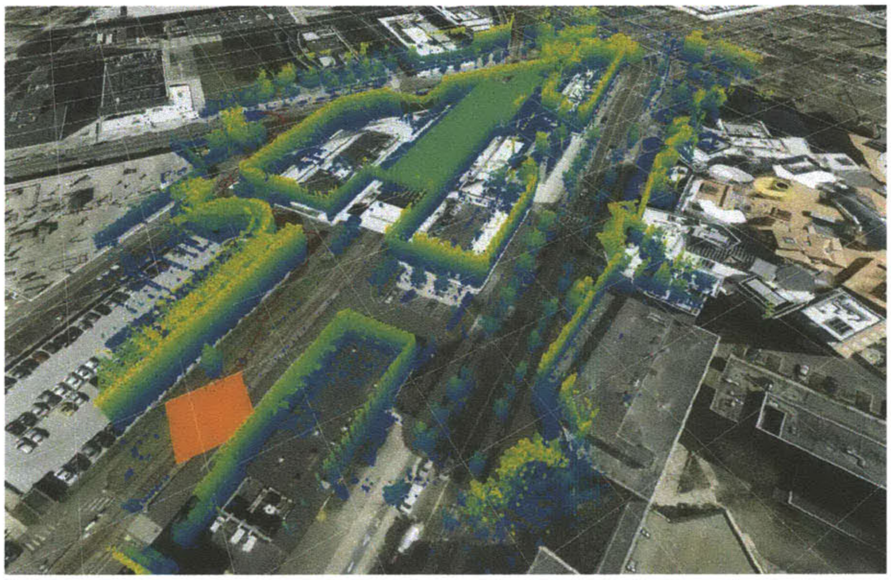
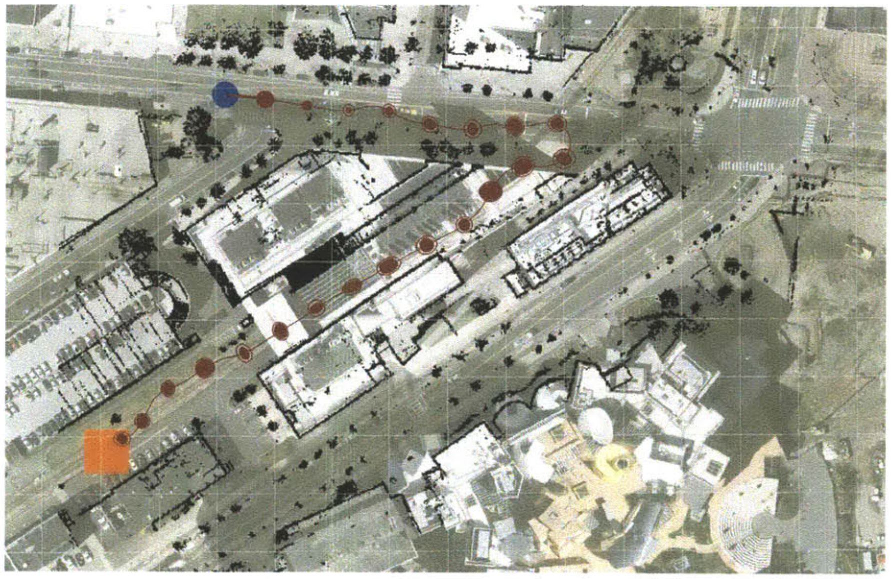


Figure 5-8: A 3D view of the environment with the octomap colored according to height. We can see the winding path the vehicle takes through the urban canyon.


(a)

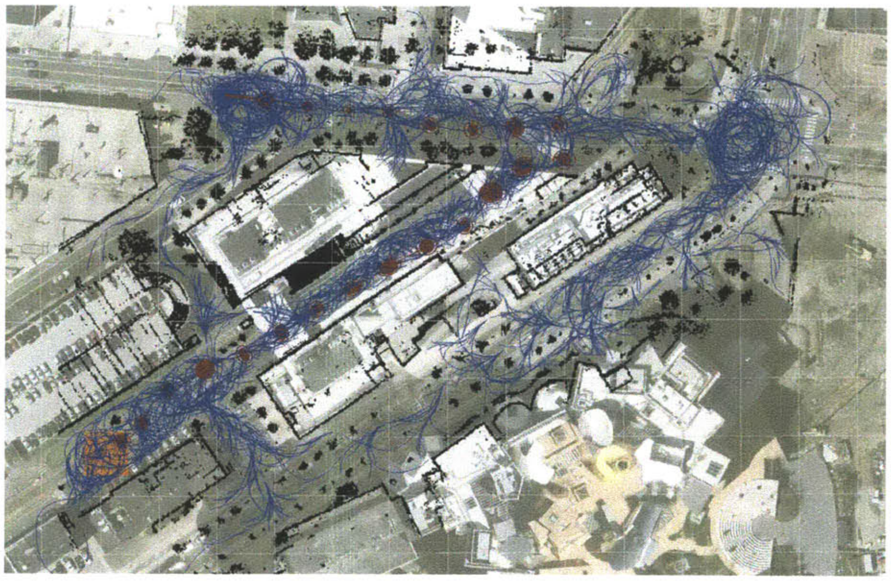

(b)

Figure 5-9: Example motion plans through the urban canyon computed using the RRT while propagating uncertainty (LQG-MP). The path the algorithm finds passes through a wide tunnel rather than the more aggressive winding directly to the goal. The reason for this is the RRT gets "blocked" by an initial solution that takes the longer route and can't rewire when it eventually does discover the alternative shorter path.

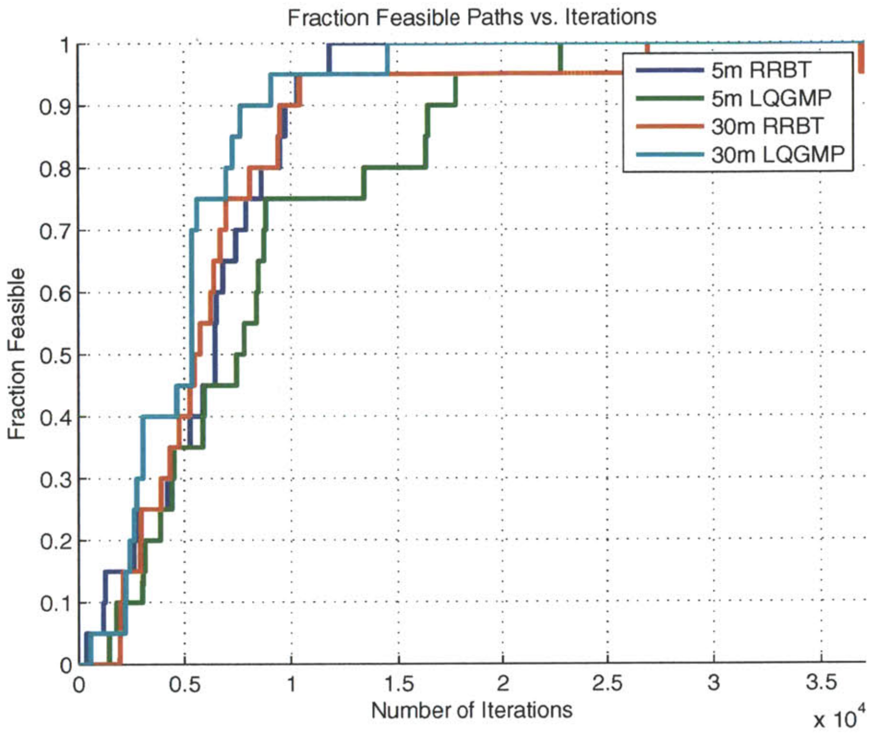
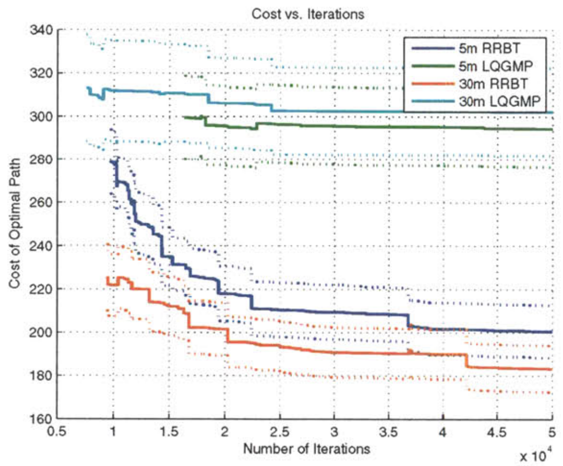
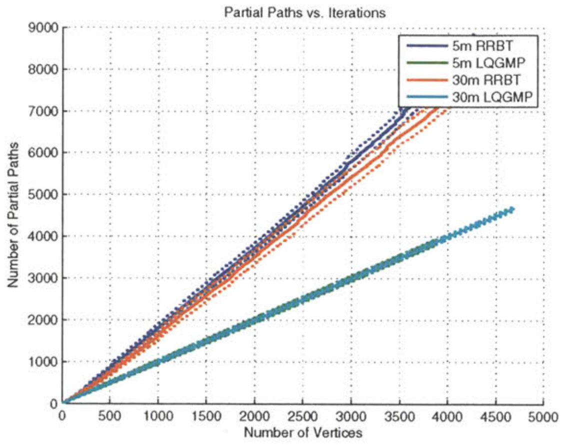


Figure 5-10: Feasible solutions vs. iteration number for RRBT and LQG-MP algorithms with different laser max ranges. This plot shows the fraction of feasible paths obtained (out of 20 runs) versus iteration number. The LQG-MP algorithm with 30 meter laser range generally finds paths the fastest, but LQG-MP with only 5 meter laser range finds paths the slowest. This makes sense since as the observability properties of the problem get harder, information gathering and information use become more important. By contrast, the RRBT algorithm displays fairly even performance between the two cases, except for a single trial taking a large number of iterations in the 30 meter case.



(a)



(b)

Figure 5-11: In (a) we see cost vs. iteration number for both laser ranges and both algorithms. The dotted lines are 1 standard-error over 20 runs, and the statistics are computed ignoring runs with infeasible solutions, but only after 90% of the runs have a feasible solution. RRBT outperforms RRT, however, there is still large variance in the solution cost up to this number of iterations due to the difficulty of finding the shortest path winding through the obstacles. In (b) we see the number of partial paths (belief nodes) for each algorithm versus the number of state nodes.

Chapter 6

Future Work and Conclusions

The autonomous flight of a fixed-wing vehicle through obstacles using only on-board sensing is a challenging algorithmic problem. In this thesis we have introduced algorithms to address the specific problems of trajectory generation, state estimation, and planning under uncertainty. The state estimation algorithm was validated and tested on real flight data, obtained from the vehicle platform, while the trajectory generation and planning algorithms were validated in simulation.

The primary piece of remaining work is the experimental demonstration of the physical system operating in a real environment. Most of the work to make this possible lies in the details of hardware implementation and testing and tuning the parameters of the trajectory generation and control system based on closed-loop flight tests.

Each of the individual algorithmic pieces also present interesting opportunities for future work. Our trajectory generation algorithm for transverse-polynomial paths could be extended to truly 3D flight by offsetting a polynomial, not only in the horizontal transverse direction, but also in the vertical transverse direction. This would allow 3D segments with 2-axis discontinuity at their junctions to be stitched together smoothly, while still minimizing the control effort follow the paths. The method could be further refined and validated with flights in a motion capture system (similar to the systems that have allowed leaps forward in helicopter flight [14], [41]), however precise motion capture systems with large enough flight volumes for fixed-wing vehicles performing extended maneuvers are not readily available.

In chapter 4 we presented the state estimation which is based on an IMU and laser range scanner. Our algorithm uses a novel extension of the Gaussian particle filter and an exponential coordinates linearization of the IMU dynamics equations. We have demonstrated the performance of our algorithms on two challenging datasets. The quantitative analysis in motion capture clearly shows the advantages of our extensions to the Gaussian particle filter algorithm, while the accurate map generated during the flight tests demonstrate the absolute accuracy of our algorithms.

In addition to the planning and control extensions, investigation of other sensing modalities such as vision are of great interest. We believe that the filtering framework developed for the laser range finder will extend to incorporate additional measurement types, thereby further improving the capabilities of our system.

From a planning perspective, robots with continuous dynamics that operate in partially observable, stochastic domains, motion planning presents significant challenges. In this thesis we present an algorithm, the Rapidly-exploring Random Belief Tree, that leverages a local LQG control solution to predict a distribution over trajectories for candidate nominal paths, and then uses incremental sampling refinement to optimize over the space of nominal trajectories. While we have demonstrated the utility of the algorithm for simulation examples, future work remains.

Further theoretical work is necessary in investigating the computational complexity. Our experimental results suggest that the complexity is sub-linear per iteration, but more analysis is necessary to confirm this. A key question is how the number of belief nodes scales relative to the number of state vertices.

While our simulations show that plans for the actual vehicle can be computed in reasonable time periods, more work remains to make the algorithm truly real-time. An A* heuristic could be used in the search to focus towards the goal. Once the goal is found, the same heuristic could be used to bound the tree and graph growth and eliminate suboptimal regions. Further, by introducing an "expected value of information" it would be possible to reduce the number of belief nodes that must be maintained at each state vertex, by narrowing in on the multi-objective front of uncertainty and cost to the most useful regions.

Perhaps most interestingly, the RRBT algorithm could be generalized to include other systems that obey the basic dominance property with the conjunction of inequalities of path statistics. This could potentially include planning with more sophisticated motion constraints, such as homotopy constraints, or specific adaptations of the algorithm for systems with stochastic dynamics that are fully observable or partially observable systems with fully actuated dynamics.

Bibliography

[1] CRRCsim, Open Source Model Flight Simulator.

[2] P. Abbeel, A. Coates, M. Montemerlo, A. Y. Ng, and S. Thrun. Discriminative training of Kalman filters. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[3] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. Stereo vision and laser odometry for autonomous helicopters in GPS-denied indoor environments. In *SPIE Unmanned Systems Tech. XI*, 2009.

[4] A. Bachrach, R. He, and N. Roy. Autonomous flight in unknown indoor environments. In *Inter. Jour. Micro Air Vehicles*, 2009.

[5] A. Bachrach, S. Prentice, R. He, and N. Roy. RANGE: Robust autonomous navigation in GPS-denied environments. *Journal of Field Robotics*, 28(5):644–666, 2011.

[6] J. V. D. Berg, P. Abbeel, and K. Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. In *Proc. RSS*, 2010.

[7] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1999.

[8] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 2nd edition, 2000.

[9] Antoine Beyeler, Jean-Christophe Zufferey, and Dario Floreano. Vision-based control of near-obstacle flight. *Autonomous Robots*, 27(3):201–219, 2009.

[10] A. Bry and N. Roy. Exact belief state computation for piecewise LQG planning. Technical report, Massachusetts Institute of Technology, 2010.

[11] A. Bry and N. Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *IEEE Int. Conf. Robotics and Automation*, May 2011.

[12] G. Buskey, J. Roberts, P. Corke, and G. Wyeth. Helicopter automation using a low-cost sensing system. *Computing Control Engineering Journal*, 15(2):8 – 9, april-may 2004.

[13] Andrea Censi, Daniele Calisi, Alessandro De Luca, and Giuseppe Oriolo. A Bayesian framework for optimal motion planning with uncertainty. In *ICRA*, Pasadena, CA, May 2008.

[14] A. Coates, P. Abbeel, and A. Ng. Learning for control from multiple demonstrations. In *Proc. ICML*, 2008.

[15] Joseph Conroy, Gregory Gremillion, Badri Ranganathan, and J Humbert. Implementation of wide-field integration of optic flow for autonomous quadrotor navigation. *Autonomous Robots*, 27(3):189–198, 2009.

[16] R. Cory and R. Tedrake. Experiments in fixed wing UAV perching. In *Proc AIAA Guidance, Navigation, and Control Conference*, 2008.

[17] Mark Drela and Harold Youngren. AVL, Extended Vortex-Lattice Model.

[18] Ivan Dryanovski, William Morris, and Jizhong Xiao. An open-source pose estimation system for micro-air vehicles. In *ICRA*, pages 4449–4454, 2011.

[19] R. T. Farouki and T. Sakkalis. Pythagorean hodographs. *IBM Journal of Research and Development*, 34(5):736 –752, sept. 1990.

[20] E. Frazzoli. *Robust Hybrid Control for Autonomous Vehicle Motion Planning.* Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, June 2001.

[21] Juan P. Gonzalez and Anthony Stentz. Using linear landmarks for path planning with uncertainty in outdoor environments. In *Proc IROS*, pages 1203–1210, Piscataway, NJ, USA, 2009. IEEE Press.

[22] S. Grzonka, G. Grisetti, and W. Burgard. Towards a navigation system for autonomous indoor flying. In *IEEE Int. Conf. Robotics and Automation*, pages 2878–2883, May 2009.

[23] John Hauser and Rick Hindman. Aggressive flight maneuvers. In *Conference on Decision and Control*, December 1997.

[24] R. He, S. Prentice, and N. Roy. Planning in information space for a quadrotor helicopter in a GPS-denied environment. In *ICRA*, 2008.

[25] R. He and N. Roy. Efficient POMDP forward search by predicting the posterior belief distribution. Technical report, Massachusetts Institute of Technology, September 2009.

[26] J.A. Hesch, F.M. Mirzaei, G.L. Mariottini, and S.I. Roumeliotis. A laser-aided inertial navigation system L-INS for human localization in unknown indoor environments. In *IEEE Int. Conf. Robotics and Automation*, pages 5376–5382. IEEE.

[27] Stefan Hrabar and Gaurav Sukhatme. Vision-based navigation through urban canyons. *Journal of Field Robotics*, 26(5):431–452, 2009.

[28] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[29] S. Karaman and E. Frazzoli. Incremental sampling-based optimal motion planning. In *Robotics: Science and Systems*, 2010.

[30] J. Kelly, S. Saripalli, and G. Sukhatme. Combined visual and inertial navigation for an unmanned aerial vehicle. pages 255–264. 2008.

[31] J. Kim and S. Sukkarieh. SLAM aided GPS/INS navigation in GPS denied and unknown environments. In *The 2004 International Symposium on GNSS/GPS, Sydney*, pages 6–8, 2004.

[32] D.B. Kingston and R.W. Beard. Real-time attitude and position estimation for small uavs using low-cost sensors. *American Institute of Aeronautics and Astronautics*, 2000.

[33] Derek Kingston, Randal Beard, Al Beard, Timothy McLain, Michael Larsen, and Wei Ren. Autonomous vehicle technologies for small fixed wing uavs. In *AIAA Journal of Aerospace Computing, Information, and Communication*, pages 2003–6559, 2003.

[34] S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning A*. *Artificial Intelligence*, 155(1-2), 2004.

[35] J. H. Kotecha and P. M. Djuric. Gaussian particle filtering. *IEEE Trans. Signal Processing*, 51(10):2592–2601, October 2003.

[36] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J.P. How. Motion planning in complex environments using closed-loop prediction. In *AIAA Conf. on Guidance, Navigation, and Control*, Honolulu, HI, 2008.

[37] S. LaValle and J. Kuffner. Randomized kinodynamic planning. *IJRR*, 20(3), 2001.

[38] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[39] J. Leonard, J. How, S. Teller, M. Berger, S., G. Fiore, L. Fletcher, E. Frazzoli, A. S. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, Ma. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams. A perception driven autonomous vehicle. *Journal of Field Robotics*, 25(10):727–774, Oct. 2008.

[40] R. Mehra. On the identification of variances and adaptive Kalman filtering. *IEEE Trans. Automatic Control*, 15(2):175 – 184, apr 1970.

[41] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE Int. Conf. Robotics and Automation*, May 2011.

[42] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Int. Symposium on Experimental Robotics*, 2010.

[43] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Proceedings of the International Symposium on Experimental Robotics*, Dec 2010.

[44] M. Ono and B. Williams. An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure. In *AAAI*, 2008.

[45] Warren F. Phillips. *Mechanics of Flight.* John Wiley and Sons, Hoboken, NJ, 2004.

[46] 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.

[47] S. Prentice and N. Roy. The belief roadmap: Efficient planning in linear POMDPs by factoring the covariance. In *Proc. ISRR*, 2007.

[48] N. Roy and S. Thrun. Coastal navigation with mobile robots. In *Proc NIPS*, 1999.

[49] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma. Flying fast and low among obstacles: Methodology and experiments. *Int. Journal of Robotics Research*, 27(5):549–574, May 2008.

[50] Sebastian Scherer, Sanjiv Singh, Lyle J. Chamberlain, and Mike Elgersma. Flying fast and low among obstacles: Methodology and experiments. *The International Journal of Robotics Research*, 27(1):549–574, May 2008.

[51] S. Shen, N. Michael, and V. Kumar. Autonomous multi-floor indoor navigation with a computationally constrained mav. In *IEEE Int. Conf. Robotics and Automation*, pages 20–25. IEEE.

[52] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous multi-floor indoor navigation with a computationally constrained mav. In *ICRA*, pages 20–25, 2011.

[53] S. Thrun, D. Fox, W. Burgard, and F.Dellaert. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence Journal*, 2001.

[54] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[55] Antonios Tsourdos, Brian A. White, and Madhavan Shanmugavel. *Cooperative Path Planning of Unmanned Aerial Vehicles.* John Wiley and Sons, 2010.

[56] R. van der Merwe and E. Wan. Sigma-Point Kalman Filters for Integrated Navigation. In *Proc. Institute of Navigation (ION)*, Dayton, OH, June 2004.

[57] O. von Stryk. Numerical solution of optimal control problems by direct collocation. In *Optimal Control*, pages 129–143, 1993.

[58] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010.

[59] Jean-Christophe Zufferey, Antoine Beyeler, and Dario Floreano. Near-obstacle flight with small UAVs. In *UAV'2008*, New York, 2008. Springer Verlag.