# *Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments*

In the International Journal of Robotics Research (IJRR), 37(7):969-1002, June 2015.

# Aggressive Flight of Fixed-Wing and Quadrotor Aircraft in Dense Indoor Environments

Adam Bry*, Charles Richter*, Abraham Bachrach and Nicholas Roy

**Abstract**

In this paper, we describe trajectory planning and state estimation algorithms for aggressive flight of micro aerial vehicles in known, obstacle-dense environments. Finding aggressive but dynamically feasible and collision-free trajectories in cluttered environments requires trajectory optimization and state estimation in the full state space of the vehicle, which is usually computationally infeasible on realistic time scales for real vehicles and sensors. We first build on previous work of van Nieuwstadt and Murray [51] and Mellinger and Kumar [38], to show how a search process can be coupled with optimization in the output space of a differentially flat vehicle model to find aggressive trajectories that utilize the full maneuvering capabilities of a quadrotor. We further extend this work to vehicles with complex, Dubins-type dynamics and present a novel trajectory representation called a Dubins-Polynomial trajectory, which allows us to optimize trajectories for fixed-wing vehicles. To provide accurate state estimation for aggressive flight, we show how the Gaussian particle filter can be extended to allow laser range finder localization to be combined with a Kalman filter. This formulation allows similar estimation accuracy to particle filtering in the full vehicle state but with an order of magnitude more efficiency. We conclude with experiments demonstrating the execution of quadrotor and fixed-wing trajectories in cluttered environments. We show results of aggressive flight at speeds of up to 8 m/s for the quadrotor and 11 m/s for the fixed-wing aircraft.

## 1 Introduction

Micro aerial vehicles (MAVs) have received considerable attention during the last decade due to their large and growing range of applications, including deployment as inexpensive mobile sensor platforms for monitoring, reconnaissance and mapping operations. Hence, most MAVs are limited to flight at altitudes well above ground-level obstacles, foliage and buildings, where basic path planning techniques and GPS are sufficient for navigation. The practical uses of MAVs would expand considerably if they were capable of flying at high speeds and among obstacles at low altitudes, enabling them to take detailed measurements of the street level of a city or the terrain in wilderness areas and cover significantly more ground than the conservative motions of many current aerial robots. Our purpose in this research is to enable quadrotor and fixed-wing vehicles to perform agile flights in obstacle-dense environments for these applications.

*These authors contributed equally to this work.

The two major algorithmic challenges that must be solved to enable aggressive flight are trajectory planning and state estimation. These challenges are especially difficult due to the limited computational resources that can be carried aboard a small MAV. Though sensing and computational hardware have advanced considerably, MAVs are still very restricted by their limited payload. Autonomous cars currently rely on a large number of heavy sensors including 3D LIDARs, which are infeasible to carry aboard small flying platforms. Furthermore, processing the high volume of data produced by these sensors and planning long range trajectories given these observations of the world typically requires much more computation than a small multicopter or fixed-wing vehicle is capable of carrying. Therefore, while many planning, estimation and control algorithms for autonomous vehicles have been described in the literature, we must adapt these algorithms in order to achieve truly aggressive flight performance with computation and sensing that fit aboard a MAV.

To address the trajectory planning challenge, we exploit the property of differential flatness, which consists of an algebraic mapping from the state and input space of a dynamical system to a different set of output variables and their derivatives. This special property enables efficient algebraic calculation of the control inputs needed to exactly execute a given trajectory. Differentially flat representations pair naturally with trajectories that are expressed as polynomials because polynomials are differentiable and are easily constrained to maintain continuity of derivatives, yielding smooth trajectories.

While the quadrotor and fixed-wing vehicles are fundamentally different in their capabilities, they both have nonlinear dynamics models that are differentially flat. Polynomial trajectories for quadrotors have been demonstrated in the literature [38], and we extend this existing work in several ways. First, we use a fast low-dimensional search algorithm to automatically generate waypoints through the environment and use those waypoints as constraints in a polynomial optimization routine rather than hand-selecting waypoints. Next, we reformulate the existing constrained polynomial optimization method as an unconstrained quadratic program, yielding a method that is much more numerically stable for complex and long-range trajectories. Finally, we address the questions of selecting vehicle speeds along the trajectory subject to actuator constraints.

To plan aggressive trajectories for a fixed-wing aircraft, we present a novel trajectory formulation that builds upon Dubins curves. The high-level motion capabilities of a fixed-wing aircraft are well approximated by straight lines and circular arcs [34]. However, simple Dubins paths have discontinuous curvature at the intersection of segments, yielding paths that are dynamically infeasible for a real airplane. Therefore, we represent trajectories as underlying Dubins curves combined with polynomial transverse offsets from those Dubins curves. We refer to trajectories with this representation as Dubins-Polynomial trajectories. This strategy allows us to plan in the space of straight lines and circular arcs, and then approximate the resulting path with a polynomial that is constrained to maintain continuity of derivatives up to the necessary order, resulting in smooth and dynamically feasible trajectories. Furthermore, by parameterizing our trajectories as transverse offsets from Dubins curves, we isolate the transverse motions of the aircraft that correspond to roll dynamics and can easily optimize the polynomial to minimize roll rates and roll accelerations.

In order to execute aggressive trajectories accurately using our control sys-

tems, we need to estimate the vehicle state at arbitrary orientations encountered during agile flight maneuvers. To meet this challenge, we present a novel formulation of the Extended Kalman filter (EKF), which uses exponential coordinates of rotation to update its estimate of vehicle orientation and to propagate uncertainty about that orientation. In addition to the IMU, the primary sensor for localization onboard both the quadrotor and the fixed-wing vehicle is a 2D planar LIDAR. Incorporating the information provided by this sensor from arbitrary vehicle orientation is one of the key challenges in our state estimation method. 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 on a partition of the state space. The pseudo-measurement is computed from a Gaussian particle filter (GPF) state update [32]. This technique dramatically reduces the number of particles required compared to a basic implementation of a GPF, which in itself provides a marked improvement over a conventional particle filter [49]. Our algorithm enables accurate state estimation and realtime performance on an single-core Intel Atom processor running onboard the vehicle. The choice of filter formulation has the additional benefit of being platform independent. We use the identical sensors and filter algorithm on the quadrotor and the fixed-wing airplane without any modification.

We have conducted flight tests using our state estimation, planning, and control algorithms to perform autonomous flight in constrained indoor spaces with the quadrotor and fixed-wing vehicles. We give results of the fixed-wing vehicle completing a seven minute flight at 10 m/s through a parking garage with approximately 0.25 m clearance between the wingtips and the environment, as well as several aggressive flights between obstacles in a gymnasium. We also give results of the quadrotor completing multiple flights in obstacle-dense office-type spaces, traveling through doors, down hallways, and between hanging lights at up to 8 m/s.

The rest of this paper is organized as follows. We first describe the high-level approach to trajectory planning using polynomials coupled with a differentially flat model. Then we describe the state estimation algorithms that are common to both the quadrotor and fixed-wing aircraft. Next, we describe vehicle-specific aspects of trajectory planning and experimental results for both platforms. For the quadrotor, we show how the output of a low-dimensional path planner can be used to automatically generate waypoints that will be connected by polynomial trajectory segments, and we also discuss a method for ensuring that the resulting trajectory is both collision free and dynamically feasible given the input limits of the vehicle. We then describe the Dubins-Polynomial trajectory formulation that extends existing polynomial trajectory planning to Dubins-type vehicles. We have separated this section from the general discussion of planning for differentially flat vehicles due to the unique demands of a fixed-wing dynamics model. While we describe a method for fully automated planning for the quadrotor, we have not extended our technique to include the corresponding automatic waypoint generation capability for the fixed-wing aircraft. We leave it as a potential future extension of this work using a low-dimensional planner for Dubins-type vehicles ([11, 26]) Finally, we offer some discussion and conclusions.

Some of the results presented in this paper have appeared in our prior work. The quadrotor trajectory planning techniques and experimental results

appeared in [45]. The state estimation algorithms used in both the quadrotor and fixed-wing experiments appeared in [8]. The Dubins-Polynomial trajectory parameterization we use for the fixed-wing aircraft is a new contribution of this paper, and the corresponding experimental results represent a considerable advance in the experimental capabilities for fixed-wing aircraft flying outside of motion capture environments among obstacles.

# 2 Related Work

While many trajectory planning approaches exist in the literature, there has yet to emerge a single algorithm capable of finding and optimizing a trajectory for an agile 12-DOF vehicle through a complex real-world environment using only several seconds of computation. Yet, there is a strong need for such algorithms to be developed for MAVs, which have limited computational resources and limited flight time to accomplish their missions. We address this technical gap using a combination of capabilities from control theory and robotic motion planning.

## 2.1 Trajectory Planning

Trajectory planning research has given rise to successful demonstrations of aerial vehicles maneuvering at the limits of their dynamic capabilities. The control community has addressed the trajectory generation problem by casting it as a nonlinear optimization problem. In contrast, the robotics community has typically solved motion planning problems using the tools of discrete or randomized search over sequences of feasible actions. While both approaches are capable of generating similar solutions, they have different advantages and disadvantages depending on the application scenario.

Trajectory optimization has a long history in the control literature [7]. These problems can take many forms, but the overall concept is to optimize some sequence of inputs to a dynamical system using the tools of optimization, subject to constraints imposed by the differential equations of motion, control inputs, and obstacles in the environment. Often, the objective function will be to minimize time of traversal, energy usage, or some penalty weighting the relative costs of deviation from a goal state and the use of control inputs (i.e., LQR). Recently, these tools have been applied to generate extremely dynamic maneuvers for quadrotor helicopters [22] in motion-capture environments. For comparison, Barry et al. report 3-5 minutes of computation time [5] to optimize a 4.5 m trajectory for a 12-state, 5-input airplane maneuvering between several cylindrical obstacles using direct collocation with the SNOPT optimization software [4].

One disadvantage to conventional numerical trajectory optimization approaches is that obstacles in the world enter the problem as constraints, which must be represented analytically in the problem formulation. When the world is represented as an occupancy-grid map (common in robotics applications), trajectory optimization may become very cumbersome, although promising work has recently emerged in this area [44, 47]. In our case, the set of obstacles is stored in a 3D occupancy grid map containing millions of cells, rendering these approaches infeasible without significant effort to handle this source of complexity.

For the important subset of nonlinear systems that are differentially flat, the trajectory optimization problem can be mapped into a different set of variables called the flat output variables. There are several potential benefits of working in the flat output space. In particular, the states and inputs of a differentially flat system can be mapped algebraically to the the flat output variables and a number of their derivatives. Since derivatives are needed, it is natural to express the evolution of the flat output variables as a sequence of differentiable trajectories such as polynomials or B-splines. Trajectory optimization over the parameters of such splines may result in a lower-dimensional or otherwise more efficient solution [39]. The property of differential flatness combined with polynomial trajectories has led to efficient computation of very high performance trajectories for quadrotors [38] and is the topic of much of this paper. We discuss the benefits of differential flatness in the planning context in Section 4.

Roboticists have often favored search-based approaches to motion planning. Search over a graph representing feasible actions can be performed efficiently using conventional algorithms such as A* and Dijkstra's Algorithm. To handle collisions with the environment, edges that traverse occupied regions are simply deleted from the graph. Graph search has been successfully applied to motion planning for dynamic vehicles, including the CMU entry in the DARPA Urban Challenge [15]. When applied to dynamical systems, it is common to pre-compute a set of feasible actions connecting discretized locations in state space, which will serve as edges in the search graph. One challenge in using these methods is that there is no simple methodology for designing a set of actions that will sufficiently span the vehicle's true capabilities while remaining sparse enough for search to be efficient. Furthermore, graph search has worked well for automobiles, which can be modeled using 4 or 5 state variables. However, to plan aggressive trajectories for a general rigid body system in 12 dimensions, it is likely that graph search would succumb to the curse of dimensionality due to the exponential growth of the number of nodes in a search graph as a function of the number of dimensions [43].

To limit computational complexity, approximations or restrictions can be made to model the vehicle capabilities in a lower dimensional space. For example, the maneuvering capabilities of fixed-wing vehicles are often represented approximately as straight lines and arcs of constant curvature, reducing the airplane trajectory generation problem to a kinematic planning problem [11]. Frazzoli et al. present a method of planning with a maneuver automaton, in which a set of carefully selected dynamically feasible "trim primitives" can be concatenated into a complex motion plan [18]. Straight-line paths returned by a low-dimensional search can also be post-processed to be made feasible for nonholonomic vehicles [16].

In contrast with traditional graph search, randomized search algorithms have proven effective for planning in high-dimensional state spaces. One prominent example for dynamical systems is the Rapidly-Exploring Random Tree (RRT), which operates by building a randomized tree of feasible trajectories through forward-simulation of the equations of a motion [35]. Frazzoli et al. extend the RRT to operate with pre-computed library of feasible trajectories for agile vehicles subject to actuator constraints [17]. The Closed-Loop RRT (CL-RRT) is an alternative that was employed in the MIT entry in the DARPA Urban Challenge [33]. The CL-RRT differs from the RRT in that the tree of feasible trajectories is generated by forward-simulating a complete closed-loop control

system. The RRT and CL-RRT are very effective for generating feasible trajectories, however they are not designed to compute optimal trajectories, so the output trajectory may be excessively long or costly.

Recently, the RRT has been modified to converge to the globally optimal solution in the limit of infinite samples (RRT*) [27], and this approach has been extended to work with dynamical systems [26]. RRT* requires a steer-function in order to generate node connections in the search tree, and for dynamical systems, this steer function must solve a boundary-value problem to drive the system between two specified points in state space. RRT* solves the trajectory planning problem, however for general dynamical systems, this boundary-value problem is costly and dominates the practical complexity. In Section 4.4, we compare our approach against RRT*.

## 2.2 State Estimation

Aggressive flight of MAVs has introduced a new need for state estimation capabilities beyond what has been developed for ground robots or quadrotors maneuvering conservatively near the hover regime. Specifically, aggressive flight requires accurate, singularity-free estimation of arbitrary vehicle attitudes, which is not handled elegantly by existing filtering techniques. Furthermore, there is no estimator as efficient as the Kalman filter that incorporates the kind of measurement data provided by a LIDAR sensor. Yet, small LIDARs are some of the only sensors capable of accurate localization that can be carried aboard a small MAV.

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. [30] involves two Kalman filters where roll and pitch are determined by a filter driven by gyro readings as system inputs while 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 onboard 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 [50]. 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 an exponential coordinates-based attitude representation instead of the quaternions used by Van der Merwe et al. By using a three-parameter exponential coordinates representation, we avoid the problems of rank-deficient covariance matrices that are encountered when Kalman filtering any attitude representation using four or more parameters. Furthermore, with this representation, we do not need to enforce any constraints between attitude parameters to ensure that our estimated attitude is in $SO(3)$.

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 [37]. 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 rangefinders combined with particle filter-based localization is widely used in ground robotic systems [49]. 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 [3], as well as others [48, 19] leveraged a 2D laser rangefinder to perform SLAM from a quadrotor in GPS-denied environments. These 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 these 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. Specifically, they require walls that are at least locally vertical, and a mostly flat floor for height estimation. As a result, these 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 at high speed [46], however they rely on accurate GPS measurements for 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 [29], as well as larger scale helicopters [10, 28, 24]. 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 computational complexity of vision algorithms and sensitivity to lighting and environment conditions.

Recently, Hesch et al. [23] 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 micro air vehicle. Furthermore, they model the environment as a set of planar structures aligned with one of 3 principle axes, which severely 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.

# 3 Nomenclature and Coordinate Frames

The state of a rigid body dynamical system is described by the quantities $\mathbf{x} = [\omega_b, v_b, \mathbf{R}, \mathbf{\Delta}]$, where $\omega_b = [p, q, r]$ is the angular velocity in body coordinates, $\mathbf{v}_b = [u, v, w]$ is the linear velocity in body coordinates, $\mathbf{R}$ is the rigid body orientation rotation matrix, and $\mathbf{\Delta} = [\Delta_x, \Delta_y, \Delta_z]$ is the translation vector from the world origin to the center of mass of the body, expressed in global coordinates. We will refer to this rigid body state in the context of state esti-

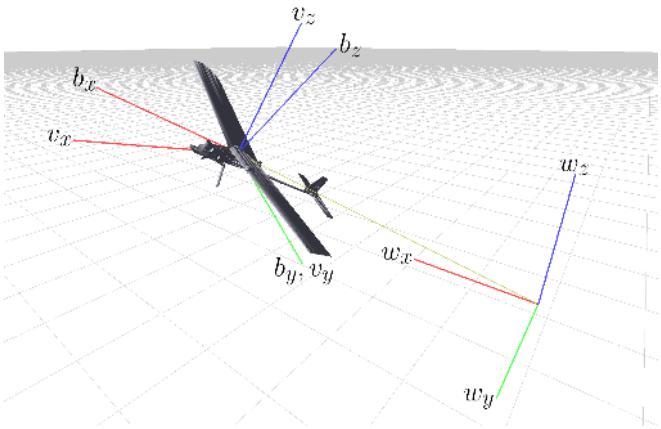



Figure 1: The global frame, body frame and velocity frame depicted with $x$-, $y$-, and $z$-axes in red, green, and blue respectively. The ENU global frame is fixed to the ground (with the $x$-direction pointing East). The body frame is fixed to the vehicle with the $x$-direction pointing forward, aligned with the longitudinal vehicle axis. The velocity frame, assuming coordinated flight, shares the $y$-axis with the body frame and has its $x$-axis aligned with velocity.

mation and in describing our vehicle models.

For planning and control, we will make use of three coordinate frames. The world frame, denoted with the letter $w$, is a static East-North-Up (ENU) coordinate frame anchored at some origin in the world. The body frame is fixed to the center of mass of the the aircraft, with the $x$-direction pointing forward and aligned with the longitudinal axis of the vehicle, the $y$-direction pointing left, and the $z$-direction pointing up (FLU). The body frame is denoted with the letter $b$, and quantities expressed in body frame coordinates will be denoted with a subscript $b$. The transformation between world and body frames is defined by the translation vector $\mathbf{\Delta}$ and orientation $\mathbf{R}$. The third frame is a "velocity" frame, which shares its origin and $y$-direction with the body frame, but has its $x$-axis aligned with the body velocity direction. The velocity frame is denoted with the letter $v$. This convention is depicted in Figure 1.

The use of the ENU-FLU coordinate frame runs counter to the convention of aerospace control literature, however it is consistent with more recent work using quadrotors indoors (which 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. We will be estimating the velocity and angular velocities in the body frame ($\mathbf{v}_b$, $\omega_b$), however in other contexts it is convenient to discuss velocity and acceleration in the world frame. In these cases we will refer to derivatives of position for clarity (i.e., $\dot{\mathbf{\Delta}}$, $\ddot{\mathbf{\Delta}}$).

If we wish to refer to a single component of a vector expressed in a certain frame, we will use the subscript of the associated unit vector. For example, the scalar forward velocity aligned with the longitudinal axis of the vehicle ($\mathbf{b}_x$) will be denoted $v_{b_x}$.

# 4 Trajectory Planning for Aggressive Flight

Given a 3D occupancy map of the environment and specified start and goal locations, we wish to compute dynamically feasible trajectories from start to goal, which do not intersect obstacles in the environment, and which minimize a penalty on the derivatives of the trajectory. To quantify our computational efficiency requirement, we require that the solution take no more than a matter of seconds to generate. This is not a hard time constraint, and the difficulty of planning will scale with the size and complexity of the environment. However, our aim is to plan trajectories much faster than existing optimal planning algorithms that might be used instead. In Section 4.4 we show that solving our planning problem with RRT* in the full 12-DOF state space of a quadrotor would take many minutes to converge, exceeding the 10-minute flight time of our MAVs. For practical use in missions consisting of multiple or changing destinations, we need to be able to generate new plans much more efficiently.

As outlined in Section 2.1, there exist many algorithms capable of planning trajectories for dynamical systems. However, there remains a significant gap in the practical capabilities of these algorithms to quickly find and optimize long-range trajectories for 12-DOF dynamical systems. Algorithms which perform the dual functions of search and optimization, such as graph search (A*) or randomized search (RRT*) will succumb to the curse of dimensionality when faced with the full dynamics of a quadrotor or fixed-wing vehicle. Therefore, these algorithms will not be able to return an optimal or near-optimal trajectory in a matter of seconds, but may instead take many minutes to terminate. On the other hand, planning algorithms which find trajectories quickly but offer no optimization capabilities, such as the RRT or CL-RRT, may return solutions that are overly high in cost according to some function, or may require abrupt or excessive control inputs.

It is useful to minimize certain trajectory derivatives for several reasons. First, these derivatives map directly to the required control inputs, and we would like to avoid unnecessary power requirements and wear on our system due to excessive actuation. Additionally, reducing the required control inputs along a trajectory helps to retain maximum control authority for feedback stabilization. Finally, trajectories requiring abrupt motions tend to excite high-order dynamic effects that are difficult to model and control, such as turbulence, actuator response and latency. Roughly speaking, minimizing derivatives can encourage "graceful" motions. For quadrotors, minimizing snap (fourth derivative) limits the motor commands required to follow a trajectory [38], while for fixed-wing vehicles, we may instead prefer to minimize the third derivative to limit roll rates. If we select polynomials as our trajectory basis function, then minimizing these cost functions on the derivatives is straightforward and computationally efficient. Furthermore, if we exploit the property of differential flatness for our vehicle models, then the mapping from trajectory derivatives to control inputs is an algebraic expression, and we can easily verify dynamic feasibility.

In the next section, we describe the role of differential flatness in the context of planning. Then, we develop the necessary tools for optimizing polynomial trajectories in order to take advantage of differential flatness and minimize our desired cost function on the trajectory derivatives. The optimization itself does not directly incorporate dynamic or kinematic constraints. Finally, in Section 4.4, we will combine these polynomial optimization tools with a low-dimensional

search algorithm to develop a full planning procedure that can both find *and* optimize trajectories for differentially flat systems in a matter of seconds.

## 4.1 Differential Flatness for Trajectory Planning

While very detailed models are available for the quadrotor and fixed-wing vehicles, capturing complex aerodynamic effects and detailed input response characteristics, we instead focus on planning with particular simplified representations. Specifically, we focus on representations for these vehicles that are differentially flat. For our purposes, the result of differential flatness is an algebraic mapping from a set of flat output variables and their derivatives to the vehicle states and inputs [51]. While there is no systematic way of determining whether such a mapping exists in general, flatness has been shown for a large number of useful systems [40].

The mapping between vehicle states and inputs and the flat output space is what makes differential flatness useful for trajectory planning. The challenge in trajectory optimization and motion planning for dynamical systems is to find a (minimum cost) sequence of control inputs that will carry the system between specified initial and terminal states subject to the differential equations of the system. This problem can be framed as a numerical optimization or graph search, but in either case, it involves an expensive search over the set of possible input sequences. The advantage of differential flatness in the context of trajectory planning is that a trajectory can be specified directly in the flat output space, and the control inputs needed to execute that trajectory can be obtained as an algebraic function of the trajectory and its derivatives. For many systems, it may be much easier to specify and optimize a trajectory in the flat output space than it is to perform a search over the input space.

In general, the flat output variables need not correspond to convenient physical quantities, but often they do. For the quadrotor, the four flat output variables are the three components of the global position vector along with the heading angle. Likewise, the three flat output variables for the fixed-wing vehicle are also the three components of the global position vector. Therefore, it is easy to specify start and goal locations in the flat output space, as well as to check for collisions with obstacles in our three dimensional environment.

To recover the states and inputs to the vehicle along a trajectory, we need not only the values of the flat outputs along the trajectory, but also their derivatives. Intuitively, the states and inputs of a vehicle at some point along a trajectory must depend not only on the position, but also the velocity, acceleration and higher order derivatives of the trajectory at that point. Therefore it is convenient to define our trajectories using a differentiable basis function. While there are a variety of choices that would work in this setting, we have chosen polynomials due to their analytical and computational tractability.

## 4.2 Polynomial Trajectory Optimization

Consider the evolution of a flat output variable over some time interval $t \in [0, \tau]$, such as a position coordinate, prescribed by a polynomial $P(t)$ between two points in the space of flat outputs. We would like to select the coefficients of $P(t)$ such that its endpoints at $t = 0$ and $t = \tau$ (and the derivatives at those endpoints) match those that have been specified. With any remaining degrees of

freedom, we would like to optimize some cost function of the derivatives of the polynomial. Mellinger and Kumar use the minimum-snap (fourth derivative) cost function for quadrotors [38]. This cost function effectively discourages abrupt changes in the motor commands to the quadrotor, leading to graceful trajectories. For other systems, we may prefer to minimize some other weighted combination of the derivatives of the trajectory.

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

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

$$= \sum_{n=0}^{N} p_n t^n. \tag{2}$$

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

$$J = \int_{t=0}^{t=\tau} c_0 P(t)^2 + c_1 P'(t)^2 + c_2 P''(t)^2 + \ldots + c_N P^{(N)}(t)^2 dt, \tag{3}$$

which can be written in quadratic form as

$$J = \mathbf{p}^T \mathbf{Q} \mathbf{p}, \tag{4}$$

where $\mathbf{p} \in \Re^{N+1}$ is the vector of polynomial coefficients and $\mathbf{Q}$ is a cost matrix corresponding to our desired penalty on each of the polynomial derivatives. In order to minimize the fourth derivative of a polynomial, for example, we would set $c_4 = 1$ and set $c_i = 0$ for $i \neq 4$.

In addition to specifying the cost function, we must also constrain the solution to enforce certain conditions on the value and derivatives of the initial and final endpoints of the polynomial. The constraints take the following form:

$$\mathbf{A}_0 \mathbf{p} = \mathbf{b}_0 \tag{5}$$

$$\mathbf{A}_\tau \mathbf{p} = \mathbf{b}_\tau \tag{6}$$

where $\mathbf{A}_0$ and $\mathbf{A}_\tau$ are matrices that map the coefficients $\mathbf{p}$ to the derivatives of the polynomial at the beginning and end, respectively. The vectors $\mathbf{b}_0$ and $\mathbf{b}_\tau$ indicate the values of the derivatives we wish to constrain. Each row of equations (5) and (6) corresponds to a particular derivative that we wish to constrain, including the $0^{\text{th}}$ derivative, which allows us to constrain the value of the polynomial (i.e., the position). In Appendix A, we show how to derive the $\mathbf{Q}$ and $\mathbf{A}$ matrices.

By concatenating the desired constraints on the initial and final derivatives of the polynomial, we have a standard quadratic programming problem

$$\begin{aligned} \min_{\mathbf{p}} \quad & \mathbf{p}^T \mathbf{Q} \mathbf{p} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{p} - \mathbf{b} = 0 \end{aligned} \tag{7}$$

which we can solve using an elimination approach [6].

### 4.2.1 Piecewise Polynomial Joint Optimization

In general, the trajectory of a given flat output variable will consist not of a single polynomial segment, but of a sequence of several segments. We will

be optimizing trajectories that pass through a sequence of waypoints in the output space of the differentially flat system. These waypoints will be obtained from a kinematic planning algorithm capable of finding a route through the environment, such as A* or RRT*.

The trajectory between each pair of waypoints will be an individual polynomial trajectory segment. Rather than optimizing each of these polynomials in isolation, we may obtain lower-cost trajectories by jointly optimizing the entire sequence of polynomials at once. Furthermore, we need the two polynomial segments that meet at every junction or waypoint to agree on the value of the trajectory derivatives at that junction. Joint optimization is a natural way to enforce these continuity constraints.

A piecewise polynomial describing the evolution of a flat output variable can be constructed as:

$$T(t) = \begin{cases} P_0(t): & 0 \leq t \leq \tau_0 \\ P_1(t-\tau_0): & \tau_0 < t \leq \tau_1 + \tau_0 \\ P_2(t-(\tau_0+\tau_1)): & \tau_0 + \tau_1 < t \leq \tau_0 + \tau_1 + \tau_2 \\ \vdots & \end{cases} \tag{8}$$

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

$$T(t) = P_k(t - \Gamma_{k-1}): \qquad \Gamma_{k-1} < t \leq \Gamma_k. \tag{9}$$

We can formulate an optimization over the polynomial coefficients by forming a vector containing all the coefficients $\mathbf{p}_i$ of all of the polynomial segments:

$$\mathbf{p} = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 & \cdots & \mathbf{p}_K \end{bmatrix} \tag{10}$$

The cost matrix is constructed as a block diagonal matrix composed of individual $\mathbf{Q}_k$ matrices. The constraints for specified derivative values are composed as a concatenation of equations of the form in equations (5) and (6). However, if we do not specify particular derivative values, we still wish to enforce continuity of the derivatives at each junction between segments. Constraints enforcing continuity take the form:

$$\begin{bmatrix} -\mathbf{A}_\tau^i & \mathbf{A}_0^{i+1} \end{bmatrix} \begin{bmatrix} \mathbf{p}_i \\ \mathbf{p}_{i+1} \end{bmatrix} = \sigma_i \tag{11}$$

where $\sigma_i$ is a vector of derivative offsets between polynomials $i$ and $i+1$. Setting this vector to zero will enforce continuity, but it will be necessary when we discuss trajectories for the fixed-wing vehicle to enforce nonzero offsets. The full constraint equation to enforce continuity (or derivative offsets) is:

$$\begin{bmatrix} -\mathbf{A}_\tau^0 & \mathbf{A}_0^1 & 0 & 0 & \dots & 0 & 0 \\ 0 & -\mathbf{A}_\tau^1 & \mathbf{A}_0^2 & 0 & \dots & 0 & 0 \\ 0 & 0 & -\mathbf{A}_\tau^2 & \mathbf{A}_0^3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & -\mathbf{A}_\tau^{K-1} & \mathbf{A}_0^K \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \vdots \\ \mathbf{p}_{K-1} \\ \mathbf{p}_K \end{bmatrix} = \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{K-1} \end{bmatrix} \tag{12}$$

We can also combine this equation with a second set of constraints to specify exact values of derivatives that we wish to fix:

$$\begin{bmatrix} \mathbf{A}_0^0 & 0 & 0 & 0 & \dots & 0 \\ 0 & \mathbf{A}_0^1 & 0 & 0 & \dots & 0 \\ 0 & 0 & \mathbf{A}_0^2 & 0 & \dots & 0 \\ 0 & 0 & 0 & \mathbf{A}_0^3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \mathbf{A}_0^K \\ 0 & 0 & 0 & 0 & \dots & \mathbf{A}_\tau^K \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \\ \vdots \\ \mathbf{p}_K \end{bmatrix} = \begin{bmatrix} \mathbf{b}_0^0 \\ \mathbf{b}_0^1 \\ \mathbf{b}_0^2 \\ \mathbf{b}_0^3 \\ \vdots \\ \mathbf{b}_0^K \\ \mathbf{b}_\tau^K \end{bmatrix} \tag{13}$$

where $\mathbf{b}_0^i$ for $i = 0, \dots, K-1$ is a vector of specified derivatives at the beginning of the $i^{th}$ piecewise polynomial segment and $\mathbf{b}_\tau^K$ is a vector of specified derivatives at the end of the final segment. Note that if we specify continuity using equation (12), then we need only specify derivatives at the beginning of each segment because the corresponding derivatives at the end of the preceding segment will also be enforced through continuity. In practice, some combination of the rows of equations (12) and (13) will enable us to specify the desired set of derivative values as well as enforce continuity up to the desired order. Having constructed the desired system of constraints, the joint optimization can then be performed using an elimination approach [6].

This constrained optimization method works well for small joint optimization problems as in [38], however this formulation becomes ill-conditioned for more than several segments, polynomials of high order, and when widely varying segment times are involved. Hence, it is only useful for short trajectories and must be improved for optimizing long range paths requiring many waypoints and segments.

We avoid the problems of ill-conditioning using a technique of substitution to convert the problem into an unconstrained QP, which allows us to solve directly for endpoint derivatives as decision variables, rather than solving for polynomial coefficients. In practice, our reformulation is substantially more stable than the constrained formulation, allowing the joint optimization of more than 50 polynomial segments in a single matrix operation without encountering numerical issues. Once the optimal waypoint derivatives are found, the coefficients of the polynomial connecting each pair of waypoints can be obtained by solving a small linear system with the appropriate constraint matrix.

To convert to an unconstrained optimization, we begin by substituting $\mathbf{p}_i = \mathbf{A}_i^{-1}\mathbf{b}_i$ from the $i^{th}$ individual segment constraint equations into the original cost function:

$$J = \begin{bmatrix} \mathbf{b}_0 \\ \vdots \\ \mathbf{b}_K \end{bmatrix}^T \begin{bmatrix} \mathbf{A}_0 & & \\ & \ddots & \\ & & \mathbf{A}_K \end{bmatrix}^{-T} \begin{bmatrix} \mathbf{Q}_0 & & \\ & \ddots & \\ & & \mathbf{Q}_K \end{bmatrix} \begin{bmatrix} \mathbf{A}_0 & & \\ & \ddots & \\ & & \mathbf{A}_K \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{b}_0 \\ \vdots \\ \mathbf{b}_K \end{bmatrix} \tag{14}$$

Now the decision variables in this new quadratic cost function are the endpoint derivatives of the segments, $\mathbf{b}_i$. We re-order these variables such that fixed/specified derivatives are grouped together ($\mathbf{b}_F$) and the free/unspecified derivatives are grouped together ($\mathbf{b}_P$). A permutation matrix of ones and zeros ($\mathbf{C}$) is used to accomplish this re-ordering. Now we have:

$$J = \begin{bmatrix} \mathbf{b}_F \\ \mathbf{b}_P \end{bmatrix}^T \underbrace{\mathbf{C}\mathbf{A}^{-T}\mathbf{Q}\mathbf{A}^{-1}\mathbf{C}^T}_{\mathbf{R}} \begin{bmatrix} \mathbf{b}_F \\ \mathbf{b}_P \end{bmatrix} = \begin{bmatrix} \mathbf{b}_F \\ \mathbf{b}_P \end{bmatrix}^T \begin{bmatrix} \mathbf{R}_{FF} & \mathbf{R}_{FP} \\ \mathbf{R}_{PF} & \mathbf{R}_{PP} \end{bmatrix} \begin{bmatrix} \mathbf{b}_F \\ \mathbf{b}_P \end{bmatrix} \tag{15}$$

where we have written the block-diagonal matrices as $\mathbf{A}$ and $\mathbf{Q}$ for simplicity of notation. We group the new augmented cost matrix into a single matrix $\mathbf{R}$ and partition it according to the indices of the fixed and free derivatives. Partitioning allows us to write out the expression for total cost as:

$$J = \mathbf{b}_F^T\mathbf{R}_{FF}\mathbf{b}_F + \mathbf{b}_F^T\mathbf{R}_{FP}\mathbf{b}_P + \mathbf{b}_P^T\mathbf{R}_{PF}\mathbf{b}_F + \mathbf{b}_P^T\mathbf{R}_{PP}\mathbf{b}_P \tag{16}$$

Differentiating $J$ and equating to zero yields the vector of optimal values for the free derivatives in terms of the fixed/specified derivatives and the cost matrix:

$$\mathbf{b}_P^* = -\mathbf{R}_{PP}^{-1}\mathbf{R}_{FP}^T\mathbf{b}_F \tag{17}$$

The polynomials can now be recovered from individual evaluations of the appropriate constraint equations mapping derivatives back into the space of coefficients.

#### 4.2.2 Performance of Polynomial Optimization

A key to the success of this trajectory planning process is the speed and numerical stability of the joint polynomial optimization method. We performed benchmark tests on an example problem consisting of four waypoints (3 polynomial segments) chosen to represent distance and time scales consistent with common environments for quadrotor flight. The results are given in Table 1 and reflect MATLAB as well as C++/Eigen implementations [20]. This computational efficiency makes it feasible to use this planning framework in online applications and to use iterative path refinement methods with polynomial optimization in the loop.

Table 1: Comparison of Polynomial Optimization Times.

| Benchmark Problem: 3-Segment Joint Optimization | |
|---|---|
| Method | Solution Time (ms) |
| MATLAB `quadprog.m` | 9.5 |
| MATLAB Constrained | 1.7 |
| MATLAB Unconstrained (Dense) | 2.7 |
| C++/Eigen Constrained | 0.18 |
| C++/Eigen Unconstrained (Dense) | 0.34 |

While the unconstrained formulation is slightly slower than the constrained formulation, its primary benefit lies in its stability. The constrained formulation encounters matrices very close to singular for joint optimizations consisting of more than three $9^{\text{th}}$ order polynomials, and therefore may return inaccurate results depending on the quality of the linear algebra solver. In contrast, the unconstrained formulation is robust to numerical issues, as shown in Table 2, which lists the results of 20 polynomial optimization problems in which the

locations of intermediate waypoints and the segment times were randomly generated in the range $[1, 3]$. Clearly, the unconstrained optimization is much more robust to numerical instability, enabling this method to be used as a reliable, efficient long-range trajectory optimization tool for navigation outside of small motion-capture environments.

Table 2: Numerical stability of optimization techniques for high-order polynomials and various numbers of segments.

| Success Rates on Randomized Polynomial Optimization Problems | | | |
|---|---|---|---|
| Formulation | Polynomial Order | Number of Segments | Success |
| Constrained | 9 | 3 | 100% |
| | 9 | 4 | 55% |
| | 9 | $\geq$5 | 0% |
| **Unconstrained** | 9 | 50+ | 100% |
| | 15 | 50+ | 100% |

Since $\mathbf{A}^{-1}$ and $\mathbf{Q}$ are sparse block-diagonal and $\mathbf{C}$ is sparse, these problems can be implemented using a sparse solver which is roughly an order of magnitude faster than the dense computation for 10-segment joint optimizations.

The unconstrained optimization is more numerically stable because it is solving for waypoint derivative values such as velocities, accelerations, etc., that are similar in magnitude. The resulting polynomial segments are not particularly sensitive to the exact values of these derivatives. On the other hand, the constrained optimization solves simultaneously for the full set of polynomial coefficients, which range in magnitude from approximately $10^1$ down to $10^{-16}$ (i.e., machine precision). The resulting solution is extremely sensitive to the values of small coefficients corresponding to high-order terms, which may be inaccurate due to rounding errors.

## 4.3 General Polynomial Trajectory Representation

One common trajectory representation for differentially flat systems is to specify a separate polynomial for each of the flat output variables of the system. For a quadrotor, for example, the four flat output variables are the $x$-, $y$-, and $z$-position coordinates and the yaw angle, $\psi$. Therefore, we can specify a polynomial trajectory through space as the set of piecewise-polynomial trajectories $\{P_x(t), P_y(t), P_z(t), P_\psi(t)\}$ describing these four variables. An example of a polynomial trajectory is given in Figure 2. This general trajectory formulation could also be used to describe the motions of rigid body chains, robotic arms, or a variety of other differentially flat systems.

For the quadrotor, motor inputs required to follow these trajectories are functions of the trajectory derivatives up to fourth order. Typically we will use polynomials with 10 coefficients ($9^{\text{th}}$ order) so that we can specify all ten derivative values if necessary. For example, for a single polynomial representing $\Delta_x(t)$, we may want to specify:

$$\mathbf{b}_0 = [\Delta_x(0), \dot{\Delta}_x(0), \ddot{\Delta}_x(0), \dddot{\Delta}_x(0), \ddddot{\Delta}_x(0)]^T \tag{18}$$

$$\mathbf{b}_\tau = [\Delta_x(\tau), \dot{\Delta}_x(\tau), \ddot{\Delta}_x(\tau), \dddot{\Delta}_x(\tau), \ddddot{\Delta}_x(\tau)]^T \tag{19}$$

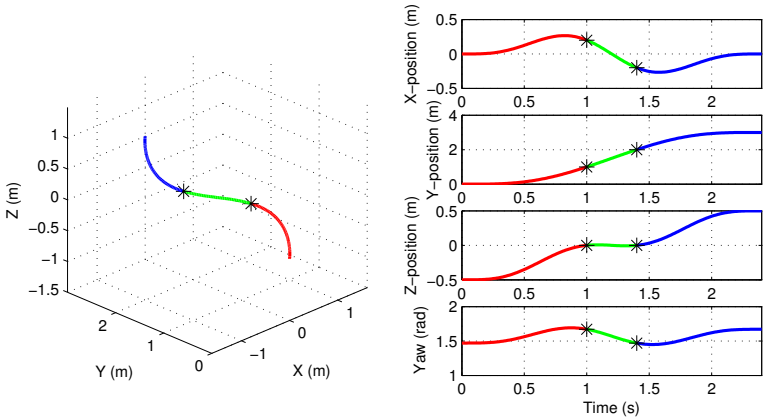



Figure 2: Example piecewise polynomial trajectory for a quadrotor. The left panel illustrates a trajectory through space. The right panel shows the evolution of the $x$, $y$, $z$ and $\psi$ coordinates through time.

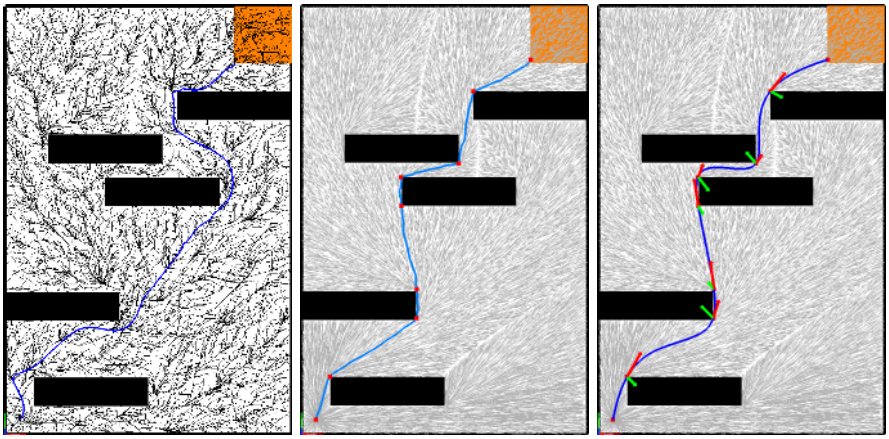
requiring 10 parameters to satisfy these constraints. In general, we will specify some subset of these derivatives, leaving the remaining parameters free for optimization.

Often we will constrain the trajectory to start and end at rest and constrain the higher derivatives to also begin at zero, as seen at the beginning of the red polynomial segments and the end of the blue segments on the right panel of Figure 2. For intermediate waypoints, we will only specify positions. The velocity, acceleration, and higher derivatives will be selected by optimization of the polynomial coefficients. This strategy is convenient for planning if we have access to a sequence of waypoints with which to constrain the optimization. In the next section we will describe a method of obtaining waypoints and using them to construct a polynomial trajectory.

## 4.4 Planning with Polynomial Trajectories

Given a 3D occupancy map of an environment, we wish to efficiently compute feasible polynomial trajectories from start to goal, while respecting obstacle constraints and input saturation limits. In the previous section, we developed the tools for polynomial optimization, which will allow us to construct trajectories that are constrained to pass through a sequence of waypoints in the flat output space. In [38], the initial and final states of the trajectory, as well as the locations of all intermediate waypoints were selected by hand. However, in order to adapt this trajectory optimization method to function in a real-world planning context, we need a way of obtaining these constraints automatically. In this section, we develop the additional methods required to apply polynomial optimization to solve the trajectory planning problem.

First, we must obtain a sequence of waypoints through the environment, which function as constraints guiding the polynomial trajectories from start to goal. Our solution to this problem is to utilize the RRT* algorithm to find a 3D collision-free path through the environment, initially considering only the kinematics of the vehicle and ignoring the dynamics. A variety of other search and



(a) (b) (c)

Figure 3: Using polynomial segments directly as an RRT* steer function (a) is computationally slow. Therefore, we run a straight-line RRT* and select waypoints from the optimal path (b). However, the straight-line RRT* ignores dynamics and returns a path that does not minimize our objective function. We therefore jointly optimize a set of polynomials through those waypoints to obtain a minimum-snap trajectory (c).

planning algorithms, such as A*, could also serve this purpose. The resulting path is pruned to a minimal set of waypoints, and a sequence of polynomial segments is jointly optimized to join those waypoints into a smooth minimum-snap trajectory from start to goal. Specifically, the waypoint positions comprise the constraint values ($\mathbf{b}_i$) for the intermediate polynomial segments in the optimization presented above. As a result of these constraints, the optimized polynomial trajectory will pass through each of these waypoints. At this point, it is possible that one or more of the polynomial trajectory segments will be in collision with the environment. In section 4.4.2 we describe our method for repairing such collisions.

A general search over the space of minimum-snap trajectories would be extremely expensive, though it would in principle return the globally optimal trajectory. To validate our approach, we ran an RRT* using individual polynomial trajectory segments as the "steer" function to grow the search tree of feasible trajectories. Figure 3(a) shows the resulting solution. Sampling for the RRT* with the polynomial steer function was performed in position and velocity space. We use the distance metric described by [25] of Euclidean distance divided by average velocity. One major difficulty with this approach is that segment times must be fixed when generating polynomials to extend the tree, however the selection of segment time can have a dramatic impact on the quality of a path, so an appropriate guess must be made *a priori* for each segment, or the segment time must be included in the sampling space. In our implementation, the segment times were chosen as the Euclidean distance between vertices divided by the desired average velocity along the segment.

Table 3 shows several statistics on the performance of the RRT* with a

Table 3: Comparison of our method with RRT* using the polynomial steer function for the 2D problem in Figure 3.

| Method | Runtime | $J_{poly.}$ | $\tau_{path}$ | $L_{path}$ |
|---|---|---|---|---|
| RRT* (Polynomial Steer) | 120 s | $5.72 \times 10^8$ | 21.94 s | 40.35 m |
| **Our Approach** | 3 s | $1.07 \times 10^5$ | 19.66 s | 35.51 m |

polynomial steer function compared to our algorithm. The RRT* runs much longer and fails to find a path as smooth or with a cost as low as our algorithm. When sampling in the full state space of the system, the RRT* with a polynomial steer function would converge to a globally optimal solution in the limit of infinite samples, however as shown here, the paths returned prior to convergence are of lower quality than those returned by our algorithm in a much shorter running time.

### 4.4.1 Time Allocation

Until this point in the trajectory optimization, we have fixed an arbitrary amount of time $\tau_i$ associated with each segment, since these times factor into the construction of the cost matrix. These segment times constrain the solution quality, but can be allowed to vary to improve the overall solution with respect to a cost function. We therefore begin with an initial guess of segment times and then iteratively refine those times using gradient descent. Several cost functions may be suitable candidates: in [12], Cutler and How minimize total time subject to constraints, while in [38], Mellinger and Kumar fix the total time by hand and minimize snap (the original cost function) with the remaining degrees of freedom. In the planning context, we do not know the total trajectory time *a priori*, so we allow it to vary in the optimization to perform a trade-off between minimizing snap and total trajectory time. We attempt to minimize:

$$J_\tau = \begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_K \end{bmatrix}^T \begin{bmatrix} \mathbf{Q}_0(\tau_0) & & \\ & \ddots & \\ & & \mathbf{Q}_K(\tau_K) \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_K \end{bmatrix} + c_\tau \sum_{i=0}^{K} \tau_i \tag{20}$$

where $c_\tau$ is a user-specified penalty on time. The first term in this cost function is simply the original cost function for polynomial optimization. When penalizing only acceleration, jerk or snap, this original cost can be driven arbitrarily close to zero by increasing the total time, but equation (20) has a minimum value for some finite $\sum_{i=0}^{K} \tau_i$ that varies with $c_\tau$. Figure 4 shows several iterations of gradient descent in which the total trajectory time is decreased from a large initial guess (red) to smaller optimal value (blue), while the ratio of times between segments also shifts to minimize the modified cost.

Rather than selecting total times arbitrarily, this cost function allows our algorithm to automatically adjust for environments of widely varying scales, or where the vehicle must slow down to navigate tightly spaced obstacles without incurring excessive snap. Furthermore, our procedure produces trajectories of comparable aggressiveness in a wide range of scenarios for a given fixed value of the single scale-independent parameter, $c_\tau$.

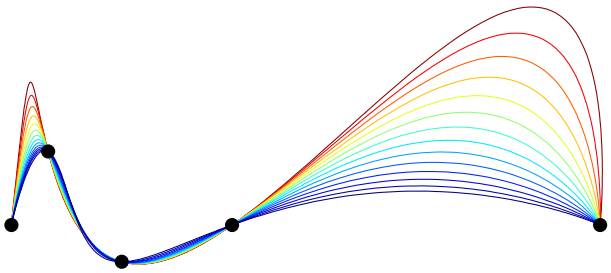
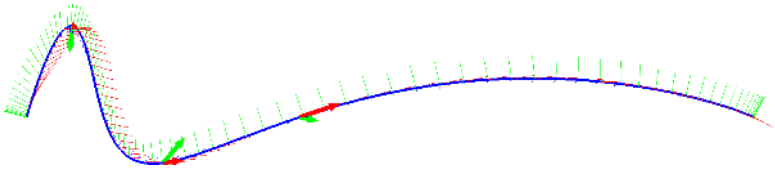
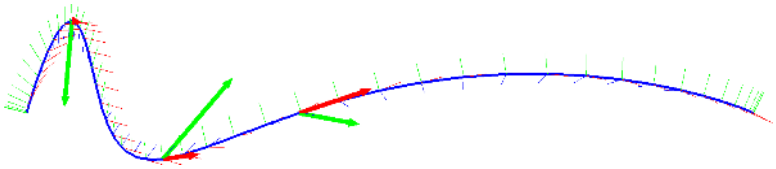


Figure 4: Illustration of the iterative refinement of segment times, color-coded by total traversal time. The initial guess of total time is 10.5 s (red) and the final optimized total time is 7 s (blue).



(a)



(b)

Figure 5: Segment time optimization with $c_\tau = 500$ (a) and $c_\tau = 50000$ (b). The optimal total trajectory times are 9.1 s and 5.1 s respectively. Vectors for waypoint velocity (red) and acceleration (green) are shown.

Figure 5 shows optimized trajectories for the same set of waypoints using two different $c_\tau$ values. The red arrows indicate waypoint velocities and the green arrows indicate accelerations. These quantities are greater in the bottom trajectory due to the higher time penalty. The vehicle axes are plotted at 0.1 s increments along the path. One emergent property resulting from time allocation is that the system moves very slowly around the sharp corner and then smoothly accelerates up to a higher speed in the straightaway where it does not incur a severe penalty on its fourth derivative. This same behavior could be equally useful for robotic arms executing smooth trajectories. Furthermore, the geometric shape of the optimal trajectory remains the same regardless of the value of $c_\tau$, indicating that the cost-minimizing *ratios* of segment times are independent of $c_\tau$ for a given arrangement of waypoints.

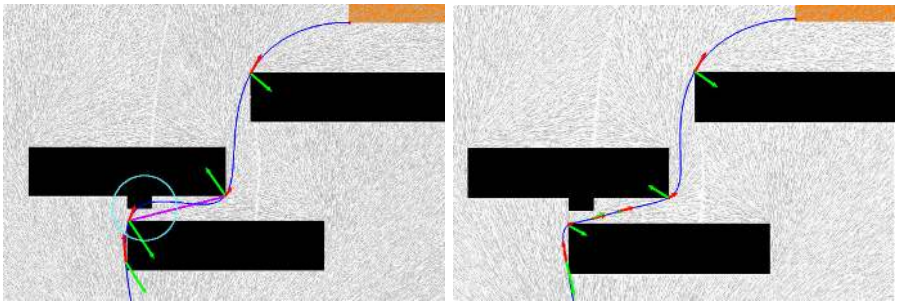
### 4.4.2 Ensuring the Trajectory is Collision-Free

If a particular trajectory segment is found to intersect an obstacle after optimization, an additional waypoint is simply added halfway between its two ends,

splitting this segment into two, as in [41]. This midpoint is known to be collision-free because it lies on the optimal piecewise-linear path returned by the search algorithm. The polynomial is re-optimized (including time allocation) with the additional waypoint, and the process is repeated recursively if necessary until the polynomial trajectory is collision free.

A finite number of additional waypoints is sufficient to pull the polynomial trajectory close enough to the underlying piecewise-linear path to repair collisions as long as the corridor between obstacles is strictly wider than the vehicle. As the number of waypoints increases, the polynomial trajectory converges to the piecewise-linear path, which is known to be feasible for the quadrotor (we do not imply that this technique would work for a fixed-wing aircraft, however). Figure 6 illustrates this procedure resolving a collision.

It is possible to construct environments and trajectories that require many additional waypoints to repair collisions, thus requiring the optimization problem to be re-solved many times to find a feasible solution. Additional waypoints also increase the computational complexity of the QP being solved in each iteration. However, in our experience with indoor environments, one or two additional waypoints in a given segment is usually sufficient to resolve collisions.



(a) Polynomial trajectory (blue) intersects an obstacle even though the underlying straight line between waypoints is collision-free.

(b) After bisecting the underlying straight line twice with two additional waypoints, the polynomial trajectory is collision-free.

Figure 6: (a) The polynomial (blue) intersects an obstacle even though the line between waypoints is collision free (magenta). (b) These scenarios are resolved by iteratively adding waypoints along the collision-free path returned by the search algorithm.

There are several strategies that could be used to prevent the number of necessary additional waypoints from growing too large. One option would be to bias the underlying search away from potential collisions by placing an elevated cost on paths that go near obstacles. Another option would be to inflate obstacles slightly larger during the low-dimensional route-finding phase to leave an allowance for the smooth polynomial to deviate from the underlying piecewise-linear path. We leave the detailed analysis of collision-repair strategies for future work.

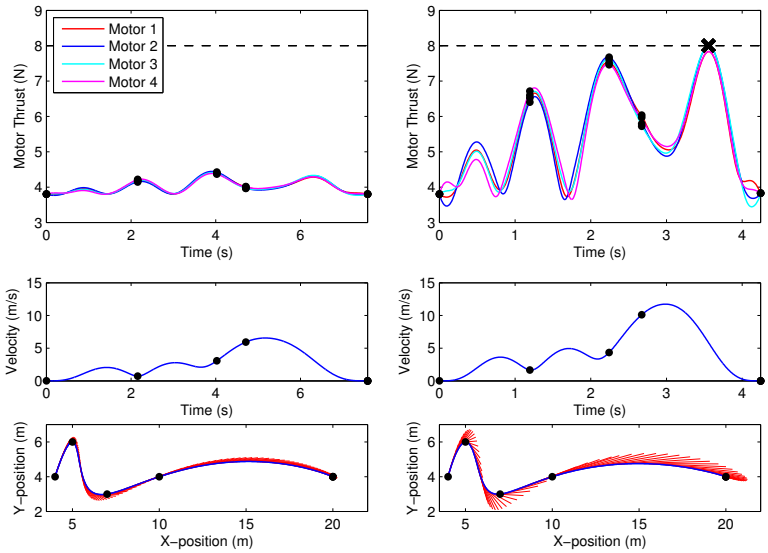



Figure 7: Comparison between two time allocations during gradient descent. The first time allocation (a) is conservative in that it is a slower trajectory than the second one (b), which activates one of the actuator constraints indicated by the dashed line in the upper right plot.

### 4.4.3 Actuator Constraints

A second major factor contributing to feasibility is to ensure that the input constraints are satisfied such that no portion of the commanded trajectory requires control inputs outside of the range that the actuators are capable of providing. Formally, solving a trajectory optimization problem in the flat output space of a differentially-flat model requires mapping the constraints into the flat output space as well as the dynamics. Some work has focused on computationally estimating the feasible set in flat output space [14], however this set is generally a non-convex function of nonlinear inequalities and is a hard optimization problem unto itself.

Instead, we address this challenge during the time-allocation step of trajectory optimization, since the distribution of time along the trajectory largely determines the required accelerations and therefore the peaks in required actuator commands. First, we observe that in the limit as $\tau \to \infty$, the motion along the trajectory slows to a near-static state, which is known to be feasible for systems like hovering quadrotors. Therefore, we initialize our time-allocation optimization step with a conservatively large guess for initial segment times. Then, as the modified cost function is minimized, we compute the actuator commands algebraically during each iteration to verify that we remain within the feasible set. Optimization is terminated when either a local minimum is obtained or an actuator constraint becomes active. Figure 7 illustrates two different time allocations during the optimization of a sample trajectory for a quadrotor. One of these time allocations is safely within the feasible set, since it

commands thrusts barely above the nominal thrust required for hover, whereas the other time allocation is very aggressive and activates an actuator constraint for one of the motors.

Due to the non-convexity of the feasible set in flat output space, the optimization algorithm may encounter an actuator limit and terminate before converging to the optimal ratio of segment times (for example, one of the red or orange lines in Figure 4). To avoid this scenario, one strategy is to first optimize the ratio of segment times via gradient descent while ignoring actuator constraints, taking advantage of the fact that the optimal ratio of times is invariant to the total time, as noted in Section 4.4.1. Then once the optimal ratio of times is achieved, scale the *total* trajectory time in a separate univariate optimization, preserving the optimal ratio, until the modified cost function is minimized or an actuator constraint becomes active. This general concept applies to a variety of other differentially flat systems, such as robotic arms, which may have rate or torque constraints that could be satisfied by extending the duration of the trajectory to slow the motion.

# 5 State Estimation

To close the loop around the trajectory we must know the state of the vehicle. Since we are operating without the use of GPS, the state must be inferred from sensor readings and knowledge of rigid-body dynamics. The state estimation algorithm is responsible for taking as input sensor readings from the IMU and laser range finder and estimating the position and velocity of the vehicle to stabilize nominal trajectories. As described in Section 2.2, there are many existing state estimation algorithms that have been applied to autonomous vehicles. However, none of them are able to accurately perform localization using LIDAR measurements in a manner that is computationally efficient enough to run on a small computer that can be carried onboard a MAV. Our approach is to use a Gaussian particle filter (GPF)-based update step together with an EKF process model driven by the IMU to efficiently and compactly filter the sensor inputs. We also provide a method for estimating the noise parameters of the model without using ground-truth data.

## 5.1 State Estimation Problem Statement

Given a 3D occupancy map of the environment and a sequence of sensor measurements, $\mathbf{Z}_{0:t}$, up to the present time, we wish to estimate the current system state $\mathbf{x}_t = [\omega_b, \mathbf{v}_b, \mathbf{R}, \mathbf{\Delta}]_t$. We model the system as a rigid body and we neglect higher-order effects resulting from aerodynamics or other disturbances. We assume a set of inertial measurements consisting of 3-axis acceleration and 3-axis angular rate measurements, and exteroceptive measurements consisting of planar laser range scans.

## 5.2 IMU Process Model

Our state estimation algorithm uses an extended Kalman filter (EKF) to estimate a Gaussian distribution over system states. The EKF process model is

based on a discrete time, nonlinear discrete transition function:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \tag{21}$$

where $\mathbf{x}_t$ is the system state vector, $\mathbf{u}_t$ is the input vector to the system, and $\mathbf{w}_t$ is a random disturbance drawn from a normal distribution $N(\mathbf{0}, \mathbf{Q})$. The EKF tracks the state at time $t$ as a Gaussian distribution with mean $\mu_t$ and covariance $\mathbf{\Sigma}_t$. These first two moments are propagated forward according to:

$$\bar{\mu}_{t+1} = f(\mu_t, \mathbf{u}_t, \mathbf{0}) \tag{22}$$

$$\bar{\mathbf{\Sigma}}_{t+1} = \mathbf{A}_t \mathbf{\Sigma}_t \mathbf{A}_t^T + \mathbf{W}_t \mathbf{Q} \mathbf{W}_t^T \tag{23}$$

where $\bar{\mu}$ and $\bar{\mathbf{\Sigma}}$ denote predicted quantities before a measurement update has occurred, and $\mathbf{A}_t$ and $\mathbf{W}_t$ are the appropriate partial derivatives of $f$. Note that in this section our use of the symbols $\mathbf{A}$ and $\mathbf{Q}$ is distinct from their use in polynomial optimization in order to remain consistent with state estimation literature.

### 5.2.1 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 $\mathbf{R}$, we can write $\mathbf{R} = \hat{\mathbf{R}}\mathbf{R}(\chi)$ where $\hat{\mathbf{R}}$ is the estimated orientation and $\mathbf{R}(\chi) = e^{\chi^\wedge}$ is the error rotation matrix. $\chi \in \Re^3$ is the perturbation rotation about the body axes. We use the $^\wedge$ symbol to the right of a vector to denote the skew symmetric matrix formed as:

$$\chi^\wedge = \begin{bmatrix} 0 & -\chi_3 & \chi_2 \\ \chi_3 & 0 & -\chi_1 \\ -\chi_2 & \chi_1 & 0 \end{bmatrix} \tag{24}$$

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

In our expression for the true orientation, $\mathbf{R}(\chi)$ post multiplies $\hat{\mathbf{R}}$ which puts the perturbations in the body frame. Since the error is parameterized by $\chi$, the covariance can be tracked in a $3 \times 3$ matrix $\mathbf{\Sigma}_\chi$. The covariance can be thought of as cones of uncertainty surrounding the body frame axes defined by the columns of $\hat{\mathbf{R}}$. A sketch of this uncertainty is shown in Figure 8 for the covariance (in degrees):

$$\mathbf{\Sigma}_\chi = \begin{bmatrix} 3^2 & 0 & 0 \\ 0 & 5^2 & 0 \\ 0 & 0 & 15^2 \end{bmatrix} \tag{25}$$

This choice of coordinates for the filter error is desirable since fundamentally rigid body orientation, denoted mathematically as the special orthogonal group ($SO(3)$), 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 pose the constant threat of

creating negative eigenvalues, and thus causing the estimator to diverge. Furthermore, an efficient linearized measurement update as is commonly-used in Gaussian filters does not respect the constraints and thus does not map onto $SO(3)$. A renormalization scheme could be used after every update, but at any given time the representation can be arbitrarily poor [50].

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 $\chi$ in the estimator state vector $\mu$.

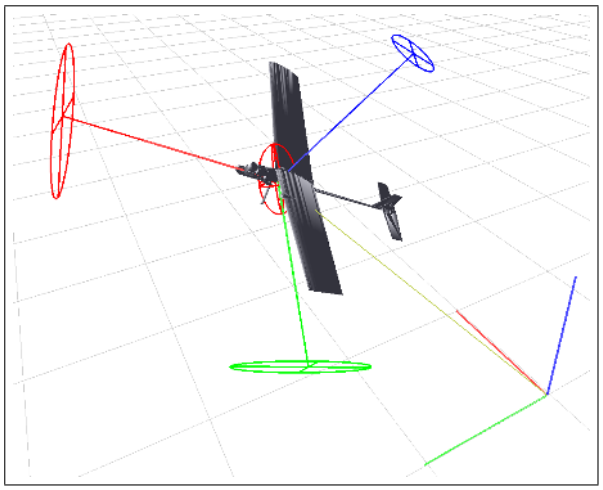


Figure 8: This figure shows the uncertainty representation in body axes. We see that high variance on the $z$-axis perturbation maps into large motions for the $x$- and $y$-bases.

### 5.2.2 Process Equations

The equations of motion for a rigid body are:

$$\dot{\omega}_b = \mathbf{J}^{-1}(-\omega_b \times \mathbf{J}\omega_b + \tau_b) \tag{26}$$
$$\dot{\mathbf{v}}_b = -\omega_b \times \mathbf{v}_b + \mathbf{R}^T\mathbf{g} + \mathbf{a}_b \tag{27}$$
$$\dot{\mathbf{R}} = \mathbf{R}\omega_b^{\wedge} \tag{28}$$
$$\dot{\mathbf{\Delta}} = \mathbf{R}\mathbf{v}_b, \tag{29}$$

where $\tau_b$ is the torque applied to the body and $\mathbf{a}_b$ is the acceleration in body coordinates. Since the IMU provides accurate measurements of $\omega_b$ and $\mathbf{a}_b$, we follow the commonly-used technique of omitting $\omega_b$ from the state, neglecting equation 26, and treating the IMU measurements as inputs to the filter using a standard linearized IMU update.

For the quantities used in equation 22 we have

$$\mathbf{x} = \begin{bmatrix} \mathbf{v}_b & \chi & \mathbf{\Delta} \end{bmatrix} \tag{30}$$
$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_{\text{gyro}} & \mathbf{u}_{\text{accel}} \end{bmatrix} \tag{31}$$
$$\mathbf{w} = \begin{bmatrix} \mathbf{w}_{\text{gyro}} & \mathbf{w}_{\text{accel}} \end{bmatrix} \tag{32}$$

Combining this state representation with equations 27-29 gives:

$$f_c(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) = \begin{bmatrix} \dot{\mathbf{v}}_b \\ \dot{\mathbf{R}} \\ \dot{\mathbf{\Delta}} \end{bmatrix} \tag{33}$$

$$= \begin{bmatrix} -\omega_b \times \mathbf{v}_b + \mathbf{R}^T\mathbf{g} + g\mathbf{u}_{\text{accel}} \\ \mathbf{R}\mathbf{u}_{\text{gyro}}^{\wedge} \\ \mathbf{R}\mathbf{v}_b \end{bmatrix}. \tag{34}$$

Taking the appropriate partial derivatives we get:

$$\frac{\partial \dot{\mathbf{v}}_b}{\partial \mathbf{x}} = \begin{bmatrix} -\omega_b^{\wedge} & (\mathbf{R}^T\mathbf{g})^{\wedge} & \mathbf{0} \end{bmatrix} \tag{35}$$

$$\frac{\partial \dot{\chi}}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{0} & -\omega_b^{\wedge} & \mathbf{0} \end{bmatrix} \tag{36}$$

$$\frac{\partial \dot{\mathbf{\Delta}}}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\mathbf{v}_b^{\wedge} & \mathbf{0} \end{bmatrix} \tag{37}$$

for a continuous dynamics linearization of:

$$\mathbf{A}_c = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} -\omega_b^{\wedge} & (\mathbf{R}^T\mathbf{g})^{\wedge} & \mathbf{0} \\ \mathbf{0} & -\omega_b^{\wedge} & \mathbf{0} \\ \mathbf{R} & -\mathbf{R}\mathbf{v}_b^{\wedge} & \mathbf{0} \end{bmatrix} \tag{38}$$

and for the input vector we have:

$$\frac{\partial \dot{\mathbf{v}}_b}{\partial \mathbf{u}} = \begin{bmatrix} \mathbf{v}_b^{\wedge} & g\mathbf{I} \end{bmatrix} \tag{39}$$

$$\frac{\partial \dot{\chi}}{\partial \mathbf{u}} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \tag{40}$$

$$\frac{\partial \dot{\mathbf{\Delta}}}{\partial \mathbf{u}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{41}$$

$$\mathbf{W}_c = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \dot{\mathbf{v}}_b}{\partial \mathbf{u}} \\ \frac{\partial \dot{\chi}}{\partial \mathbf{u}} \\ \frac{\partial \dot{\mathbf{\Delta}}}{\partial \mathbf{u}} \end{bmatrix}. \tag{42}$$

While more sophisticated approximations could be used, we construct the discrete quantities for the filter $f$, $\mathbf{A}_t$, and $\mathbf{W}_t$ using Euler integration:

$$f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{0}) = \mathbf{x}_t + f_c(\mathbf{x}_t, \mathbf{u}_t, \mathbf{0})dt \tag{43}$$

$$\mathbf{A}_t = \mathbf{I} + \mathbf{A}_c dt \tag{44}$$

$$\mathbf{W}_t = \mathbf{W}_c dt. \tag{45}$$

We integrate the attitude separately as

$$\mathbf{R}_{t+1} = \mathbf{R}_t \mathbf{R}(\mathbf{u}_{\text{gyro}}^{\wedge}). \tag{46}$$

## 5.3 Laser 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. Using such sensors directly in an EKF requires the extraction and correspondence of features such as corners, and line segments from the sensor measurements; an error prone process that limits the applicability of the algorithms to environments with specific structure [23]. In contrast Monte-Carlo techniques are widely used in laser-based localization algorithms because they allow for the LIDAR range measurement model to be used directly in the measurement function [49].

While particle filters are efficient enough for effective use in localizing a 2D mobile robot, they require too many particles to be used for the estimation of a 3D MAV. Fortunately, we can obtain the best aspects of both algorithms, and a significant speedup can be realized by employing a hybrid filter that uses an IMU-driven EKF process model with pseudo-measurements computed from Gaussian particle filter (GPF) laser measurement updates [32].

### 5.3.1 Gaussian Particle Filters

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

$$\mu_{t+1} = \frac{\sum_{j=1}^M \mathbf{w}_t^{(j)} \mathbf{x}_t^{(j)}}{\mathbf{w}_t^{(j)}} \tag{47}$$

$$\mathbf{\Sigma}_{t+1} = \frac{\sum_{j=1}^M \mathbf{w}_t^{(j)} (\mathbf{x}_t^{(j)} - \mu_{t+1})(\mathbf{x}_t^{(j)} - \mu_{t+1})^T}{\mathbf{w}_t^{(j)}}. \tag{48}$$

Assuming the underlying system is linear-Gaussian, the filter is shown to approximate the true distribution arbitrarily well with a large number of samples. The GPF filter differs from a standard particle filter by maintaining a unimodal Gaussian distribution over the posterior state instead of the arbitrary (possibly multi-modal) distribution represented by the set of particles in a conventional particle filter.

A straightforward implementation of the GPF for state estimation using a LIDAR 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. Thus, a particle process model adds significant computational burden and sampling error, without significantly improving the estimate of the posterior density.

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 $\mathbf{\Delta}$ and $\chi$ in our formulation. In fact, most of the orientation information in the measurement exists in the plane of the LIDAR, corresponding to $\chi_z$.

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, $N(\bar{\mu}, \bar{\mathbf{\Sigma}})$. To address the second issue 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 the result to compute a pseudo-measurement which is then used to update the full state.

### 5.3.2 Partitioned State Update

The state is partitioned as

$$\mathbf{x}_t = \begin{bmatrix} \mathbf{x}_t^m & \mathbf{x}_t^p \end{bmatrix}, \tag{49}$$

where $\mathbf{x}_t^m \in \Re^k$ is the part of the state that affects the measurement, and $\mathbf{x}_t^p \in \Re^{n-k}$ is independent from the measurement. As an example of this partitioning, the linear and angular velocities of the vehicle do not affect the ranges measured by the LIDAR sensor. Therefore, these state variables can be considered independent from the measurement. More formally we assume our measurement function has the form

$$\mathbf{z}_t = h(\mathbf{x}_t^m, \mathbf{v}_t), \tag{50}$$

permitting the independence factorization

$$P(\mathbf{z}_t|\mathbf{x}_t^p, \mathbf{x}^m) = P(\mathbf{z}|\mathbf{x}^m). \tag{51}$$

We can similarly partition the covariance

$$\bar{\mathbf{\Sigma}}_t = \begin{bmatrix} \bar{\mathbf{\Sigma}}_t^{(m^2)} & \bar{\mathbf{\Sigma}}_t^{(mp)} \\ \bar{\mathbf{\Sigma}}_t^{(pm)} & \bar{\mathbf{\Sigma}}_t^{(p^2)} \end{bmatrix}. \tag{52}$$

To perform the measurement update we draw samples $\{\mathbf{x}_t^{m(j)}\}_{j=1}^M$ from $N(\bar{\mu}_t^m, \bar{\mathbf{\Sigma}}_t^m)$. The samples are weighted with the measurement function in equation 51. From these weighted samples we can compute an update for $P(\mathbf{x}_t^m|\mathbf{z}_{0:t})$ using the conventional GPF weighted mean and covariance as in equations 47 and 48. 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 $\mathbf{z}_t$, the covariance of the measurement $\mathbf{R}$, and the observation matrix $\mathbf{C}$. First, we set $\mathbf{C}$ as a selector matrix for the measurement part of the state

$$\mathbf{C} = \begin{bmatrix} \mathbf{I}_k & \mathbf{0}_{n-k} \end{bmatrix}. \tag{53}$$

A measurement update on $\mathbf{x}^m$ would proceed as

$$\mathbf{K}^m = \bar{\mathbf{\Sigma}}_t^m (\mathbf{C}^m)^T (\mathbf{C}^m \bar{\mathbf{\Sigma}}_t (\mathbf{C}^m)^T + \mathbf{R})^{-1} \tag{54}$$

$$\mu_t^m = \bar{\mu}_t^m + \mathbf{K}^m (\mathbf{z}_t - \mathbf{C}^m \bar{\mu}_t^m) \tag{55}$$

$$\mathbf{\Sigma}_t^m = (\mathbf{I} - \mathbf{K}^m \mathbf{C}^m) \bar{\mathbf{\Sigma}}_t^m. \tag{56}$$

Plugging in the identity matrix for $\mathbf{C}^m$, the above equations can be solved for $\mathbf{R}_t$

$$\mathbf{\Sigma}_t^m = \bar{\mathbf{\Sigma}}_t^m - \bar{\mathbf{\Sigma}}_t^m (C^m)^T (C^m \bar{\mathbf{\Sigma}}_t (C^m)^T + R_t)^{-1} \bar{\mathbf{\Sigma}}_t^m \tag{57}$$

$$\mathbf{R}_t = (\bar{\mathbf{\Sigma}}_t^{m^{-1}} - \bar{\mathbf{\Sigma}}_t^{m^{-1}} \Sigma_t^m \bar{\mathbf{\Sigma}}_t^{m^{-1}})^{-1} - \bar{\mathbf{\Sigma}}_t^m \tag{58}$$

$$= (\Sigma_t^{m^{-1}} - \bar{\mathbf{\Sigma}}_t^{m^{-1}})^{-1}, \tag{59}$$

where we make use of the matrix inversion lemma between equations 58 and 59.

Using $\mathbf{R}_t$ we can now solve for the Kalman gain that would have produced the same change between our prior and posterior covariance using equation 54 and then recover the actual measurement that would have produced the same change in the mean of prior vs. posterior distributions:

$$\mathbf{z}_t = \mathbf{K}^{m^{-1}} (\mu_t^m - \bar{\mu}_t^m) + \bar{\mu}_t^m. \tag{60}$$

A Kalman gain for the entire state can then be computed using $\mathbf{R}_t$ and $\mathbf{z}_t$, and a standard Kalman measurement update performed.

The posterior distribution quantities $\mu_t^{m^{-1}}$ and $\mathbf{\Sigma}_t^{m^{-1}}$ are readily available from the GPF measurement update on the measurement part of the state vector. Naïvely one might use the Gaussian prior from which the samples were drawn to evaluate equations 59 and 60. However, the quantities we care about, $\mathbf{R}_t$ and $\mathbf{z}_t$, 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 $\{\mathbf{x}_t^{m(j)}\}_{j=1}^M$ 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 $\bar{\mathbf{\Sigma}}_t^m$. In practice, this substitution makes an enormous difference, particularly with low numbers of particles (which is highly desirable in a real-time application).

The solutions for $\mathbf{R}_t$ and $\mathbf{z}_t$ hinge on the invertibility of $\mathbf{C}^m$ which is a proxy for the invertibility of our measurement function $h$ in equation 50 with respect to $\mathbf{x}_t^m$. 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 does not actually contain information about some piece of $\mathbf{x}_t^m$) then the $\mathbf{R}_t$ matrix may not be positive-definite, leading to a filter divergence. Thus it is necessary in practice to perform an eigenvalue decomposition on $\mathbf{R}_t$ 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 errors.

### 5.3.3 LIDAR Likelihood Computation

The LIDAR likelihood evaluation proceeds according to standard techniques used in 2D localization. We blur the a 3D occupancy map stored as an OctoMap [52] using a Gaussian kernel around occupied cells. To perform particle measurement updates we project the current scan into the map using the sampled particle state, and sum the log-likelihood of the relevant cells before exponentiating to obtain a probability with which to weight the particles.

An interesting question is the appropriate partitioning of the state vector for the updates described in the previous section. The use of planar LIDARs

to localize in the horizontal plane is ubiquitous, suggesting that when working in 3D, laser range scans should at least contain information about $x$, $y$ and $\chi_z$ (orientation about the yaw axis of the vehicle). In general, a planar slice of a 3D environment may contain some information about the full orientation. However, populating the 6D pose space parameterized by $\chi$ and $\mathbf{\Delta}$ 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 measurements. We investigate different choices for $\mathbf{x}^m$ in our experiments in section 5.5.

## 5.4 Identifying the Process Noise Parameters

Accuracy and precision in state estimation are very important during aggressive flight, since our MAVs fly within centimeters of obstacles and small deviations from the desired trajectory could result in collision. In this section, we investigate a strategy for identifying process noise parameters in order to maximize the performance of our state estimate.

Due to the symmetry of the inertial sensors in the IMU, we assume the process noise covariance $\mathbf{Q}$ is a diagonal matrix populated as

$$\mathbf{Q} = \begin{bmatrix} q_{\text{gyro}}\mathbf{I}_3 & 0 \\ 0 & q_{\text{accel}}\mathbf{I}_3 \end{bmatrix}, \tag{61}$$

where $q_{\text{accel}}$ and $q_{\text{gyro}}$ are the parameters we wish to identify. Two issues make it difficult to find these values. First, the way the noise projects onto the state changes with the time-varying $\mathbf{W}_t$ matrix such that the $\mathbf{Q}$ matrix cannot be recovered in closed form simply by summing the outer product of sampled error. More importantly we cannot depend on the availability of ground-truth measurements of the measured quantities, since even accurate positioning systems do not directly measure acceleration and angular rate. Further, the behavior of the sensor may be different under actual flight conditions due to vibration and loading effects and thus the values obtained in a static test may not hold.

Nonetheless it is desirable 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 5.3.3, it is important that an accurate covariance of the state estimate be maintained when sensor data is sparse or absent, such that the state estimate can be properly distributed to incorporate measurements when they become available.

While we do not have access to ground-truth acceleration and angular rate with which to estimate the noise parameters, we can post-process data using a Kalman smoothing algorithm to obtain a state history $\mathbf{X} = [\hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_T]$ with the error associated with each smoothed state estimate given by

$$\mathbf{\Gamma}_t = E\left[(\hat{\mathbf{x}}_t - \mathbf{x}_t)(\hat{\mathbf{x}}_t - \mathbf{x}_t)^T\right] \tag{62}$$

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. This

approach works because the IMU process equations are neutrally stable and thus the perturbing noise results in unbounded growth in covariance without position updates. The error on the smoothed estimate (with position updates), on the other hand, must be bounded (even if the smoothing occurs with incorrect noise parameters) since the system is observable. 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 predictions [9] and the parameters will work with intermittent measurement functions as may be the case for laser localization in sparse environments.

Using the linearized dynamics from the EKF we can project the filter covariance forward $N$ steps by repeatedly applying equations 22 and 23. Neglecting the error on the smoothed estimate, we obtain the expression:

$$E\left[(\hat{\mathbf{x}}_{t+N} - \hat{\mathbf{x}}_t)(\hat{\mathbf{x}}_{t+N} - \hat{\mathbf{x}}_t))^T\right] = \mathbf{\Sigma}_{t,N} \tag{63}$$

$$= \sum_{i=0}^{N-1} \mathbf{G}_{t+i,N}\mathbf{Q}\mathbf{G}_{t+i,N}^T \tag{64}$$

where $\mathbf{G}_{t,N} = \prod_{j=t+1}^{t+N-1} \mathbf{A}_j \mathbf{W}_t$. This is an important quantity for our noise identification algorithm because it maps the noise at each time step onto the state vector at time $t+N$. We can see that for identifying characteristics of the process noise, $\mathbf{A}_t$ must be neutrally stable and $\mathbf{W}_t$ must have full column rank. If $\mathbf{A}_t$ is highly unstable, the $\mathbf{\Sigma}_{t,N}$ will be overly sensitive to the noise values $\mathbf{w}_i$ for small $i$, whereas if $\mathbf{A}_t$ is highly stable, $\mathbf{\Sigma}_{t,N}$ 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 $\mathbf{\Sigma}_t >> \mathbf{\Gamma}_t$. We can then divide up the dataset $\mathbf{X}$ 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 $\mathbf{y}_i = \mathbf{x}_{t_{\text{end}}} - \mathbf{x}_{t_{\text{begin}}}$ drawn from distributions $N(\mathbf{0}, \mathbf{\Sigma}_{t_{\text{begin}},N}) = P(\mathbf{x}_{t_{\text{end}}}|\mathbf{x}_{t_{\text{begin}}})$. We have a joint likelihood function for our data given the parameters of $\mathbf{Q}$ as:

$$P(\mathbf{Y}|\mathbf{x}_0, \mathbf{Q}) = \prod_{i=0}^{M-1} P(\mathbf{x}_{iN+N-1}|\mathbf{x}_{iN}, \mathbf{Q}). \tag{65}$$

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

$$l(\mathbf{Y}|\mathbf{x}_0, \mathbf{Q}) = -\frac{1}{2}\sum_{i=0}^{M-1} \log|\mathbf{\Sigma}_i| + \mathbf{y}_i^T \mathbf{\Sigma}_i \mathbf{y}_i. \tag{66}$$

From an intuitive standpoint we are optimizing for the $\mathbf{q}$ parameters that would produce the observed drift away from the smoothed estimate given by the samples $\mathbf{y}_i$.

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 $q_{\text{gyro}}$ and $q_{\text{accel}}$. These new values are then used to obtain the Kalman smoothed trajectory, and the process is repeated until convergence.

| Source | Gyro Noise (degrees/s) | Accelerometer Noise (g) |
|---|---|---|
| Vicon Optimization | 0.35 | 0.0042 |
| GPS Optimization | 0.34 | 0.0182 |
| Manufacturer | 0.2 | 0.005 |

Table 4: Noise parameter values.

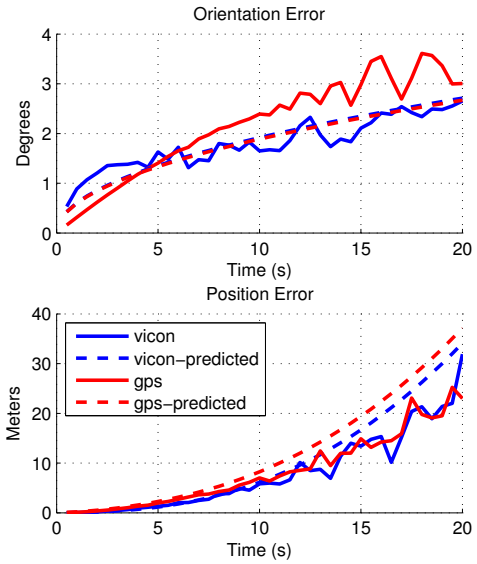



Figure 9: This figure shows the predicted normed error and the actual normed deviation from the smoothed estimates as a function of look-ahead time for the optimization run on both the Vicon and GPS datasets. With the optimized values we can accurately predict uncertainty for both estimation and planning purposes.

To identify the noise parameters of the IMU we flew our experimental vehicle (described in Section 7.4) outdoors with a low cost uBlox GPS unit. We also collected a dataset in a high accuracy Vicon indoor motion capture system. Optimized noise parameters for a look-ahead time of 20 seconds are shown in Table 4 with the manufacturer specified values for comparison.

The optimization on the Vicon dataset converges quickly and consistently. However, when the optimization is performed on the GPS dataset the optimization is more sensitive to initial conditions and window size. The Vicon system measures attitude directly, thus the smoothed attitude estimate is dominated by the actual measurement. With the GPS dataset, attitude must be inferred from position updates which means the attitude estimate will be more strongly correlated with the IMU noise, thus making it more difficult find the underlying noise parameter. Additionally, the GPS measurements are subject to time-varying bias which is not modeled in our filter. Nonetheless, the optimization for Vicon and GPS converge to nearly identical values for the gyro noise at at 20 second window. The relative sensitivity to the window size for the GPS optimization can be seen in Figure 10.

The noise parameters in Table 4 were used to generate the predicted error

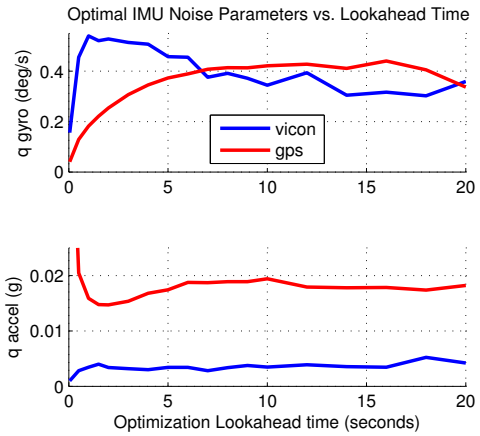



Figure 10: This figure shows values for $q_{\text{gyro}}$ and $q_{\text{accel}}$ obtained by optimizing equation 66 for different look-ahead times (values of $N$ scaled by sampling frequency) for both GPS and Vicon. For small time the optimal noise parameters obtained with GPS are dominated by the error in the smoothed estimates, $\mathbf{\Gamma}_t$, but we see for large $N$ consistent values are reached. The Vicon dataset is less susceptible to this issue. It is interesting to note that as look-ahead time increases fewer "samples" are available from a dataset of fixed size, and thus the computed noise values have higher variance, implying some optimal look-ahead window to identify the parameters.

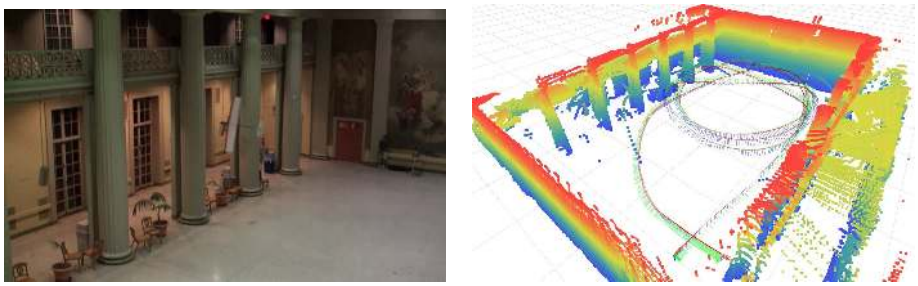
lines in Figure 9. We can see that the predicted error for GPS and Vicon are very close for orientation as we would expect from the tabular values. In position, the deviation is also small, which is surprising given the large difference in optimized accelerometer noise values. The reason for this observation is that the position uncertainty is largely a function of angular uncertainty resulting in the gravity vector being misinterpreted as lateral acceleration. This highlights another difficulty in separating the relative effects of the noise parameters during aggressive flight.

## 5.5 State Estimation Experimental Results

For each experimental test of our state estimation algorithm, we first collect a 3D map of the test environment using a pair of planar LIDARs mounted orthogonally on a wheeled tripod. We push the tripod around manually to obtain laser scans from every region of the environment. Then we run a SLAM algorithm to estimate the poses of the horizontal LIDAR, and project the corresponding scans from the vertically-mounted LIDAR to fill out the 3D map. This procedure takes several minutes and is the only preparation needed to run our estimation algorithm in a new environment.

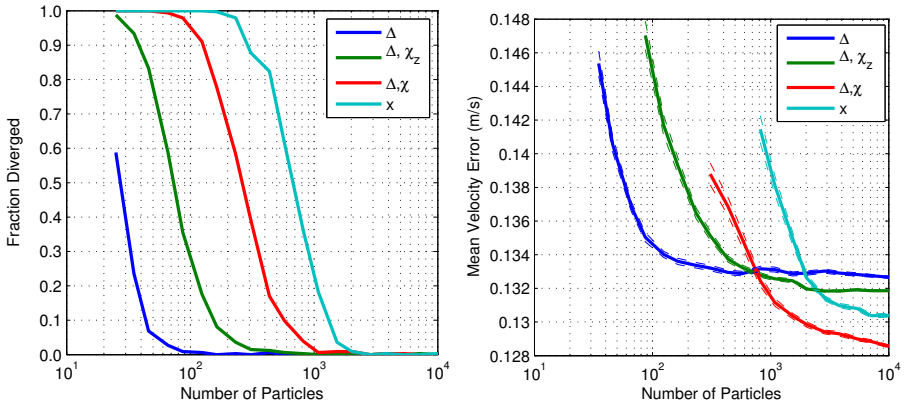
We conducted a number of manually flown flight tests in the indoor environment shown in Figure 11(a). The accuracy of our state estimates are validated qualitatively by looking at the accurate reconstruction of the 3D environment by re-projecting the laser points using our state estimates. One such 3D point cloud is shown in Figure 11(b). A video of these experiments is available at: `http://groups.csail.mit.edu/rrg/icra12-agile-flight`.

To quantify the error of the state estimator, we aggressively maneuvered the



(a) (b)

Figure 11: 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 airborne. 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.




(a) (b)

Figure 12: This figure shows the percentage of trials where the filter diverged (a) and the mean velocity error verses the number of particles used in the GPF (b) 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 naïve implementation where the full state is used in the measurement and thus a standard GPF update performed, we require 2000 particles to get similar performance to a measurement update in $\mathbf{\Delta}$ using only 100 particles. Thus our algorithm yields an effective 20x speedup.

sensors in a Vicon motion capture environment. While the motion of the sensors will certainly be very different when the vehicle is flying, the data allows us to evaluate our algorithms with a ground-truth comparison. These ground-truth state estimates allow us to evaluate the properties of our state estimation algorithm. Results for different numbers of particles and different partitions of the state vector are summarized in Figure 12. We can see that by not partitioning the state and performing standard GPF updates we incur significant computational cost in terms of number of particles needed to achieve the same level of accuracy. This increase in the number of particles is to be expected given that we are using particles to capture the same correlations that are well captured analytically by the Kalman pseudo-measurement update.

The experiments demonstrate the ability of our algorithm to maintain accurate state estimates during fast motion, with linear velocities up to 11 m/s, and angular rates of up to 360 degrees per second. Furthermore, during the closed-loop flight tests described in Sections 6.1 and 7.5, our algorithm provided a consistently accurate state estimate and never diverged. While a naïve 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 naïve GPF implementation would not be able to run in real time onboard the vehicle given the computation power available.

# 6 Quadrotor Trajectory Planning and Experimental Results

In Section 4, we developed a procedure for efficiently computing polynomial trajectories. In this section, we apply this procedure to solve the planning problem for quadrotors. As is common in the literature, we model the quadrotor as a rigid body. We assume that certain higher-order effects such as aerodynamic drag are negligible for our purposes. The equations governing the motion of a rigid body were given in the context of state estimation (26-29). In the case of a multi-copter helicopter, we exert moments by commanding different thrusts from opposing rotors. In a simplified representation of a conventional helicopter, we can exert a moment on the body through the use of the cyclic control. In either case, we assume that our low-level inputs can be selected in order to achieve the desired roll, pitch, and yaw moments comprising the $\tau_b$ vector. The major restriction we must make is that the net thrust of a helicopter is always aligned with the rotor axis of rotation. Therefore, we can replace $\mathbf{a}_b$ in equation (27) with $(f/m)\mathbf{b}_z$, where $f$ is the scalar sum of thrusts provided by the rotors and $m$ is the vehicle mass. This set of equations has been used in the literature to describe the simplified dynamics of quadrotors [36] and conventional helicopters [31, 17].

After restricting the direction of thrust to align with the $\mathbf{b}_z$-direction, these equations have been shown to be differentially flat in reference to quadrotors. Mellinger and Kumar provide a detailed description of the mapping from the states and inputs of the vehicle to the flat output space, which we will not repeat here [38]. We follow Mellinger and Kumar in expressing a quadrotor trajectory segment as a set of four independent polynomials that describe the evolution

of the four flat output variables through time, namely positions $\Delta_x(t)$, $\Delta_y(t)$, $\Delta_z(t)$, and the yaw angle $\psi(t)$.

The nonlinear controller employed to follow differentiable trajectories was developed in [36], and includes the methods of computing desired orientations and angular velocities from $\mathbf{\Delta}(t)$ and $\psi(t)$ and their derivatives. In the following equations we use the subscript $d$ to refer to the desired position ($\mathbf{\Delta}_d$), orientation ($\mathbf{R}_d$), and angular velocity ($\omega_d$) as specified by the trajectory.

$$f = (-k_p \mathbf{e}_p - k_d \mathbf{e}_d + mg\mathbf{w}_z + m\ddot{\mathbf{\Delta}}_d) \cdot \mathbf{b}_z \tag{67}$$

$$\begin{aligned} \tau_b = &- k_R \mathbf{e}_R - k_\omega \mathbf{e}_\omega + \omega_b \times \mathbf{J}\omega_b \\ &- \mathbf{J}(\omega_b^\wedge \mathbf{R}^T \mathbf{R}_d \omega_d - \mathbf{R}^T \mathbf{R}_d \dot{\omega}_d) \end{aligned} \tag{68}$$

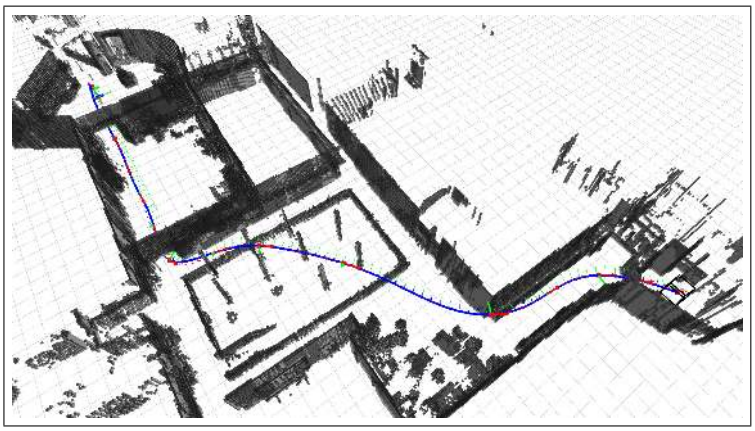
where $\mathbf{e}_p$, $\mathbf{e}_d$, $\mathbf{e}_R$, and $\mathbf{e}_\omega$ are the error vectors in position, velocity, orientation and angular velocity, and $k_p$, $k_d$, $k_R$, and $k_\omega$ are associated control gains. The mass of the vehicle is $m$.

These equations include a combination of feed-forward terms, representing the open-loop commands that would be needed to execute a given trajectory in the absence of disturbances, and feedback commands that stabilize the vehicle during its execution of the trajectory. Since the desired trajectory and its derivatives are sufficient to compute the states and control inputs at every point along the path in closed form (equations 67-68), these quantities effectively serve as a simulation of the vehicle's motion in the absence of disturbances. This is the powerful capability enabled by differential flatness and the chosen trajectory parameterization that eliminates the need for iterated numerical integration of equations of motion, or a search over the space of inputs during planning.

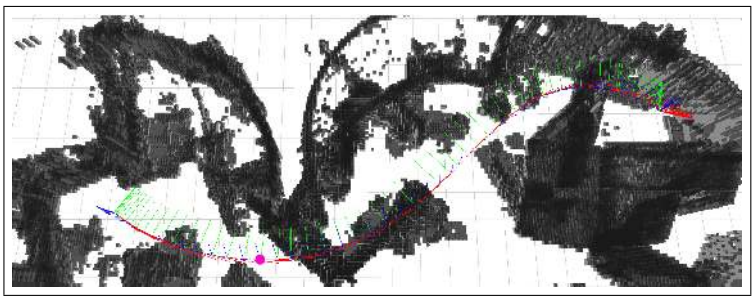
## 6.1 Quadrotor Experimental Flight Results

We tested the performance of our quadrotor trajectory planning strategy on challenging real-world planning problems by planning and flying trajectories through complex indoor lab spaces in the MIT Stata Center. The environment used for these tests has curved non-vertical walls, interior columns, barriers aligned at oblique angles and hanging lights, representing a challenging arrangement of obstacles for our planning algorithm. Interestingly, the complexity of the environment and density of obstacles actually makes localization easier due to the unique pattern of objects in each laser scan as compared to a large empty environment. An OctoMap representation of the lab was generated using a pair of planar laser range finders and each occupied cell was dilated with a radius of 0.65 m to leave room for the 0.35 m radius of the vehicle and a minimal allowance for error in estimation and control. Table 5 gives tracking errors of the controller incurred during experiments. These errors are sufficiently small to navigate within one vehicle-width of obstacles, which was required during significant portions of both trajectories.

The trajectories returned by our algorithm are shown in Figure 13, and they both exhibit approximately 2 m of altitude variation in order to fly through doorways and navigate over tall shelves and dividing walls. These trajectories were generated in several seconds on a ground-based Intel Core 2 Duo laptop for the purposes of visualization prior to execution, however the planning procedure is efficient enough to run easily on currently available small form-factor computers that could be carried onboard a quadrotor. The quadrotor used for these



(a)



(b)

Figure 13: Automatically generated trajectories navigating atrium (a) and laboratory (b) environments in the MIT Stata Center at up to 8 m/s. Due to the tight spacing of obstacles, these environments are particularly challenging for motion planning of high-speed vehicles.

| Environment | Mean Position Error (m) | Mean Velocity Error (m/s) |
|---|---|---|
| Atrium | 0.3631 | 0.3363 |
| Laboratory | 0.3054 | 0.2717 |

Table 5: Position and velocity tracking errors incurred during execution of the quadrotor trajectories through an atrium 13(a) and a laboratory 13(b). Since we do not have access to ground-truth state estimation, our tracking errors are reported with respect to the onboard state estimate.

experiments was an Asctec Pelican, equipped with a Hokuyo UTM-30LX laser rangefinder, a Microstrain 3DM-GX3-25 IMU, and a 2.0GHz Intel Atom based flight computer. Figure 14 shows onboard video frames taken while executing these trajectories at speeds up to 8 m/s. Video of these trajectories and flights is available at: `http://groups.csail.mit.edu/rrg/quad_polynomial_trajectory_planning`.

(a) (b)

Figure 14: Onboard video frames from aggressive quadrotor flight up to 8 m/s.

# 7 Fixed-Wing Trajectory Planning and Experimental Results

In Section 4, we saw how it was possible to efficiently plan polynomial trajectories for systems in which the three position coordinates can be controlled independently. For example, a quadrotor can modulate its pitch angle to generate motions forward and backward, while simultaneously modulating roll to move left or right and adjusting the total thrust to move up and down. In these cases, a suitable trajectory representation is to describe the motion of each flat output with its own polynomial trajectory.

In principle, we could also use this trajectory representation to describe the motions of a fixed-wing aircraft. As we will see in this section, the differentially flat model for the fixed-wing vehicle uses the three components of its global position vector as the flat outputs. However, the key difference between this model and the quadrotor model is that the fixed-wing cannot independently control these three position coordinates. Rather, its motions are characterized as Dubins-type motions, which means qualitatively that it is capable of traveling along trajectories consisting of lines and arcs of bounded curvature, but cannot translate sideways or vertically like a quadrotor. Therefore, describing the motion of the three position coordinates using independent polynomials is not ideal for this type of system.

Instead, we will develop a special trajectory formulation for fixed-wing aircraft and vehicles with similar maneuvering capabilities, called Dubins-Polynomial trajectories. These trajectories are defined as a sequence of straight lines and arcs of constant curvature, combined with a piecewise-polynomial transverse offset from those underlying arcs. The transverse dynamics of the fixed-wing aircraft correspond roughly to roll maneuvers, so we can now use polynomial optimization to minimize quantities of interest like roll rate and roll acceleration, which are important for generating graceful trajectories. We will first describe the vehicle model, then develop this Dubins-Polynomial trajectory formulation, and finally we will describe simulation and experimental results for the fixed-wing platform.

## 7.1 Fixed-Wing Coordinated Flight Model

The planning and control algorithms for the fixed-wing vehicle are based on a coordinated flight model due to Hauser and Hindman [21]. In order to plan trajectories for fixed-wing aircraft, we must understand its dynamic capabilities and the flat output space associated with its differentially flat representation.

Coordinated flight is defined as a flight condition in which the body velocity of the vehicle is contained within the longitudinal plane, hence $v_{b_y} = 0$, where $v_{b_y}$ is the component of velocity directed along the $\mathbf{b}_y$ axis aligned with the axis of the wing. 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 $\mathbf{R}_v$. The unit vector of the first column aligns with the velocity such that $v_{v_y} = v_{v_z} = 0$. To map velocity and acceleration back into the world frame, we have $\dot{\mathbf{\Delta}} = \mathbf{R}_v \mathbf{v}_v$ and $\ddot{\mathbf{\Delta}} = \mathbf{g} + \mathbf{R}_v \mathbf{a}_v$.

To maintain coordinated flight, the second and third components of angular velocity (pitch and yaw rates) are constrained to be:

$$\omega_{v_y} = -(a_{v_z} + g_{v_z})/V \tag{69}$$

$$\omega_{v_z} = g_{v_2}/V, \tag{70}$$

where $V = ||\dot{\Delta}||$ and $g_{v_y}$ and $g_{v_z}$ represent the components of the gravity vector projected along the $\mathbf{v}_y$ and $\mathbf{v}_z$ directions, respectively.

In these equations, $\mathbf{a}_v$ is the acceleration of the body expressed in the velocity frame. Components $a_{v_x}$ and $a_{v_z}$ represent the axial and normal accelerations of the vehicle, which are determined by the forward thrust of the propellers and the lift force provided by the wing. In the differentially flat model, derivatives of these quantities will serve as two of the inputs to the system.

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 differentially flat with inputs $(\omega_{v_x}, \dot{a}_{v_x}, \dot{a}_{v_z})$ and outputs $\mathbf{\Delta}$. The mapping is:

$$\begin{bmatrix} \dot{a}_{v_x} \\ \omega_{v_x} \\ \dot{a}_{v_z} \end{bmatrix} = \begin{bmatrix} -\omega_{v_y} a_{v_z} \\ \omega_{v_z} a_{v_x}/a_{v_z} \\ \omega_{v_y} a_{v_x} \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1/a_{v_z} & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{R}^T \dddot{\mathbf{\Delta}} \tag{71}$$

Equation (71) is powerful, in that it gives the necessary inputs for the simplified coordinated flight model to achieve an arbitrary third derivative on the position of the airplane.

The differentially flat representation also 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), [21]):

$$\dddot{\mathbf{\Delta}} = \dddot{\rho} + k_2 \ddot{\mathbf{e}} + k_1 \dot{\mathbf{e}} + k_0 \mathbf{e} \tag{72}$$

where $\rho(t)$ is the path to be followed, $\mathbf{e}(t) = \rho(t) - \mathbf{\Delta}(t)$, and $k_0$, $k_1$ and $k_2$ are feedback gains. The disadvantage of this approach is that the aircraft will modulate tangential acceleration (thrust) to correct axial errors along the path

(i.e., if the vehicle gets ahead or behind its target point). However, fixed-wing aircraft are designed for a small range of safe cruising velocities and it is often assumed that the thrust will be fixed to maintain a particular safe airspeed. In the worst case, if the aircraft slowed down too aggressively to track a reference state, it could lose the necessary lift to stay aloft or otherwise exit the safe operating envelope.

Therefore, we opt for an alternative strategy, which is to alter the state representation to include a path parameter $s^{(3)}$ in place of $\dot{a}_{v_x}$, which will be integrated along with the rest of the state. Following equation (11) in [21] we have:

$$\begin{aligned}\mathbf{M}\begin{bmatrix} s^{(3)} \\ \omega_{v_x} \\ \dot{a}_{v_z} \end{bmatrix} &= \begin{bmatrix} a_{v_z}\omega_{v_y} + \dot{a}_{v_x} \\ a_{v_x}\omega_{v_z} \\ -a_{v_x}\omega_{v_y} \end{bmatrix} \\ &\quad - \mathbf{R}^T\left(3\rho''(s)\ddot{s}\dot{s} + \rho'''(s)\dot{s}^3 + k_2\ddot{\mathbf{e}} + k_1\dot{\mathbf{e}} + k_0\mathbf{e}\right),\end{aligned} \tag{73}$$

where,

$$\mathbf{M} = \begin{bmatrix} \vdots & 0 & 0 \\ \mathbf{R}^T\rho'(s) & a_{v_z} & 0 \\ \vdots & 0 & -1 \end{bmatrix} \tag{74}$$

Introduction of the path parameter $s$ enables dynamic inversion of the trajectory when the throttle is effectively fixed, which would be impossible without it. 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 $\mathbf{M}$ be invertible, which requires that the first column of $\mathbf{R}_v \propto \dot{\mathbf{\Delta}}$ not be orthogonal to $\rho'$. Intuitively, if the plane is flying perpendicular to the desired path, no control authority on the error dynamics is available through $\dddot{s}$. The singularity in $\mathbf{M}$ is also not of practical concern so long as the controller is initialized from reasonable conditions. In Appendix B, we show how to translate the variables of the differentially flat model into control surface deflections for the low-level controller onboard the aircraft.

## 7.2 Dubins-Polynomial Trajectory Representation for Fixed-Wing Aircraft

In the preceding discussion of the differentially flat fixed-wing representation, we have seen how it is possible to obtain the necessary inputs to the vehicle directly from the third derivative of the desired position trajectory. The next challenge is to define a trajectory representation that is capable of expressing paths that we would like our aircraft to follow. Additionally, this representation should be differentiable so that we can easily compute the quantities needed to perform control. Finally, we would like the trajectory representation to enforce continuity of derivatives up to third order so that the resulting motions of the aircraft will be smooth, and in particular, have continuous roll rate.

Except for takeoff and landing maneuvers, we restrict the motion of the vehicle to the horizontal plane at constant altitude and fixed speed. For planning purposes we wish to be able to generate trajectories from an initial position, $(\Delta_x, \Delta_y)$, yaw angle $(\psi)$, roll angle $(\phi)$, and roll rate $(\dot{\phi})$, to a final configuration of those variables.

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

$$\phi = \tan^{-1}\left(\frac{V^2\kappa}{g}\right), \tag{75}$$

is a special case of equation 71, where $\kappa$ is the curvature of the path (inverse radius, positive turning to the left or counterclockwise (CCW)).

The most commonly used approach for planar path planning for fixed-wing vehicles is to use Dubins curves [34]. Dubins curves represent the optimal (shortest distance) path between two position and orientation, $(\Delta_x, \Delta_y, \psi)$, configurations respecting a minimum turning radius. The path between any two configurations will be made up of 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 is selected. It is also possible to analytically determine which word will be shortest based on an algebraic partitioning of $SE(2)$, 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 75 that discontinuity in curvature will translate to discontinuity in roll angle. This is clearly kinodynamically infeasible for a fixed-wing vehicle. 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) [34]. However, clothoid paths would still be discontinuous in roll rate. By differentiating equation 75, we can see that roll rate $\dot{\phi}$ is proportional to the derivative of curvature. Therefore, if we consider the roll angle to be a second order system with aileron input ([42]), then a trajectory must maintain continuity up to the derivative of curvature to be kinodynamically feasible. Furthermore, we can minimize roll rates and roll accelerations by penalizing the first and second derivatives of curvature of the trajectory.

A quadrotor is capable of rolling about any axis in the body's horizontal plane and thus minimizing the snap or $4^{\text{th}}$ derivative of independent splines in three dimensions will minimize abrupt changes in control inputs needed to follow the path. For a fixed-wing vehicle, however, the picture is substantially more complicated. The axis of roll motion rotates with the heading of the vehicle and the vehicle must turn with a finite radius, all while traveling at a nearly constant speed. This makes the independent-axes spline representation inapplicable.

To meet the needs of a dynamically feasible fixed-wing trajectory, we define a "Dubins-Polynomial" trajectory as a Dubins curve combined with a transverse polynomial offset from that Dubins curve. The key idea behind the Dubins-Polynomial formulation is that by parameterizing a lateral offset from a nominal path, we isolate the roll dynamics of interest for steering a fixed-wing aircraft.

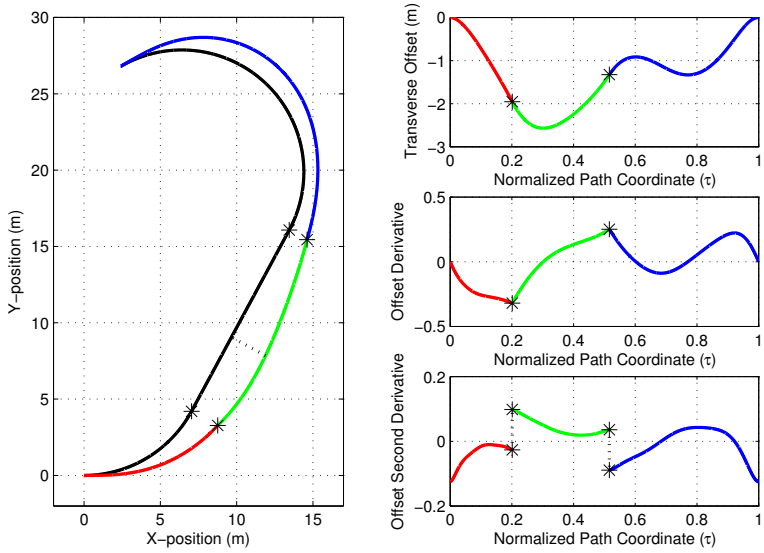



Figure 15: Example trajectory for a fixed-wing aircraft. The underlying Dubins curve (LSL) is illustrated in black in the left panel. The three polynomial segments representing the transverse offset from this Dubins curve are illustrated in red, green and blue. These polynomial segments are also illustrated with their first and second derivatives in the right panel. Note the intentional discontinuity in the second derivative, which corrects for the change in curvature between the circular arcs and the straight line segment.

An example Dubins-Polynomial trajectory is illustrated in Figure 15. The sequence of polynomials is illustrated in red, green and blue. In this figure, the red arc is offset from the first curve, which is a left turn with an 8 m radius. The green arc is an offset from the middle segment, which is a straight line. Finally, the blue arc is an offset from the third segment, which is another left turn with 8 m radius.

The polynomials themselves are illustrated on the right side of Figure 15. Notice that the offset and its first derivative are constrained to be zero at the start and end of this trajectory. The second derivative, which corresponds to curvature, is constrained to have a value equal to the curvature of the underlying Dubins segment at the beginning and end of the trajectory. In this example, the initial and final curvatures are equal to 0.125 $m^{-1}$, corresponding to the 8 m radius.

Since the curvature of the underlying Dubins curve changes from 0.125 $m^{-1}$ to 0 $m^{-1}$ at the intersection between the circular arcs and the straight line, we must enforce an offset in the second derivatives of the polynomials at these junctions. These offsets are seen in the second derivative plot in the bottom-right of Figure 15. These offsets correct for the discontinuities in curvature between segments of the underlying Dubins curve. Note that the spatial locations of the intermediate junctions between polynomial segments, as well as higher derivatives, are not specified directly. Rather they are optimized according to a cost function.

### 7.2.1 Procedure for Generating Fixed-Wing Trajectories

The optimization method described in Section 4.2 is capable of jointly optimizing polynomial segments subject to offset constraints on their derivatives where two segments meet. While the idea of jointly optimizing polynomial segments is straightforward, the expressions for the transverse derivatives of a polynomial offset are rather complex. In Sections 7.2.2 and 7.2.3 we develop the necessary expressions. As seen in these expressions, the transverse derivatives of the resulting spatial curve are nonlinear functions of the polynomial derivatives. Therefore, the initial joint optimization will, in general, be an approximation that must be corrected to ensure true continuity of derivatives. The approximation lies in the fact that we are using linear constraints in our quadratic program to enforce continuity of polynomials, when in fact the derivatives we actually want to constrain (the derivatives of the trajectory through space) are nonlinear functions of those polynomials.

The overall strategy is to first generate the underlying Dubins curve based on initial and final points, $(\Delta_x, \Delta_y, \psi)_0$ and $(\Delta_x, \Delta_y, \psi)_1$. Then, we solve for the piecewise-polynomial transverse offset using the joint optimization method outlined in Section 4.2.1, with specified boundary constraints, $(\kappa, \dot{\kappa}, \ddot{\kappa})_0$ and $(\kappa, \dot{\kappa}, \ddot{\kappa})_1$. These boundary constraints on curvature correspond to endpoint constraints on the the second, third and fourth derivatives of the polynomial. For the purpose of achieving derivative continuity between Dubins curve segments, all the elements of $\sigma_i$ in equation (12) are set to zero except for the second derivative, which is set to the curvature difference between the preceding and following segments. As illustrated in Figure 15, this offset corrects for the underlying curvature difference between the arc segments and straight line segments.

We have now optimized a polynomial, however as stated above, the transverse derivatives of the resulting spatial curve that we actually wish to constrain are nonlinear functions of the polynomial derivatives. Therefore, we must repair our polynomial solution to obtain a trajectory whose transverse derivatives are in fact continuous. In order to do so, we compute "true" transverse derivative values at each junction by taking the average of the transverse derivatives for the preceding and following segment at that junction. These transverse derivatives are derived below, and are given in equations 88-90 (or 98-100 for a line).

The purpose of computing an average at each junction is so that the preceding and following segments can be re-optimized individually, using that set of average derivatives as boundary conditions. Thus, the resulting sequence of individually optimized polynomials will have truly continuous transverse derivatives. To accomplish this goal, we invert the transverse derivatives using equations 91-93 (101-103) to obtain polynomial boundary conditions for each segment. Finally, we perform individual optimizations for each segment using the procedure given in Section 4.2. In the next two sections we develop the necessary equations to compute these transverse derivatives.

### 7.2.2 Polar Coordinates Corrections

The polynomial offset from a circular arc in a Dubins curve is effectively a polynomial in polar coordinates, expressed as $R(\theta) = R_0 \pm P(\theta)$, where the sign of addition depends on whether the arc is CCW (-) or CW (+). $R_0$ is the

nominal radius of the underlying circular arc, $P(\theta)$ is the polynomial offset from the circular arc, and $\theta$ is the parameter indicating position along the arc.

In this section we will use unit vectors $\hat{r}$ and $\hat{\theta}$, denoting the radial and angular directions, respectively. Note that $\hat{\theta}$ may be CCW-positive or CW-positive depending on the intrinsic arc direction, in contrast to conventional CCW-only use. The parameter $\theta$ sweeps positive in the $\hat{\theta}$ direction. When used in a standard coordinate system $\hat{r}(\theta) = \cos(\theta)\mathbf{w}_x + \sin(\theta)\mathbf{w}_y$ and $\hat{\theta}(\theta) = -\sin(\theta)\mathbf{w}_x + \cos(\theta)\mathbf{w}_y$ for a CCW arc and $\hat{\theta}(\theta) = \sin(\theta)\mathbf{w}_x - \cos(\theta)\mathbf{w}_y$ for a CW arc.

To analyze the true derivatives of the trajectory in polar coordinates, we use the arc path $\rho$ in the $\hat{r}$ direction. Using the identities $\hat{r}'(\theta) = \hat{\theta}(\theta)$ and $\hat{\theta}'(\theta) = -\hat{r}(\theta)$ we obtain

$$\rho(\theta) = R(\theta)\hat{r}(\theta) \tag{76}$$

$$\rho'(\theta) = P'(\theta)\hat{r}(\theta) + R(\theta)\hat{\theta}(\theta) \tag{77}$$

$$\rho''(\theta) = (P''(\theta) - R(\theta))\hat{r}(\theta) + 2P'(\theta)\hat{\theta}(\theta) \tag{78}$$

$$\rho'''(\theta) = (P'''(\theta) - 3P'(\theta))\hat{r}(\theta) + (3P''(\theta) - R(\theta))\hat{\theta}(\theta), \tag{79}$$

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 $\theta$ as a function of the distance $s$ to get the arc as $\rho(\theta(s))$ and take the derivatives with respect to the distance.

$$\frac{d\rho(\theta(s))}{ds} = \rho(\theta(s))'\theta'(s) \tag{80}$$

$$\frac{d^2\rho(\theta(s))}{ds^2} = \rho''(\theta(s))\theta'(s)^2 + \rho'(\theta(s))\theta''(s) \tag{81}$$

$$\frac{d^3\rho(\theta(s))}{ds^3} = \rho'''(\theta(s))\theta'(s)^3 + 3\rho''(\theta(s))\theta''(s)\theta'(s) + \rho'(\theta(s))\theta'''(s). \tag{82}$$

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

$$||\rho'(\theta(s))|| = 1 = \theta'(s)\sqrt{P'(\theta(s))^2 + (R(\theta(s)))^2}, \tag{83}$$

which yields the relationship:

$$\theta'(s) = \frac{1}{\sqrt{P'(\theta(s))^2 + (R(\theta(s)))^2}}, \tag{84}$$

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

$$\theta' = \frac{1}{L} \tag{85}$$

$$\theta'' = -\frac{P'(P'' + R)}{L^4} \tag{86}$$

$$\theta''' = -\frac{L^2(P''(P'' + R) + P'(P''' + P')) - 4P'^2(P'' + R)^2}{L^7}. \tag{87}$$

For convenience we define $L = \sqrt{P'^2 + R^2}$ and omit the argument $\theta$ for compactness in the expressions. Equations 77 - 79 and 85-87 can then be plugged into 80-82 to obtain expressions for the true path derivatives in the $\hat{r}$ direction:

$$\hat{r}^T \frac{d\rho(\theta(s))}{ds} = \frac{P'}{\sqrt{P'^2 + R^2}} \tag{88}$$

$$\hat{r}^T \frac{d^2\rho(\theta)}{ds^2} = P'' \left( \frac{1}{L^2} - \frac{P'^2}{L^4} \right) - \frac{R}{L^2} - \frac{P'^2 R}{L^4} \tag{89}$$

$$\begin{aligned} \hat{r}^T \frac{d^3\rho(\theta(s))}{ds^3} = \frac{P''' - 3P'}{L^3} - 3\frac{P'(P'' - R)^2}{L^5} \\ - P' \left( \frac{P''^2 + P''R + P'P''' + P'^2}{L^5} - \frac{4P'^2(P'' + R)^2}{L^7} \right), \end{aligned} \tag{90}$$

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:

$$P' = \frac{R\hat{r}^T \frac{d\rho(\theta(s))}{ds}}{\sqrt{1 - \left( \hat{r}^T \frac{d\rho(\theta(s))}{ds} \right)^2}} \tag{91}$$

$$P'' = \frac{\hat{r}^T \frac{d^2\rho(\theta)}{ds^2} + \frac{R}{L^2} + \frac{P'^2 R}{L^4}}{\frac{1}{L^2} - \frac{P'^2}{L^4}} \tag{92}$$

$$\begin{aligned} P''' = L^3 \hat{r}^T \frac{d^3\rho(\theta(s))}{ds^3} + 3P' + 3\frac{P'(P'' - R)^2}{L^2} \\ + P' \left( \frac{P''^2 + P''R + P'P''' + P'^2}{L^2} - \frac{4P'^2(P'' + R)^2}{L^4} \right). \end{aligned} \tag{93}$$

### 7.2.3 Line Segment Corrections

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

$$\rho(\theta) = P(\theta)\hat{r} + \theta\hat{\theta} \tag{94}$$

$$\rho'(\theta) = P'(\theta)\hat{r} + \hat{\theta} \tag{95}$$

$$\rho''(\theta) = P''(\theta)\hat{r} \tag{96}$$

$$\rho'''(\theta) = P'''(\theta)\hat{r}. \tag{97}$$

Following the same procedure as for arc segments we obtain:

$$\hat{r}^T \frac{d\rho(\theta(s))}{ds} = \frac{P'}{\sqrt{P'^2 + 1}} \tag{98}$$

$$\hat{r}^T \frac{d^2\rho(\theta)}{ds^2} = \frac{P''}{L^2} - \frac{P'^2 P''}{L^4} \tag{99}$$

$$\hat{r}^T \frac{d^3\rho(\theta(s))}{ds^3} = \frac{P'''}{L^3} - 3\frac{P'P''^2}{L^5} - P'\frac{L^2(P''^2 + P'P''') - 4P'^2P''^2}{L^7}, \tag{100}$$

and solved for the polynomial derivatives:

$$P' = \frac{\hat{r}^T \frac{d\rho(\theta(s))}{ds}}{\sqrt{1 - \left(\hat{r}^T \frac{d\rho(\theta(s))}{ds}\right)^2}} \tag{101}$$

$$P'' = L^2 \hat{r}^T \frac{d^2\rho(\theta)}{ds^2} + \frac{P'^2 P''}{L^2} \tag{102}$$

$$P''' = L^3 \hat{r}^T \frac{d^3\rho(\theta(s))}{ds^3} + 3\frac{P' P''^2}{L^2} + P' \frac{L^2(P''^2 + P' P''') + 4P'^2 P''^2}{L^4}. \tag{103}$$

### 7.2.4 Example Dubins-Polynomial Paths

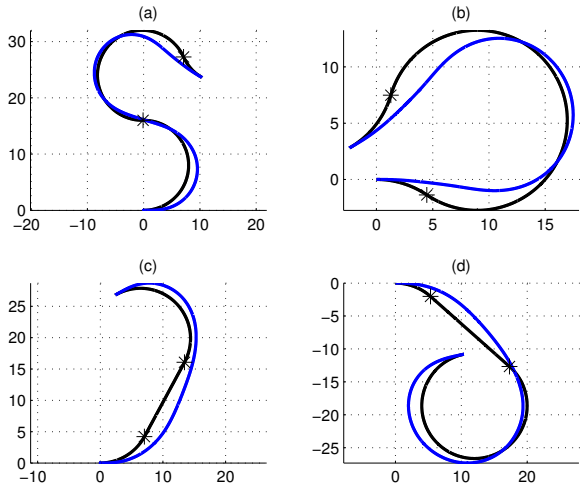



Figure 16: This figure shows Dubins-Polynomial paths for various configurations. The blue line in the transverse polynomial path and the black line is the Dubins path from which it is offset. For each of these examples, the curvature is constrained to be 0 at the start and end of the path. We can see that generally the optimization yields paths that distribute the shifts in curvature across the Dubins segment junctions (denoted with black stars), reducing the need for abrupt changes in roll angle and roll rate.

It is important to note that the polynomials used in equations 91-93 and 88-90 are in terms of the arc angle $\theta$ 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. To verify the correctness of our expressions for derivatives, we numerically integrated the third path derivative ($\ddot{\boldsymbol{\Delta}}$) for example paths and confirmed that the numerical integration converged exactly to the example path as discretization was refined. Example curves for various configurations are shown in Figure 16.

### 7.2.5 Limitations of Dubins-Polynomial Trajectories

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 7.2.2 and 7.2.3 that the approximation breaks down for $P'$ and $P$ far from zero. 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.

## 7.3 Fixed-Wing 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 a model of our fixed-wing aircraft in AVL, an open source extended vortex lattice aerodynamic analysis program [13].

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 $R_0 = 8$ m with a desired flight velocity of 7 m/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 zero. The paths were sampled in a 40 m 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.

If the simulated vehicle deviated by more than 10 m from the planned path, the case was marked as a failure and discarded. These samples were used to generate the results shown in Table 6. 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

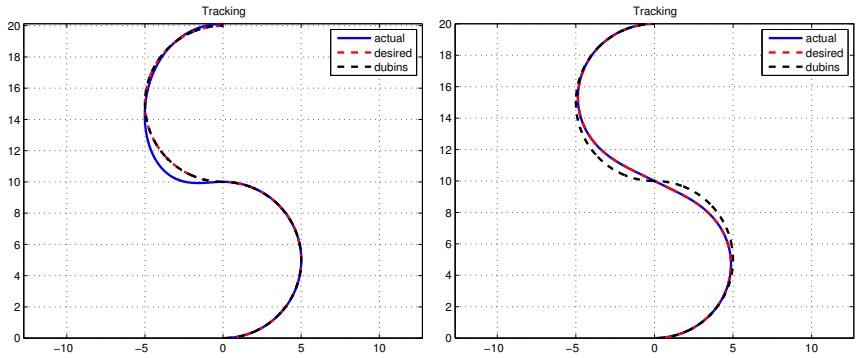
| Path Type | Error Norm (m) | Fraction Failed | Computation Time ($\mu$s) |
|---|---|---|---|
| Dubins | 1.60 | 0.194 | 20.0 |
| 3rd Order | 0.67 | 0.006 | 495.7 |
| 4th Order | 0.75 | 0.003 | 516.6 |
| Full Model | - | - | 10609.3 |

Table 6: This table shows path-following error, fraction of simulations diverged, and computation time for a path for Dubins curves, transverse-polynomial curves enforcing continuity up to 3rd and 4th order, and a full simulation of the nonlinear model.

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 20 times 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 [7].

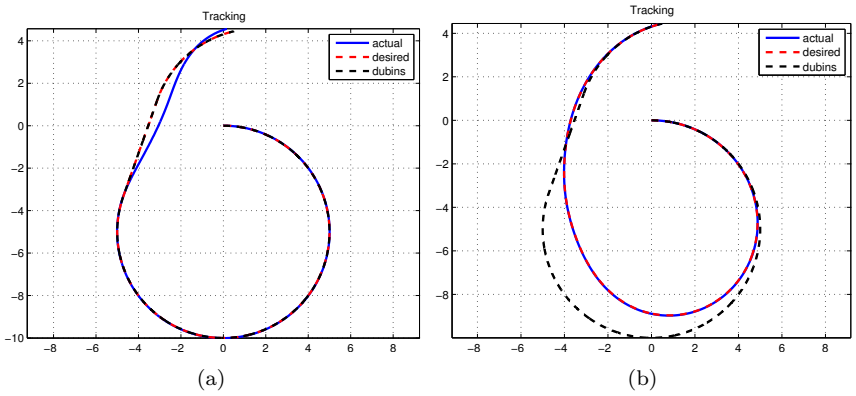
Figures 17 and 18 give some intuition for why the optimized paths are superior for tracking. Each depicts an example of tracking performance with the coordinated flight model for both Dubins curves and the same Dubins curve with an optimized polynomial offset. As we would expect, the coordinated flight model exhibits perfect tracking behavior on these paths.




(a) (b)

Figure 17: 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.

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 123, 124, and 125). This is not surprising since we approximated the mapping using simple linear fitting,




(a) (b)

Figure 18: 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.

treating the nonlinear model as a black box. A more refined model fitting might be useful to further improve performance. 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 step as future work on the actual vehicle.

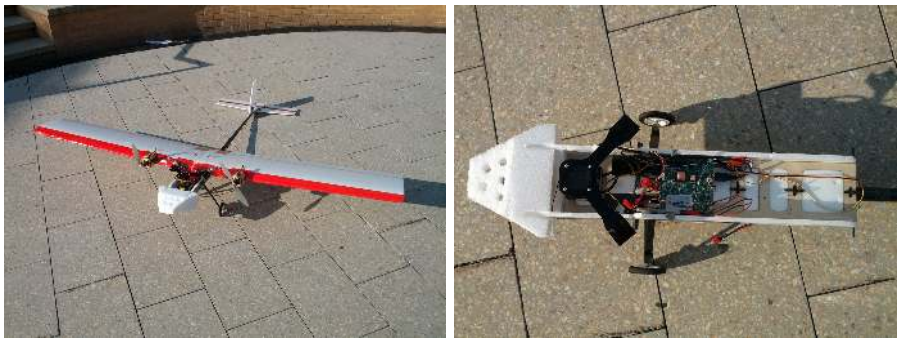
## 7.4 Custom Fixed-Wing Vehicle Platform

While not a primary focus of our research, the payload, speed, and size constraints for flying in confined indoor spaces led us to design and build a custom fixed-wing vehicle, shown in Figure 19. Our experimental platform consists of a custom built fixed-wing vehicle carrying a Hokuyo UTM-30LX laser rangefinder, a Microstrain 3DM-GX3-25 IMU, and a 2.0GHz Intel Atom based flight computer.

## 7.5 Fixed-Wing Experimental Results

We conducted closed-loop autonomous flights in a gymnasium and in a parking garage. In total, we have logged over 20 minutes of autonomous flight time, at speeds ranging from 7 m/s to 11 m/s. Initial closed-loop flight tests were conducted with circular holding patterns in the gymnasium. We then performed "simple" and "slalom" paths through curtains hung from the gymnasium ceiling, and a figure-eight through the pillars of a parking garage. These trajectories are depicted in Figure 20. They were constructed by hand-selecting waypoints and optimizing Dubins-Polynomial trajectories between the waypoints.

During early testing, failures were caused by dynamics mismatches between the model and the actual vehicle. However, once parameters of the dynamics model and controller were sufficiently fit, the fixed-wing flights were very repeatable. The trajectories depicted in Figure 20 were flown without failure



(a) (b)

Figure 19: This figure shows our custom vehicle (a) and an internal view of the fuselage (b). The vehicle is designed to fly slowly enough to fly in tight spaces, while still maintaining maneuvering capability. The construction makes use of foam, carbon fiber, and thin plywood.

and the number of laps around each trajectory was limited only by battery life. The trajectories in the gymnasium lasted several minutes each, and the parking garage trajectory accounted for 8 minutes of continuous flight or approximately 5 km of flight distance.

Table 7 gives the controller tracking errors incurred during execution of these trajectories. Tracking performance for the simple gymnasium trajectory and the parking garage figure-eight was excellent, with an average position error of only approximately 30 cm over many laps around these environments. Given the 2 m wingspan of the vehicle, this level of tracking performance is sufficient to fly very close to obstacles without risking collision.

It is interesting to note that tracking of the slalom path was not as accurate. Average tracking error for this trajectory was more than twice the tracking error on the other two trajectories (though still less than half the wingspan). The slalom trajectory is composed of a sequence of seven back-to-back turns, most of which are tight turns nearly at the estimated curvature limit of the vehicle. Therefore, this trajectory required the most dramatic rolling motions to track. Furthermore, this trajectory has no straight segments of any significant length, so it is much more difficult for the control system to re-converge if tracking errors develop. The drop in performance observed in the slalom trajectory suggests the need for a more accurate estimate of the dynamic limits of the vehicle.

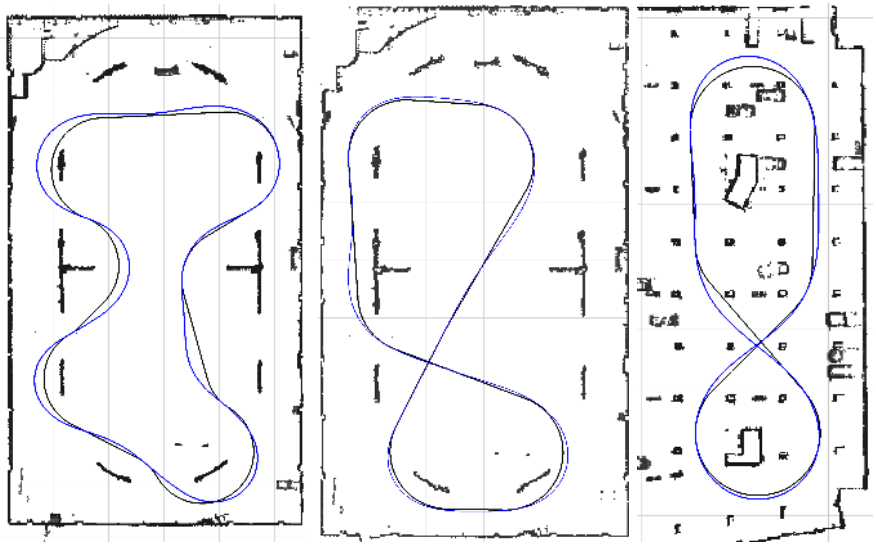
The fixed-wing vehicle during autonomous flight is shown in Figure 21 and onboard camera views are shown in Figure 22. A video illustrating the fixed-wing experimental results is available at: `http://groups.csail.mit.edu/rrg/fixed_wing_flight`.

# 8 Conclusion and Future Work

In this paper, we have presented several novel algorithms for trajectory planning and state estimation needed to achieve aggressive autonomous flight in obstacle-dense indoor environments. The experimental results presented on both the

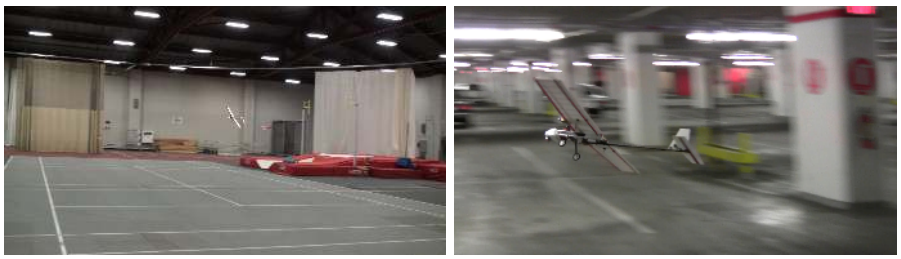
| Environment | Mean Position Error (m) | Mean Velocity Error (m/s) |
|---|---|---|
| Gymnasium ("slalom") | 0.8313 | 1.3121 |
| Gymnasium ("simple") | 0.3489 | 0.4839 |
| Parking Garage | 0.2653 | 0.3206 |

Table 7: This table gives position and velocity tracking errors incurred during execution of the fixed-wing trajectories through a gymnasium (Figure 20(a) and Figure 20(b)), and a parking garage (Figure 20(c)). Since we do not have ground-truth state estimation, these errors are reported with respect to the onboard state estimate.



(a) (b) (c)

Figure 20: Dubins-Polynomial trajectories in a gymnasium ("slalom") (a) and ("simple") (b), and in a parking garage (c). The black lines are the underlying Dubins paths, and the blue lines are the polynomial offset trajectories.



(a) (b)

Figure 21: Autonomous flight in a gymnasium (a) and a parking garage (b).

(a) (b)

Figure 22: Onboard camera images from autonomous flight in a gymnasium (a) and a parking garage (b).

fixed-wing and quadrotor vehicles highlight the generality of our approach. Since our state estimator is based upon the rigid-body equations of motion, it makes no assumptions about vehicle type, and therefore can be applied without modification to any vehicle capable of carrying the required sensors. This generality makes it an extremely powerful tool for cross-platform experimentation.

While our planning approaches are vehicle-specific, they both make use of the same underlying property of differential flatness and reuse a significant portion of the mathematical tools needed to optimize trajectories. We believe that our overall strategy could be applied to a range of vehicles beyond what we have presented here. In particular, the results for our quadrotor could apply to any multi-copter, simplified conventional helicopter, submarine or satellite whose thrusters are capable of exerting forces and moments like a quadrotor. Likewise, the results for the fixed-wing airplane would be equally applicable to any curvature-constrained or Dubins-type vehicle, including automobiles.

There are several simple extensions to the planning tools we have presented here, which would improve the performance of automatic trajectory generation. For the quadrotor, we take waypoints from a low-dimensional search algorithm as constraints in the optimization. However, it is often possible to move these waypoints within their local neighborhood to achieve lower-cost trajectories. Moving the waypoints may result in relaxing unnecessarily tight turns in a trajectory, or moving the whole trajectory into a different, preferable homotopy

class. Such an optimizer might take the form of one presented in [39].

For the fixed-wing vehicle, we have not presented any results using a low-dimensional search to seed the polynomial optimization, as we did for the quadrotor. It would be straightforward to apply a simple search algorithm to automatically compute a Dubins path and then optimize corresponding Dubins-Polynomial trajectories to completely automate the planning procedure. Furthermore, we are currently optimizing the Dubins-Polynomial trajectory associated with each 3-segment Dubins word individually. It would also be straightforward to jointly optimize the entire sequence of polynomials along an entire trajectory to obtain lower cost trajectories.

In this paper, we have focused entirely on planning complete trajectories in pre-mapped environments. However, given the speed with which we can plan trajectories, these tools may apply equally well to the task of planning partial trajectories in a receding-horizon fashion for cases where the environment is not entirely known but is periodically updated during execution.

## Acknowledgements

The support of the ARO MAST CTA and ONR under MURI N00014-09-1-1052 and NDSEG is gratefully acknowledged.

# A Polynomial Trajectory Cost Matrix and Constraints

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

$$(P^2)_n = \sum_{j=0}^{N} p_j p_{n-j}. \tag{104}$$

The $r$th derivative is given by

$$P^{(r)}(t) = \sum_{n=r}^{N} \left( \prod_{m=0}^{r-1} (n-m) \right) p_n t^{n-r}. \tag{105}$$

Using equations 104 and 105 we can write the $r$th component of the cost as a quadratic function of the polynomial coefficients:

$$\begin{aligned}
J_r &= \int_0^\tau P^{(r)}(t)^2 dt && (106)\\
&= \int_0^\tau \left( \sum_{n=r}^{N} \left( \prod_{m=0}^{r-1} (n-m) \right) p_n t^{n-r} \right)^2 dt && (107)\\
&= \int_0^\tau \sum_{n=0}^{2N} \sum_{j=0}^{N} \left( \prod_{m=0}^{r-1} (j-m)(n-j-m) \right) p_j p_{n-j} t^{n-2r} dt && (108)\\
&= \sum_{n=0}^{2N} \sum_{j=0}^{N} \left( \prod_{m=0}^{r-1} (j-m)(n-j-m) \right) p_j p_{n-j} \frac{\tau^{n-2r+1}}{(n-2r+1)}. && (109)
\end{aligned}$$

We find $\mathbf{Q}_r$ matrices by taking the Hessian of this expression:

$$\begin{aligned}
\frac{\partial J_r}{\partial p_i} &= \frac{\partial}{\partial p_i} \sum_{n=0}^{2N} \sum_{j=0}^{N} \left( \prod_{m=0}^{r-1} (j-m)(n-j-m) \right) p_j p_{n-j} \frac{\tau^{n-2r+1}}{(n-2r+1)} && (110)\\
&= 2 \sum_{n=0}^{2N} \left( \prod_{m=0}^{r-1} (i-m)(n-i-m) \right) p_{n-i} \frac{\tau^{n-2r+1}}{(n-2r+1)} && (111)\\
\frac{\partial^2 J_r}{\partial p_i \partial p_l} &= 2 \sum_{n=0}^{2N} \left( \prod_{m=0}^{r-1} (i-m)(n-i-m) \right) \frac{\partial p_{n-i}}{\partial p_l} \frac{\tau^{n-2r+1}}{(n-2r+1)} && (112)\\
&= 2 \left( \prod_{m=0}^{r-1} (i-m)(l-m) \right) \frac{\tau^{i+l-2r+1}}{(i+l-2r+1)} && (113)\\
\mathbf{Q}_r^{il} &= \begin{cases} 2 \left( \prod_{m=0}^{r-1} (i-m)(l-m) \right) \frac{\tau^{i+l-2r+1}}{(i+l-2r+1)} : & i \geq r \wedge l \geq r \\ 0 : & i < r \vee l < r. \end{cases} && (114)
\end{aligned}$$

We form the complete cost matrix as a weighted sum of all $\mathbf{Q}_r$ matrices, where the weights $c_r$ indicate the penalty on the $r^{th}$ derivative:

$$\mathbf{Q} = \sum_{r=0}^{N} c_r \mathbf{Q}_r. \tag{115}$$

The constraints on the endpoint values of $P$ and its derivatives can be written as linear functions of the coefficients:

$$\mathbf{Ap} - \mathbf{b} = \mathbf{0}. \tag{116}$$

The value of the $r$th derivative at $\tau$ is given by:

$$P^{(r)}(\tau) = \sum_{n=r}^{N} \left( \prod_{m=0}^{r-1} (n-m) \right) p_n \tau^{n-r}. \tag{117}$$

For a polynomial segment spanning from 0 to $\tau$ satisfying derivative constraints on both ends of the interval, $\mathbf{A}$ and $\mathbf{b}$ are constructed as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_0 \\ \mathbf{A}_\tau \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \mathbf{b}_0 \\ \mathbf{b}_\tau \end{bmatrix} \tag{118}$$

$$\mathbf{A}_{0_{rn}} = \begin{cases} \prod_{m=0}^{r-1}(r-m): & r = n \\ 0: & r \neq n \end{cases} \tag{119}$$

$$\mathbf{b}_{0_r} = P^{(r)}(0) \tag{120}$$

$$\mathbf{A}_{\tau_{rn}} = \begin{cases} \left( \prod_{m=0}^{r-1}(n-m) \right) \tau^{n-r}: & n \geq r \\ 0: & n < r \end{cases} \tag{121}$$

$$\mathbf{b}_{\tau_r} = P^{(r)}(\tau). \tag{122}$$

The $\mathbf{A}$ matrix simply maps the polynomial coefficients to the derivatives at the endpoints of the polynomial. The vector $\mathbf{b}$ is used to specify or constrain certain derivative values. These constraints are useful, for example, when we want to enforce that velocity and acceleration and higher derivatives are zero at the beginning of a quadrotor trajectory, or when we want to match the values of derivatives at the junction between two polynomial trajectory segments.

# B Fixed-Wing Differentially Flat Control Input Conversion

The trajectory stabilization algorithm outlined in Section 7.1 provides us with $\omega_{v_x}$, $\dot{a}_{v_z}$, and $s^{(3)}$. 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. Since we have a path parameter taking the place of $\dot{a}_{v_x}$ in the differentially flat representation, we can use the the throttle to maintain a desired cruise speed.

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 [42]. Thus one possibility would be to differentiate equation 71 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.

The approach we use for control is to integrate $\dot{a}_{v_z}$ and maintain $a_{v_z}$ (normal acceleration) as a controller state. Desired roll rate and normal acceleration are then translated algebraically into elevator, aileron, and rudder commands. Using basic results from aircraft stability and control, we can convert desired normal acceleration, steady-state roll rate ($p$), and steady-state yaw rate ($r$) to control surface deflections as follows:

$$\delta_e = c_0 \frac{a_{v_z}}{V^2} + c_1 \tag{123}$$

$$\delta_a = c_2 \frac{p}{V} + c_3 \frac{r}{V} \tag{124}$$

$$\delta_r = c_4 \frac{p}{V} + c_5 \frac{r}{V} \tag{125}$$

where $\delta_e$ is the elevator deflection, $\delta_a$ is the aileron deflection, and $\delta_r$ is the rudder deflection. Equations 123, 124, and 125 give us the relationships we need. Coefficients $c_0$-$c_5$ can be empirically fit from flight or simulation data and then used to map the desired normal acceleration to actual control inputs.

# References


[1] CRRC simulator. `http://sourceforge.net/apps/mediawiki/crrcsim/`. Accessed: April 2014.

[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, 2005.

[3] 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.

[4] A. J. Barry, T. Jenks, A. Majumdar, H.-T. Lin, I. G. Ros, A. Biewener, and R. Tedrake. Flying between obstacles with an autonomous knife-edge maneuver. In *IEEE Int. Conf. on Robotics and Automation*, 2014.

[5] A. J. Barry and A. Majumdar. Personal communication, 2014.

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

[7] J. T. Betts. *Practical methods for optimal control and estimation using nonlinear programming*, volume 19. Siam, 2010.

[8] A. Bry, A. Bachrach, and N. Roy. State estimation for aggressive flight in GPS-denied environments using onboard sensing. In *IEEE Int. Conf. on Robotics and Automation*, 2012.

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

[10] G. Buskey, J. Roberts, P. Corke, and G. Wyeth. Helicopter automation using a low-cost sensing system. In *Proceedings of the Australasian Conf. on Robotics and Automation*, 2003.

[11] H. Chitsaz and S. LaValle. Time-optimal paths for a dubins airplane. In *IEEE Conf. on Decision and Control*, pages 2379–2384, 2007.

[12] M. Cutler and J. P. How. Actuator constrained trajectory generation and control for variable-pitch quadrotors. In *AIAA Conf. on Guidance, Navigation, and Control*, 2012.

[13] M. Drela and H. Youngren. Athena vortex lattice (AVL). `http://web.mit.edu/drela/Public/web/avl/`, April 2014. Accessed: April 2014.

[14] N. Faiz, S. K. Agrawal, and R. M. Murray. Trajectory planning of differentially flat systems with dynamics and inequalities. *Guidance, Control, and Dynamics*, 24(2):219–227, 2001.

[15] D. Ferguson, T. M. Howard, and M. Likhachev. Motion planning in urban environments. *Journal of Field Robotics*, 25(11-12):939–960, 2008.

[16] T. Fraichard. Smooth trajectory planning for a car in a structured world. In *IEEE Int. Conf. on Robotics and Automation*, pages 318–323. IEEE, 1991.

[17] E. Frazzoli, M. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Guidance, Control, and Dynamics*, 25(1):116–129, 2002.

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

[19] S. Grzonka, G. Grisetti, and W. Burgard. Towards a navigation system for autonomous indoor flying. In *IEEE Int. Conf. on Robotics and Automation*, 2009.

[20] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[21] J. Hauser and R. Hindman. Aggressive flight maneuvers. In *IEEE Conf. on Decision and Control*, 1997.

[22] M. Hehn, R. Ritz, and R. D'Andrea. Performance benchmarking of quadrotor systems using time-optimal control. *Autonomous Robots*, 33(1-2):69–88, 2012.

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

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

[25] J. hwan Jeon, S. Karaman, and E. Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*. In *IEEE Decision and Control and European Control Conf. (CDC-ECC)*, pages 3276–3282, 2011.

[26] S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *IEEE Conf. on Decision and Control*, pages 7681–7687, 2010.

[27] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 30(7):846–894, 2011.

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

[29] 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*, 2004.

[30] D. B. Kingston and R. W. Beard. Real-time attitude and position estimation for small uavs using low-cost sensors. In *AIAA Unmanned Unlimited Technical Conf., Workshop and Exhibit*, pages 2004–6488, 2004.

[31] T. J. Koo and S. Sastry. Output tracking control design for a helicopter model based on approximate linearization. In *IEEE Conf. on Decision and Control*, pages 3635–3640, 1998.

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

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

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

[35] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Int. Journal of Robotics Research*, 20(5):378–400, 2001.

[36] T. Lee, M. Leoky, and N. H. McClamroch. Geometric tracking control of a quadrotor UAV on SE(3). In *IEEE Conf. on Decision and Control*, pages 5420–5425, 2010.

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

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

[39] M. Milam, K. Mushambi, and R. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *IEEE Conf. on Decision and Control*, volume 1, pages 845–851 vol.1, 2000.

[40] R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. In *ASME International Mechanical Engineering Congress and Exposition*. Citeseer, 1995.

[41] J. Pan, L. Zhang, and D. Manocha. Collision-free and smooth trajectory computation in cluttered environments. *Int. Journal of Robotics Research*, 31(10):1155–1175, 2012.

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

[43] M. Pivtoraiko and A. Kelly. Kinodynamic motion planning with state lattice motion primitives. In *IEEE Int. Conf. on Intelligent Robots and Systems*, 2011.

[44] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE Int. Conf. on Robotics and Automation*, pages 489–494, 2009.

[45] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Int. Symposium on Robotics Research*, 2013.

[46] 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, 2008.

[47] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Proceedings of Robotics: Science and Systems*. Citeseer, 2013.

[48] S. Shen, N. Michael, and V. Kumar. Autonomous multi-floor indoor navigation with a computationally constrained MAV. In *IEEE Int. Conf. on Robotics and Automation*.

[49] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1):99–141, 2001.

[50] R. van der Merwe and E. Wan. Sigma-point Kalman filters for integrated navigation. In *Proc. Institute of Navigation (ION)*, Dayton, OH, 2004.

[51] M. J. Van Nieuwstadt and R. M. Murray. Real-time trajectory generation for differentially flat systems. *Int. J. Robust and Nonlinear Control*, 8(11):995–1020, 1998.

[52] 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, 2010.