Polynomial Trajectory Planning for Quadrotor Flight

Charles Richter, Adam Bry and Nicholas Roy

***Abstract*— We explore the challenges of planning trajectories through complex environments for quadrotors. We use the RRT* algorithm to generate an initial route through a 3D environment and then construct a trajectory consisting of a sequence of polynomial spline segments to follow that route. We present a method of jointly optimizing polynomial path segments that is numerically stable for high-order polynomials and large numbers of segments, and is easily formulated for efficient sparse computation. Using a differentially flat representation of the quadrotor, these polynomial trajectories encode the complete dynamics of the vehicle and allow calculation of feed-forward control commands in closed form, eliminating the need to sample in a high-dimensional state space or carry out expensive dynamics simulations during planning. Our approach generates high-quality trajectories much faster than purely sampling-based kinodynamic planning methods, but sacrifices convergence to the global optimum. We demonstrate the performance of our algorithm by efficiently generating a trajectories through challenging indoor spaces and successfully traversing them at speeds up to 8m/s.**

I. INTRODUCTION

Recent advances in small unmanned aircraft have enabled precise, dynamic flight maneuvers in indoor environments [1], [2], [3]. Simultaneously, advances in fast, accurate state estimation methods have enabled these vehicles to fly through dense, cluttered spaces without the need for a motion capture system [4]. These capabilities together motivate the challenge addressed in this paper, which is to efficiently generate trajectories for agile quadrotor flight through maps of real-world environments.

While there exist advanced techniques for robotic navigation and trajectory optimization, there has yet to emerge a single algorithm that can both find *and* optimize a quadrotor trajectory through a complex real-world environment quickly enough to be useful for a deployable robotic system. While algorithms such as RRT* provably converge to the optimal solution in the limit of infinite samples, it is often impractical to rely on this limit to perform optimization for vehicles with nonlinear 12-DOF dynamics. These algorithms have been most successful for simple Dubins-vehicle or double-integrator systems where analytical techniques can be used to steer between two points in state space [5]. For other systems, the search over dynamically feasible trajectories often requires iterative simulation of the equations of motion.

Nonlinear programming techniques for trajectory optimization, such as direct collocation and shooting methods, can also be used to find optimal paths for systems with general dynamics. However, these methods are computationally intensive and often require accurate analytical representations of all environmental constraints in order to compute cost gradients with respect to obstacles. These limitations make them impractical when constraints are represented in the form of an occupancy map.

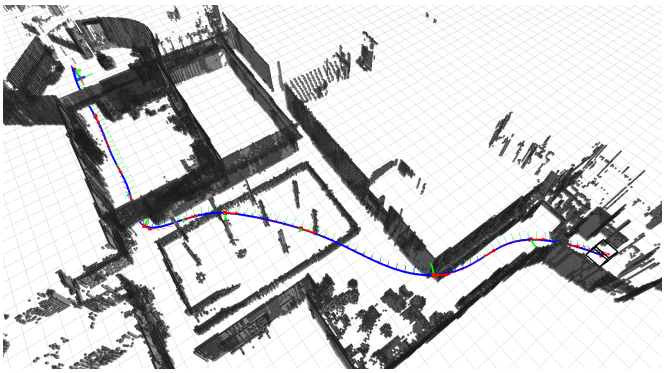


Fig. 1: Automatically generated 3D trajectory navigating a real-world environment with closely-spaced obstacles.

Nevertheless, an explicit optimization step is useful for high-speed trajectories through cluttered environments. Minimum-snap polynomial splines have proven very effective as quadrotor trajectories, since the motor commands and attitude accelerations of the vehicle are proportional to the snap, or forth derivative, of the path [6]. Minimizing the snap of a trajectory quantifies a notion of "smoothness" that is desirable for maintaining the quality of onboard sensor measurements as well as avoiding paths that would require abrupt or excessive control inputs.

The differentiability of polynomial trajectories makes them a natural choice for use in a *differentially flat* representation of the quadrotor dynamics. Differential flatness provides an analytical mapping from a path and its derivatives to the states and control input required to follow that path. This powerful property effectively guarantees feasibility of any differentiable trajectory, provided that its derivatives are sufficiently bounded to avoid input saturation, thus eliminating the need for iterative simulation in the search for trajectories.

A. Problem Statement

Given a 3D occupancy map of an environment, we wish to efficiently compute feasible, minimum-snap trajectories that follow the shortest route from start to goal utilizing the full dynamic capabilities of the quadrotor.

B. Solution Outline

While it may be possible to compute quadrotor trajectories using existing methods of sampling-based kinodynamic planning or trajectory optimization, we find that these methods are inefficient or impractical in complex real-world environments for the reasons listed above. Instead, our solution to

this problem is to utilize the RRT* algorithm to find a route through the environment, initially ignoring the dynamics of the vehicle. That route is pruned to a sequence of waypoints representing the optimal route through the visibility graph of the environment. Then, a sequence of polynomial segments is jointly optimized to join those waypoints into a smooth minimum-snap trajectory from start to goal. Utilizing a differentially flat model of the quadrotor and the associated control techniques, we can follow these paths precisely.

The paper proceeds as follows. We first discuss the differentially flat quadrotor model and its implications for planning and polynomial trajectories. We then present a closed-form solution to the quadratic program (QP) used to obtain the polynomial trajectory that is numerically stable for both high-order polynomials and large numbers of segments. For comparison with purely sampling-based approaches, we compare our process with an RRT* algorithm that uses polynomial segments to grow a tree of candidate trajectories (i.e., as its *steer function* to connect sampled points in state space). We show that our process returns superior paths in much shorter running time. Finally, we highlight the performance of our QP formulation and show the results of flight tests in real-world environments.

II. QUADROTOR DYNAMICS AND CONTROL

In order to guarantee that we can precisely follow the polynomial trajectories we intend to generate, we utilize the property of differential flatness for the standard quadrotor equations of motion:

$$m\ddot{\mathbf{r}} = mg\mathbf{z}_W - f\mathbf{z}_B \tag{1}$$

$$\dot{\omega} = J^{-1}\left[-\omega \times J\omega + M\right] \tag{2}$$

Differential flatness of this model was demonstrated in [6]. Here, $\mathbf{r}$ is the position vector of the vehicle in a global coordinate frame, ω is the angular velocity vector in the body-fixed coordinate frame and f and M are the net thrust and moments in the body-fixed coordinate frame. J and m are the inertial tensor and mass of the quadrotor. $\mathbf{z}_B$ is the unit vector aligned with the axis of the four rotors and indicates the direction of thrust, while $\mathbf{z}_W$ is the unit vector expressing the direction of gravity. There exists a simple mapping from f and M to the four desired motor speeds.

A polynomial trajectory segment is in fact three polynomial functions of time specifying the independent evolution of the so-called *flat output* variables, x, y, and z between two positions in 3D space. The nonlinear controller employed to follow differentiable trajectories was developed in [7], and consists of independent calculations for thrust and moments:

$$f = (-k_x e_x - k_v e_v + mg\mathbf{z}_W + m\ddot{\mathbf{r}}_d) \cdot R\mathbf{z}_w \tag{3}$$

$$\begin{aligned} M = &-k_R e_R - k_\omega e_\omega + \omega \times J\omega \\ &- J(\hat{\omega}R^T R_d \omega_d - R^T R_d \dot{\omega}_d) \end{aligned} \tag{4}$$

where e_x,e_v,e_R, and e_ω are the error vectors in position, velocity, orientation and angular velocity, k_x,k_v,k_R, and k_ω are associated control gains, and R is the rotation matrix representing the orientation of the quadrotor.

This control system has several unique features that overcome the limitations of linear models and feedback-only control design. First, it uses a rotation matrix to express orientation without small-angle approximations, allowing large deviations from the horizontal hover regime. Second, the controller includes feed-forward terms that supply the appropriate force and moment, computed analytically from the derivatives at every point along the trajectory. Similar to model-predictive approaches, feed-forward control eliminates the lag incurred by following a moving set point using feedback alone. A transport map is used to map feed-forward commands appropriately from the desired body frame into the true body frame if the quadrotor deviates from the desired trajectory during flight. Finally, proportional-derivative (PD) control terms are included to stabilize the position and orientation about the moving reference along the trajectory.

The intuition behind differential flatness lies in the fact that the quadrotor must always align its axis of thrust with the total acceleration vector prescribed at every point along the trajectory, thus determining its exact orientation and required control inputs. In principle, if the model were a perfect representation of the dynamics, and in the absence of disturbances, the feed-forward control effort would carry the quadrotor precisely along 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 3-4), it effectively serves as a simulation of the vehicle's motion. This is the powerful feature of the quadrotor's differentially flat model that eliminates the need for accurate numerical integration of equations of motion, or a search over the space of inputs during each iteration of the planning algorithm.

III. POLYNOMIAL TRAJECTORY OPTIMIZATION

We now develop an analytical method for generating minimum-snap polynomial trajectories to be followed by a quadrotor using the control techniques outlined above. We assume that we have obtained a sequence of waypoints in 3D space representing the optimal path through the visibility graph of the environment, and we wish to generate a minimum-snap polynomial path passing through each of those waypoints. For this purpose, we use a simple RRT* algorithm to obtain the optimal straight-line path from start to goal, and then select waypoints from that optimal path according to a line-of-sight technique. Figure 3b shows the sequence of waypoints obtained by this method.

The choice of polynomial trajectories is natural for highly dynamic vehicles and robots since these trajectories can be obtained efficiently as the solution to a quadratic program that minimizes a cost function of the path derivatives. This optimization framework allows the endpoints of path segments to be optionally fixed to desired values or left free, and the polynomials can be jointly optimized while maintaining continuity of the derivatives up to arbitrary order. Maintaining continuity of derivatives ensures smooth motions and can be used to generate trajectories that do not require step inputs to the vehicle's actuators.

Polynomial trajectories allow for a analytical solution via elimination as a constrained QP [8]. While this method is acceptable for joint optimization of a few segments, it involves the inversion of matrices that may be very close to singular, along with high sensitivity to coefficients on the order of 10^{-20} or smaller, leading to inaccurate results. We present this constrained QP solution here and then use it in the following section as the basis for an unconstrained QP reformulation, which is robust to numerical instability.

For the following derivations, we require that the vector of segment times is fixed. That is, we require an *a priori* selection of the amount of time required to traverse between one waypoint and the next. These times can be selected approximately based on a desired average speed of the vehicle, however in general an arbitrary selection of times will not yield the lowest-cost solution. Therefore, we relax this assumption in a subsequent section where we iteratively refine the vector of times.

A. Cost Function for Minimizing Derivatives

For quadrotors, a single trajectory segment between two points in state space is composed of independent polynomial trajectories for the flat output variables x, y, z and yaw angle. Following the formulation in [9], each polynomial segment is represented as:

$$P(t) = p_n t^N + P_{n-1} t^{N-1} + \cdots + p_0 = \sum_{n=0}^{N} p_n t^n \tag{5}$$

The cost function for optimization of each polynomial is:

$$J = \int_0^T c_0 P(t)^2 + c_1 P'(t)^2 + c_2 P''(t)^2 + \ldots + c_N P^{(N)}(t)^2 dt \tag{6}$$

where T is the traversal time for the trajectory segment. To solve for a minimum-snap trajectory, c_4 would be nonzero while all other coefficients would be set to zero. This function can be written in matrix form as:

$$J = \bar{p}^T Q \bar{p} \tag{7}$$

where $\bar{p}$ is a vector of polynomial coefficients and Q is a cost matrix constructed as the weighted sum of Hessian matrices for each of the polynomial derivatives. These Hessian matrices will be derived by writing the cost function in terms of the polynomial coefficients and then differentiating twice with respect to those coefficients. Since we are optimizing the integral of squares of derivatives, we begin by writing the square of the polynomial as a convolution sum:

$$(P^2)_n = \sum_{j=0}^{N} p_j p_{n-j} \tag{8}$$

where $(P^2)_n$ is the n$^{\text{th}}$ coefficient of the squared polynomial. The r$^{\text{th}}$ derivative of a polynomial is:

$$P^{(r)}(t) = \sum_{n=r}^{N} \left(\prod_{m=0}^{r-1} (n-m) \right) p_n t^{n-r} \tag{9}$$

Hence, the component of the cost function associated with the r$^{\text{th}}$ derivative is:

$$J_r = \int_0^T P^{(r)}(t)^2 dt \tag{10}$$

$$= \sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m)(n-j-m) \right) p_j p_{n-j} \frac{T^{n-2r+1}}{n-2r+1} \tag{11}$$

Computation of the Hessian begins by differentiating J_r.

$$\begin{aligned} \frac{\partial J_r}{\partial p_i} =& \sum_{n=0}^{2N} \sum_{j=0}^{N} \left(\prod_{m=0}^{r-1} (j-m)(n-j-m) \right) \\ & \cdot \left(\frac{\partial p_j}{\partial p_i} p_{n-j} + \frac{\partial p_{n-j}}{\partial p_i} p_j \right) \frac{T^{n-2r+1}}{n-2r+1} \end{aligned} \tag{12}$$

$$\begin{aligned} =& 2 \sum_{n=0}^{2N} \left(\prod_{m=0}^{r-1} (i-m)(n-i-m) \right) \\ & \cdot p_{n-i} \frac{T^{n-2r+1}}{n-2r+1} \end{aligned} \tag{13}$$

Then differentiating again with respect to each of the polynomial coefficients yields:

$$\begin{aligned} \frac{\partial^2 J_r}{\partial p_i \partial p_l} =& 2 \sum_{n=0}^{2N} \left(\prod_{m=0}^{r-1} (i-m)(n-i-m) \right) \\ & \cdot \frac{\partial p_{n-i}}{\partial p_l} \frac{T^{n-2r+1}}{n-2r+1} \end{aligned} \tag{14}$$

$$= 2 \left(\prod_{m=0}^{r-1} (i-m)(l-m) \right) \frac{T^{i+l-2r+1}}{i+l-2r+1} \tag{15}$$

Finally,

$$Q_r^{il} = \begin{cases} 2 \left(\prod_{m=0}^{r-1} (i-m)(l-m) \right) \frac{T^{i+l-2r+1}}{i+l-2r+1} & \text{if } i \geq r \wedge l \geq r \\ 0 & \text{if } i < r \vee l < r \end{cases} \tag{16}$$

where the complete cost matrix Q is given by:

$$Q = \sum_{r=0}^{N} c_r Q_r \tag{17}$$

and c_r is the user-specified penalty on the r$^{\text{th}}$ derivative of the trajectory.

B. Constraints

The constraints on the the endpoints of a polynomial segment, which are used to either fix a given derivative to a desired value or ensure continuity of free derivatives, are imposed as a linear function of the coefficients:

$$A\bar{p} - b = 0 \tag{18}$$

$$A = \begin{bmatrix} A_0 \\ A_T \end{bmatrix}, \quad b = \begin{bmatrix} b_0 \\ b_T \end{bmatrix} \tag{19}$$

where A is constructed by evaluating the component of the derivative in equation 9 corresponding to the the appropriate

coefficient:

$$A_{0_{rn}} = \begin{cases} \prod_{m=0}^{r-1}(r-m) & \text{if } r = n \\ 0 & \text{if } r \neq n \end{cases} \tag{20}$$

$$b_{0_r} = P^{(r)}(0) \tag{21}$$

$$A_{T_{rn}} = \begin{cases} \left(\prod_{m=0}^{r-1}(r-m)\right) T^{n-r} & \text{if } n \geq r \\ 0 & \text{if } r < n \end{cases} \tag{22}$$

$$b_{T_r} = P^{(r)}(T) \tag{23}$$

These constraints either fix the position, velocity, acceleration, and higher order derivatives to desired values, or allow them to float subject to minimization of the cost function. Having assembled Q, A and b, the QP can now be written:

$$\min_{\bar{p}} \quad \bar{p}^T Q \bar{p} \tag{24}$$

$$\text{s.t.} \quad A\bar{p} - b = 0 \tag{25}$$

where the decision variables are the coefficients of the polynomial trajectory. This optimization problem can be solved in a straightforward manner using an elimination approach. The joint optimization of multiple segments is accomplished by concatenating individual segment optimization problems.

C. Reformulation as an Unconstrained QP

While the method above works well for single segments and small joint optimization problems, the matrices involved quickly become ill-conditioned for larger joint optimization problems of more than a few segments, for polynomials of higher order, and for problems in which the traversal time for segments varies widely. To avoid ill-conditioning, we reformulate the problem as an unconstrained QP to solve for endpoint derivatives directly as the decision variables, rather than the indirect method of solving for polynomial coefficients. In practice, this method has proven substantially more stable than the method above, allowing the joint optimization of at least 50 polynomial segments in a single matrix operation without encountering numerical issues. Once the optimal waypoint derivatives are found, the minimum-order polynomial connecting each pair of waypoints can be obtained by inverting the appropriate constraint matrix.

Beginning with the original cost function:

$$J = \bar{p}^T Q \bar{p} \tag{26}$$

we utilize the A matrix as a mapping between polynomial coefficients and the endpoint derivatives:

$$\bar{d} = A\bar{p} = \begin{bmatrix} A_0 \\ A_T \end{bmatrix} \bar{p} \tag{27}$$

and therefore the cost function for a single polynomial segment becomes:

$$J = \bar{d}^T A^{-T} Q A^{-1} \bar{d} \tag{28}$$

At this stage it is convenient to discuss the joint optimization problem as the general formulation encompassing the solution of a single polynomial segment. Given an initial state, a final state, and a sequence of intermediate waypoint locations, we wish to find the waypoint velocities, accelerations, and higher order derivatives such that the minimum-order polynomials connecting those waypoints will minimize the cost function above.

In the case of a joint optimization, we construct a Q_{joint} and A_{joint} matrices, which are simply block diagonal matrices composed of the Q and A matrices for the individual segment subproblems. The derivatives involved in the joint optimization are concatenated into a vector D. Typically, D will include a full set of derivatives to be fixed at the beginning and end of the trajectory (i.e., begin and end with zero velocity, acceleration, etc.) along with the derivatives to be fixed at the waypoints (i.e. positions), however this formulation is easily adapted to fix or float any of the derivatives. We sort D into a block of derivatives to be fixed in the optimization (D_F) and a block of free derivatives we intend to optimize (D_P).

$$D = \begin{bmatrix} D_F \\ D_P \end{bmatrix} \tag{29}$$

We then rely on a selector matrix M to map the derivatives in D to an arrangement that is consistent with the sequence of block-diagonal elements in A_{joint}. In particular, since each block-diagonal element of A_{joint} represents the optimization of a single segment, the M matrix also serves to duplicate each intermediate waypoint derivative value to appear both at the end of one segment and at the beginning of the subsequent segment, therefore maintaining continuity of the derivatives.

We now have the following total cost function for the joint optimization:

$$J = \begin{bmatrix} D_F \\ D_P \end{bmatrix}^T M A^{-T} Q A^{-1} M^T \begin{bmatrix} D_F \\ D_P \end{bmatrix} \tag{30}$$

Let R denote the new augmented cost matrix:

$$R = M A^{-T} Q A^{-1} M^T \tag{31}$$

Note that by formulating the problem with the waypoint derivatives as the decision variables, our constraints are now embedded within the cost function yielding an unconstrained optimization problem.

We proceed by partitioning R according to the number of fixed and free derivatives and then expanding the cost function:

$$R = \begin{bmatrix} R_{FF} & R_{FP} \\ R_{PF} & R_{PP} \end{bmatrix} \tag{32}$$

$$J = D_F^T R_{FF} D_F + D_F^T R_{FP} D_P + D_P^T R_{PF} D_F + D_P^T R_{PP} D_P \tag{33}$$

where the first term is simply a fixed cost incurred by satisfying the fixed derivatives. Differentiating J with respect D_P and equating to zero yields the optimal values for the free derivatives:

$$D_P^* = -R_{PP}^{-1} R_{FP}^T D_F \tag{34}$$

These optimal waypoint derivatives imply the minimum-order polynomial segments needed to construct the complete trajectory.

D. Time Allocation

Until this point in our optimization process, we have been required to fix an arbitrary amount of time associated with each segment in the complete trajectory since these times factor into the construction of the cost matrix. These segment times act as constraints on the solution quality, but can be allowed to vary to improve the overall solution. We therefore choose initial segment times and then iteratively refine the times in order to obtain better paths with respect to the cost function. We propose a simple extension of the polynomial cost function to choose segment times and thus determine the total trajectory traversal time. We attempt to minimize:

$$J_T = \bar{p}^T Q \bar{p} + k_T T \tag{35}$$

where T is the sum of segment times for the complete path and k_T 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 small by increasing the total time. Therefore, this modified cost performs a trade off between minimizing the polynomial cost and traversing the path quickly. Increasing the penalty on time, k_T, results in more aggressive trajectories.

We optimize the modified cost function via gradient descent where the gradient is estimated numerically by perturbing each segment time by some δt as in [6], however we do not require that the total path time be conserved in a given perturbation. Therefore, the path time will grow or diminish as necessary to optimize the modified cost. Figure 2 shows optimized trajectories for the same set of waypoints using two different k_T values. The red arrows indicate waypoint velocities while the green arrows indicate accelerations, and these quantities are greater in the bottom trajectory due to the higher time penalty. The quadrotor axes are plotted at $0.1s$ increments along the path. Notice also the emergent property resulting from time allocation that the quadrotor moves very slowly around the sharp corner (as shown by the lengths of the red velocity vectors), but it smoothly accelerates up to a higher speed in the straightaway where it does not incur a severe penalty on its fourth derivative.

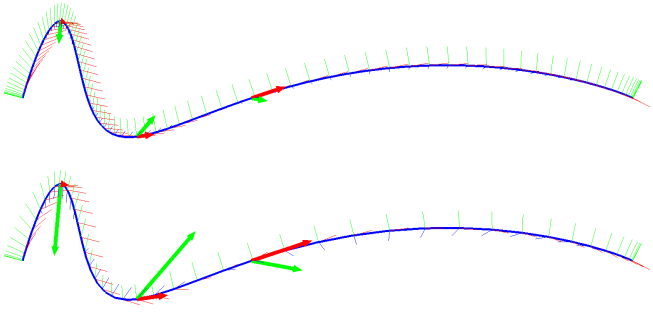


Fig. 2: Segment time optimization with the penalty on time k_T set at 500 (top) and 50000 (bottom). The optimal total trajectory times are $9.1s$ and $5.1s$ respectively. Vectors for waypoint velocity (red) and acceleration (green) are shown.

E. Ensuring Feasibility

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. This midpoint is known to be collision-free because it lies on the optimal route through the visibility graph. These additional waypoints are added incrementally until the polynomial trajectory is free of collision. A similar technique is used in [10].

IV. RESULTS

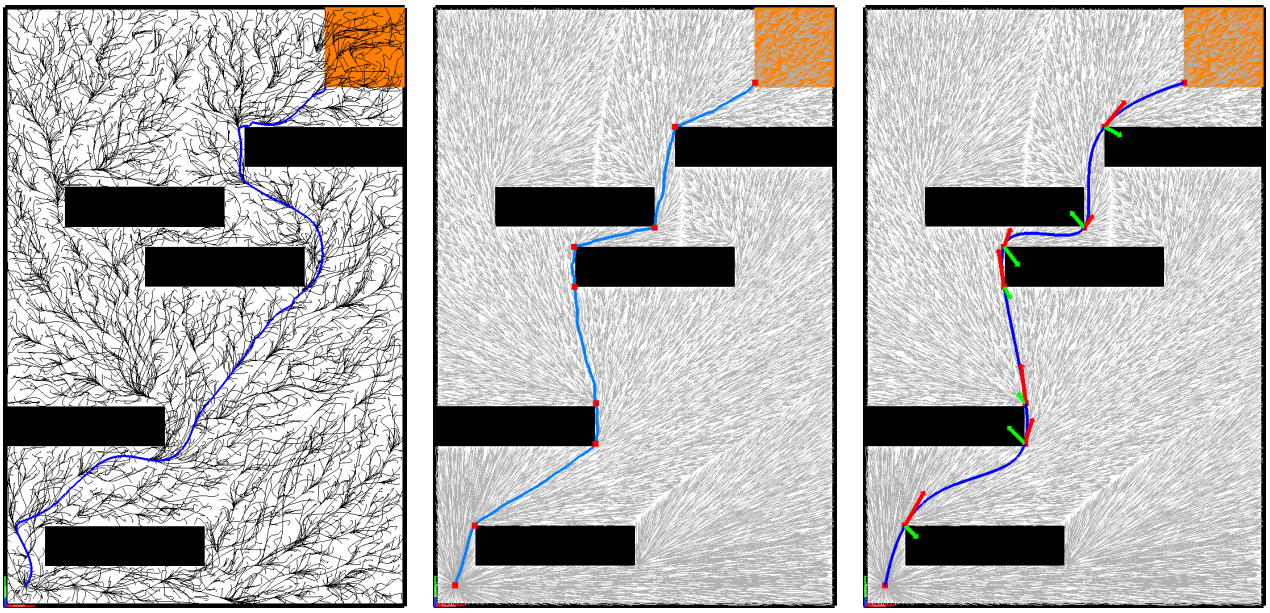
We have tested our trajectory generation process in a variety of environments. Figures 1 and 3 show solutions to challenging 2D and 3D problems. The use of a minimal set of waypoints and the joint polynomial optimization described above yields paths that are typically composed of natrual high-speed arcs in unconstrained regions of the environment while slowing in tight spaces to minimize accelerations around sharp corners. Our process sacrifices convergence to global optimality, but returns superior paths in much shorter running times than a purely sampling-based approach.

A. Comparison with RRT* using Polynomial Steer Function

For comparison to a strictly sampling-based planning approach, we implemented an RRT* algorithm using polynomial segments as the steer function to extend the search tree. Figure 3a shows the resulting solution. Sampling was performed in position and velocity space. We use the distance metric described in [11] 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 as discussed above, 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 I shows several statistics on the performance of the RRT* with a 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. The path generated by the RRT* is longer in distance and takes longer to traverse while having a much higher cost according to the objective function of polynomial optimization. The high cost is due to the unnecessary accelerations and higher derivatives incurred along the trajectory, since these derivatives are penalized in the cost function. When sampling in the full state space of the system, the RRT* with a polynomial steer function would

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

	Runtime	$J_{poly.}$	T_{path}	L_{path}
RRT*/Poly.	120s	5.72×10^8	21.94s	40.35m
Our Process	3s	1.07×10^5	19.66s	35.51m



(a) RRT* with polynomial steer function after 120s running time.

(b) Pruned waypoints from RRT* with straight-line steer function.

(c) Solution by our algorithm after 3s running time.

Fig. 3: 2D demonstration of the algorithm for comparison. The natural approach of using polynomial segments directly as a 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 match our objective function. We therefore jointly optimize a set of polynomials through those waypoints to obtain a minimum-snap path (c).

converge to a globally optimal solution in the limit of infinite samples, however as shown here, the paths returned prior to convergence are clearly of lower quality than those returned by our algorithm in a much shorter running time.

B. 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 II. We observe a significant improvement in speed performing calculations in C++ using the linear algebra library Eigen [12], especially compared with a baseline comparison to the MATLAB QP solver `quadprog.m`. 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. 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^{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 III. To measure the robustness of both C++ implementations, we solved a batch of 20 randomized polynomial optimization problems for each method, since numerical instability can be triggered by particular combinations of time scales and waypoint locations. In these tests, 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 even for higher order polynomials. In many indoor environments, the desired path can be accomplished in fewer than 15 segments, which is well within the range of stability for the unconstrained formulation. 9^{th} order polynomials are used because they have 10 coefficients, which is the minimum order needed to ensure continuity of 0^{th} through 4^{th} derivatives at the beginning and end of every segment. The unconstrained formulation is also stable for polynomials of 15^{th} order and higher. Finally, since A^{-1} and Q are sparse block-diagonal and M is sparse, these problems can be easily implemented using a sparse formulation which is roughly an order of magnitude faster than the dense computation for 10-segment joint optimizations.

TABLE II: 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

TABLE III: Numerical Stability of Optimization Techniques

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

C. Experimental Flight Tests

We demonstrate the performance of our algorithm on a challenging real-world planning problem by generating and flying a trajectory through a complex indoor lab space in the Stata Center (MIT). The environment used for these tests was a lab space with curved, non-vertical walls, interior columns and barriers aligned at oblique angles. 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 $0.65m$ to leave room for the $0.35m$ radius of the vehicle and a minimal allowance for error in estimation and control. Estimation and control were performed completely onboard the AscTec Pelican aircraft, using a Hokuyo LIDAR, a Microstrain IMU and an Intel Atom processor. The trajectory returned by our algorithm is shown in figure 4, and was generated in roughly $3s$. Figure 5 shows a frame from the onboard video taken during flight, illustrating the complex structure of environment. We have generated many trajectories through other interior spaces, typically requiring only seconds of computation time, and we have flown these trajectories successfully at speeds up to $8m/s$.

Fig. 5: Onboard video frame from quadrotor flight.

V. RELATED WORK

The literature on motion planning for autonomous robots and vehicles is extensive, and has considered both simple holonomic systems as well as those with differential constraints. Randomized algorithms such as probabilistic road maps (PRM), rapidly-exploring random trees (RRT) and RRT* have enjoyed great success in the last decade in part due to their simplicity and performance in high-dimensional state spaces [13], [14], [15]. These algorithms perform the function of filling the configuration space or state space with random samples and connecting them to find feasible or optimal paths from start to goal. The Euclidean distance metric reduces the nearest-neighbor search from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$ complexity through the use of the k-d tree structure. The performance of these algorithms on systems which do not follow straight-line paths depends partially on the extent to which the Euclidean distance metric remains reasonable.

Many sampling-based algorithms have also been demonstrated for motion planning under differential constraints [16]. These algorithms often perform very well when there exist simple analytical techniques for obtaining a steer function from one vertex in state space to the next, combined with an efficient calculation of the distance or cost of traversing that path [5]. However, for general dynamical systems, the key challenge is to find a feasible sequence of control inputs that will drive the system from one vertex toward another while respecting input limits and environmental constraints. Finding these inputs requires either a solution to the two-point boundary value problem for a system of nonlinear ordinary differential equations, or a process of iteratively simulating the vehicle dynamics [11]. Searching over possible control inputs is computationally costly and dramatically reduces the performance of the algorithms. Furthermore, the nearest vertex according to a Euclidean distance metric is not, in general, the vertex that will yield an optimal (or even feasible) path to a new sample in state space [17]. Nevertheless, sampling-based methods have proven successful in real-world applications to motion planning of vehicles with non-trivial dynamics [18].

Many successful methods exist for optimizing trajectories between two states of a dynamical system [19], and have been successfully applied to quadrotor control [20]. In trajectory optimization, other basis functions such as a B-splines [10] and Legendre polynomials [21] have been used to avoid ill-conditioning, however these options preclude the efficient optimization method presented here. Finally, our method is not limited to quadrotor control, as there exist simple differentially flat representations of fixed-wing aircraft [22] and cars [23] among many other systems.

VI. CONCLUSIONS

We have presented an algorithm for generating trajectories for the differentially flat quadrotor model through complex real-world environments that is computationally much faster than solving the same problems using a pure sampling approach, though at the expense global optimality. We observe that in this domain it is infeasible to rely on the limit of infinite sampling to produce smooth paths, and instead run a sampling-based motion planner as a route finder, followed by an optimization step in which the straight-line route is translated into a smooth dynamically feasible polynomial trajectory. We then iteratively refine the polynomial trajectory by a time allocation scheme that naturally performs a trade off to minimize accelerations while attempting to fly at a

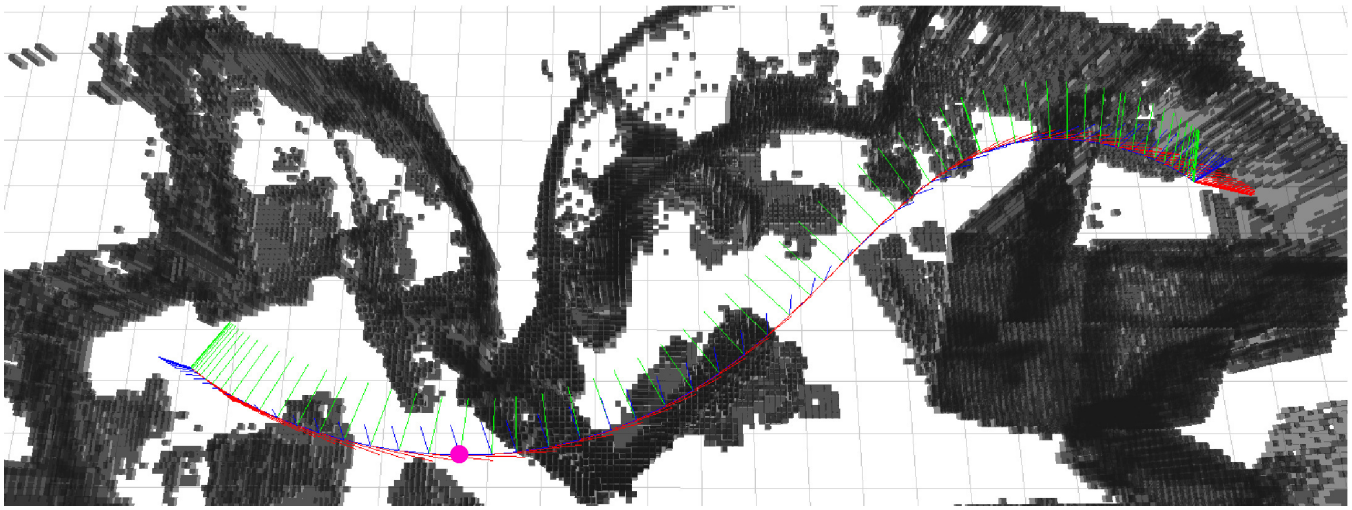


Fig. 4: Automatically generated trajectory through a map of a laboratory environment in the Stata Center, MIT. The magenta dot indicates the location of the onboard photo in figure 5.

desired velocity. This method is applicable to a large class of differentially flat models and approximate models of fixed-wing aircraft, automobiles and other systems.

VII. FUTURE WORK

The straight-line RRT* algorithm returns the shortest path from start to goal, however this path may not be the most desirable path to follow given the dynamics of the vehicle. When navigating cluttered environments with tight corners and narrow passageways, it is often preferable to navigate a smooth path through open space even if it results in a longer overall route. One extension of this work will be to generate trajectories through all homotopy classes to obtain the true optimal solution. Another valuable extension will be to generalize this method to allow the movement of waypoints to positions yielding lower cost paths.

REFERENCES

[1] J. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *Control Systems, IEEE*, vol. 28, no. 2, pp. 51 –64, april 2008.

[2] G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter trajectory tracking control," in *Proceedings of the IEEE Conference on Decision and Control (CDC 2008)*, 2008.

[3] M. Hehn and D. Raffaello, "Quadrocopter trajectory generation and control," in *International Federation of Automatic Control (IFAC), World Congress 2011*, 2011.

[4] A. Bry, A. Bachrach, and N. Roy, "State estimation for aggressive flight in gps-denied environments using onboard sensing," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2012)*, St Paul, MN, 2012.

[5] Karaman and Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, December 2010.

[6] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, may 2011, pp. 2520 –2525.

[7] T. Lee, M. Leoky, and N. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, dec. 2010, pp. 5420 –5425.

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

[9] A. Bry, "Control, estimation, and planning algorithms for aggressive flight using onboard sensing," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2012.

[10] J. Pan, L. Zhang, and D. Manocha, "Collision-free and smooth trajectory computation in cluttered environments," *The International Journal of Robotics Research*, 2012.

[11] Jeon, Karaman, and Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*," in *IEEE Conference on Decision and Control (CDC)*, 2011.

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

[13] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566 –580, aug 1996.

[14] S. M. LaValle, J. J. Kuffner, and Jr., "Rapidly-exploring random trees: Progress and prospects," 2000.

[15] Karaman and Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[16] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[17] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, may 2009, pp. 2859 –2865.

[18] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *Control Systems Technology, IEEE Transactions on*, vol. 17, no. 5, pp. 1105 –1118, sept. 2009.

[19] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[20] R. Ritz, M. Hehn, S. Lupashin, and R. D'Andrea, "Quadrocopter performance benchmarking using optimal control," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, sept. 2011, pp. 5179 –5186.

[21] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, may 2012, pp. 477 –483.

[22] J. Hauser and R. Hindman, "Aggressive flight maneuvers," in *in Proc. of the IEEE*, December 1997.

[23] R. M. Murray, M. Rathinam, and W. Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems," in *Proceedings of the 1995 ASME International Congress and Exposition*, 1995.