Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments

In Proceedings of the International Symposium of Robotics Research (ISRR 2013).

Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments

Charles Richter, Adam Bry, and Nicholas Roy

Abstract We explore the challenges of planning trajectories for quadrotors through cluttered indoor environments. We extend the existing work on polynomial trajectory generation by presenting a method of jointly optimizing polynomial path segments in an unconstrained quadratic program that is numerically stable for high-order polynomials and large numbers of segments, and is easily formulated for efficient sparse computation. We also present a technique for automatically selecting the amount of time allocated to each segment, and hence the quadrotor speeds along the path, as a function of a single parameter determining aggressiveness, subject to actuator constraints. The use of polynomial trajectories, coupled with the differentially flat representation of the quadrotor, eliminates the need for computationally intensive sampling and simulation in the high dimensional state space of the vehicle during motion planning. Our approach generates high-quality trajectories much faster than purely sampling-based optimal kinodynamic planning methods, but sacrifices the guarantee of asymptotic convergence to the global optimum that those methods provide. We demonstrate the performance of our algorithm by efficiently generating trajectories through challenging indoor spaces and successfully traversing them at speeds up to 8 m/s. A demonstration of our algorithm and flight performance is available at: `http://groups.csail.mit.edu/rrg/quad_polynomial_trajectory_planning`.

1 Introduction

Recent advances in small unmanned aircraft have enabled highly dynamic, aerobatic flight maneuvers [1, 9, 10, 21]. 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, 25]. However motion planning algorithms have not yet succeeded in joining these capabilities to enable quadrotors to navigate autonomously at high speeds using their full dynamic capabilities. This pa-

Charles Richter, Adam Bry, and Nicholas Roy
Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA 02139, e-mail: `car,abry,nickroy@mit.edu`

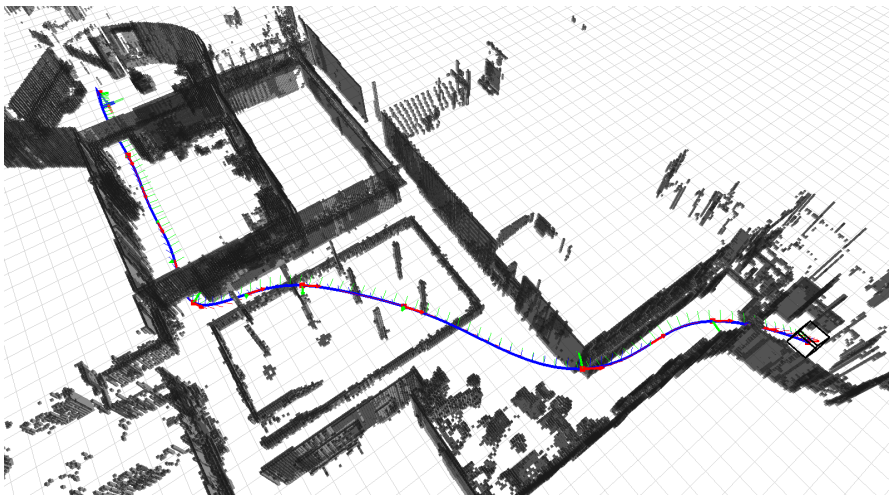


Fig. 1: Automatically generated 3D trajectory navigating a real-world environment with closely-spaced obstacles.

per addresses that need and provides a planning algorithm that enables autonomous, aggressive, high-speed quadrotor flight through complex indoor 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 [13]. 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 locally optimal paths for systems with general dynamics. However, these methods are also computationally intensive and may require accurate analytical representations of environmental constraints in order to efficiently compute cost gradients with respect to obstacles. These limitations make them impractical when constraints are represented in the form of an occupancy map.

Nevertheless, explicit optimization 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 fourth derivative, of the path [19]. Minimizing the snap of a trajectory quantifies a notion of gracefulness that is desir-

able for maintaining the quality of onboard sensor measurements as well as avoiding 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.

Our contribution is to extend the work of Mellinger, et al. [19], and show that their minimum-snap trajectory generation can be solved in a numerically stable unconstrained quadratic program (QP) for long-range trajectories composed of many segments. We show that this minimum-snap technique can be coupled with an appropriate kinematic planner to generate fast, graceful flight paths in cluttered environments, while accounting for collisions of the resulting polynomial trajectories. This combination of search and optimization significantly outperforms pure search-based planning methods in computational performance. Finally, we modify their strategy of allocating time along the trajectory to allow the planner to automatically adjust to widely varying size scales with a single user-set parameter on aggressiveness.

1.1 Problem Statement and Solution Outline

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

Our solution to this problem is to utilize the RRT* algorithm to find a collision-free path through the environment, initially considering only the kinematics of the vehicle and ignoring the dynamics. That 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. 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 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.

2 Quadrotor Dynamics and Control

In order to ensure 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 [19]. 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 consists of four polynomial functions of time specifying the independent evolution of the so-called *flat output* variables, x, y, z, and ψ (yaw angle) between two states in flat output space. The nonlinear controller employed to follow differentiable trajectories was developed in [18], and consists of independent calculations for thrust and moments:

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

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

where $\mathbf{e}_x$,$\mathbf{e}_v$,$\mathbf{e}_R$, and $\mathbf{e}_\omega$ 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.

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), these quantities serve as a simulation of the vehicle's motion in the absence of disturbances. This is the powerful capability enabled by differential flatness that eliminates the need for iterated numerical integration of equations of motion, or a search over the space of inputs during each iteration of the planning algorithm.

3 Polynomial Trajectory Optimization

We now describe 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 shortest piecewise-linear path through 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 6b 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 QP 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 [2]. 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 next and then use it in a 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.

3.1 Cost Function for Minimizing Derivatives

For quadrotors, a single trajectory segment between two points in flat output space is composed of independent polynomials, $P(t)$, for the flat output variables x, y, z and yaw angle. The cost function penalizing the squares of the derivatives of $P(t)$ can be written as:

$$J(T) = \int_0^T c_0 P(t)^2 + c_1 P'(t)^2 + c_2 P''(t)^2 + \ldots + c_N P^{(N)}(t)^2 dt = \mathbf{p}^T Q(T) \mathbf{p} \qquad (5)$$

In this expression, $\mathbf{p}$ is a vector of the N coefficients of a single polynomial. In order to minimize snap, all derivative penalties in the cost function except for c_4 would be set to zero. The construction of the Hessian matrix Q is omitted for brevity, but follows from differentiation of the square of the polynomial with respect to each of its coefficients. Since the cost of a given polynomial is a function of its duration T, we must fix T prior to optimization. M polynomial segments can be jointly optimized by concatenating their cost matrices in a block-diagonal fashion:

$$J_{total} = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} Q_1(T_1) & & \\ & \ddots & \\ & & Q_M(T_M) \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \qquad (6)$$

3.2 Constraints

Constraints in the polynomial optimization are imposed on the endpoints of each segment. These constraints allow the endpoints to be pinned to known locations in space, or assigned specific values of velocity, acceleration, jerk or snap. Such constraints are useful to enforce, for example, that the quadrotor start from rest at the beginning of a trajectory. Constraints on the i^{th} segment in a trajectory are formulated using a mapping matrix (A) between coefficients and endpoint derivatives of a polynomial:

$$A_i \mathbf{p}_i = \mathbf{d}_i, \quad A_i = \begin{bmatrix} A_0 \\ A_T \end{bmatrix}_i, \quad \mathbf{d}_i = \begin{bmatrix} d_0 \\ d_T \end{bmatrix}_i \tag{7}$$

where $\mathbf{d}_i$ is a vector containing the derivative values for the beginning (d_0) and end (d_T) of the i^{th} segment. If specific derivatives are not known, then continuity constraints must be imposed to ensure that the derivatives at the end of the i^{th} segment match the derivatives at the beginning of the $(i+1)^{th}$ segment:

$$A_{T,i} \mathbf{p}_i = A_{0,i+1} \mathbf{p}_{i+1} \tag{8}$$

These constraints can be compiled into a single set of linear equality constraints for the joint optimization problem:

$$A_{total} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix} \tag{9}$$

Standard methods can be used to solve the resulting constrained QP.

3.3 Reformulation as an Unconstrained QP

While the method above works well for single segments and small joint optimization problems as in [19], 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 to be practical for optimizing long range paths requiring many waypoints and segments.

We improve upon the solution above using a technique of substitution to convert the problem into an unconstrained QP, and solve directly for endpoint derivatives as decision variables, rather than solving for polynomial coefficients. In practice, our reformulation is substantially more stable than the method above, 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 minimum-order polynomial connecting each pair of waypoints can be obtained by inverting the appropriate constraint matrix.

We begin by substituting the constraints into the original cost function:

$$J = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix}^T \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_M \end{bmatrix}^{-T} \begin{bmatrix} Q_1 & & \\ & \ddots & \\ & & Q_M \end{bmatrix} \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_M \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_M \end{bmatrix} \tag{10}$$

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

$$J = \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}^T \underbrace{CA^{-T}QA^{-1}C^T}_{R} \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix} = \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix}^T \begin{bmatrix} R_{FF} & R_{FP} \\ R_{PF} & R_{PP} \end{bmatrix} \begin{bmatrix} \mathbf{d}_F \\ \mathbf{d}_P \end{bmatrix} \tag{11}$$

where we have written the block-diagonal matrices as A and Q for simplicity of notation. We group the new augmented cost matrix into a single matrix 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:

$$J = \mathbf{d}_F^T R_{FF} \mathbf{d}_F + \mathbf{d}_F^T R_{FP} \mathbf{d}_P + \mathbf{d}_P^T R_{PF} \mathbf{d}_F + \mathbf{d}_P^T R_{PP} \mathbf{d}_P \tag{12}$$

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:

$$\mathbf{d}_P^* = -R_{PP}^{-1} R_{FP}^T \mathbf{d}_F \tag{13}$$

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

3.4 Time Allocation

Until this point in the optimization, we have fixed an arbitrary amount of time 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: [5] minimizes total time subject to constraints, while [19] fixes the total time by hand and minimizes 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:

$$J_T = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} Q_1(T_1) & & \\ & \ddots & \\ & & Q_M(T_M) \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T + k_T \sum_{i=1}^{M} T_i \tag{14}$$

where 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 close to zero by increasing the total time, but equation (14) has a definite minimum value that varies with k_T. Figure 2 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.

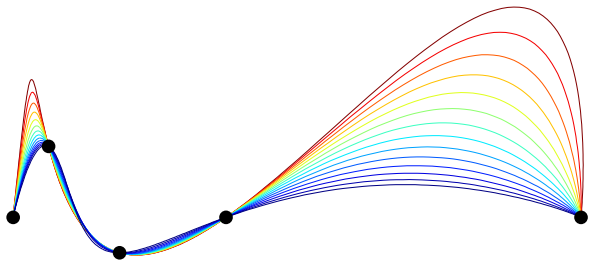


Fig. 2: Illustration of the iterative refinement of segment times, color-coded by total traversal time. The initial guess of total time is 10.5s (red) and the final optimized total time is 7s (blue).

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, k_T.

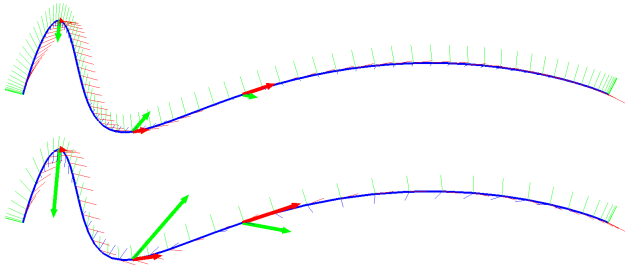
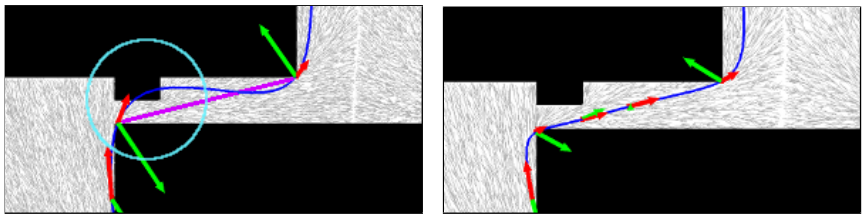


Fig. 3: Segment time optimization with the penalty on time k_T set at 500 (top) and 50000 (bottom). The optimal total trajectory times are 9.1s and 5.1s respectively. Vectors for waypoint velocity (red) and acceleration (green) are shown.

Figure 3 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. 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. One emergent property resulting from time allocation is that the quadrotor 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 snap. Furthermore, the geometric shape of the optimal trajectory remains the same regardless of the value of k_T, indicating that the minimum-snap *ratios* of segment times are independent of k_T.

3.5 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. 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 with the additional waypoint, and the process is repeated if necessary until the polynomial trajectory is collision free. A similar technique is used in [23]. Figure 4 illustrates this process successfully resolving a collision.

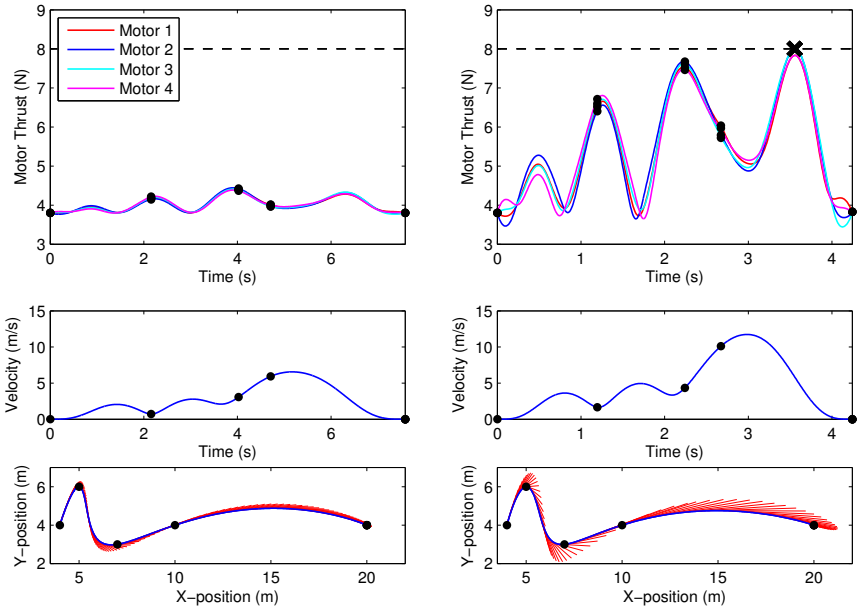


(a) Polynomial trajectory (blue) intersects an obstacle even though the underlying straight line between waypoints is collision-free.

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

Fig. 4: (a) The polynomial (blue) intersects an obstacle even though the line between waypoints is collision free (magenta). These scenarios are resolved by iteratively adding waypoints along the collision-free path returned by the search algorithm (b).

In very dense environments, trajectories may need many additional waypoints to repair collisions, thus requiring the optimization problem to be re-solved many times to find a feasible solution. Furthermore, additional waypoints increase the computational complexity of the QP being solved in each iteration. However, in our experience with indoor environments, the number of additional waypoints required to repair collisions was usually less than half of the original number of waypoints in the trajectory, representing only a modest increase in computational complexity.



(a) Motor commands for a conservative time allocation, barely exceeding the 3.8 N per motor required for hover (top). Velocity along the trajectory is low (middle). Trajectory with velocity vectors is shown for reference, with black dots indicating waypoints (bottom).

(b) Motor commands for aggressive time allocation, with one motor command reaching the maximum available thrust, indicated by the 'X' (top). Velocity along the trajectory is high (middle). Trajectory with larger velocity vectors is shown for reference(bottom).

Fig. 5: Comparison between two time allocations during the gradient descent procedure. The first time allocation (a) is conservative in that it is a slower trajectory than the second one (b), which reaches one of the actuator constraints during the final deceleration.

3.6 Actuator Constraints

The second major factor contributing to feasibility is to ensure that the input constraints of the quadrotor are satisfied such that no portion of the commanded trajectory requires a thrust outside the range that the motors 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 [6], 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 thrust. First, we observe that in the limit as $T \to \infty$, the quadrotor states along the trajectory converge

to hover, which is known to be feasible. 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 5 illustrates two different time allocations during the optimization of a sample trajectory. 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.

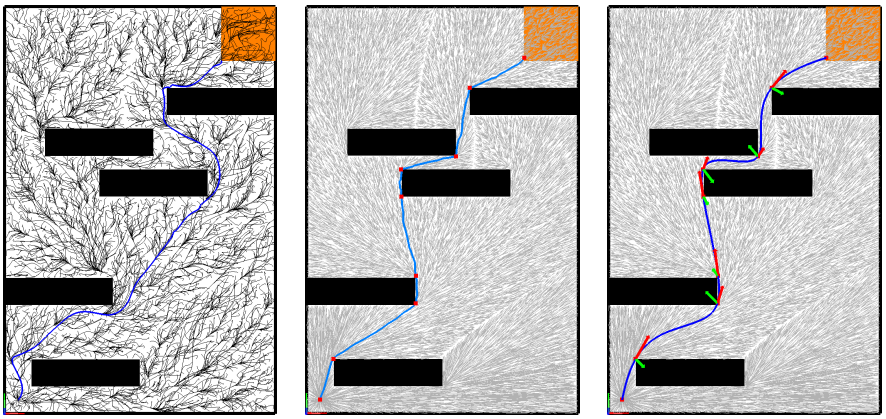
Due to the non-convexity of the feasible set in flat output space, the optimization algorithm may encounter an actuator limit and terminate before converging to the optimal ratio of segment times (for example, one of the red or orange lines in Figure 2). To avoid this scenario, one strategy is to first 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 3.4. 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.

4 Results

We have tested our trajectory generation process in a variety of environments. Figures 1 and 6 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 natural high-speed arcs in unconstrained regions of the environment while slowing in tight spaces to minimize snap around sharp corners. Our process sacrifices the guarantee of asymptotic convergence to a globally optimal solution provided by sampling-based approaches, but returns superior paths in much shorter running times than a purely sampling-based approach.

4.1 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 grow a search tree. Figure 6a shows the resulting solution. Sampling was performed in position and velocity space. We use the distance metric described by [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.



(a) RRT* with polynomial steer function terminated after 120s returns high-cost path.

(b) Pruned waypoints from straight-line RRT* become waypoints in 6c.

(c) Solution by our algorithm after 3s running time, finds much lower cost than 6a.

Fig. 6: 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).

Table 1: Comparison of our method with RRT* using the polynomial steer function for the 2D problem in figure 6.

Method	Runtime	$J_{poly.}$	T_{path}	L_{path}
RRT* with Polynomial Steer Function	120s	5.72×10^8	21.94s	40.35m
Low-Dim. Search + Unconstrained QP Optimization	3s	1.07×10^5	19.66s	35.51m

Table 1 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. 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.

4.2 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 2 and reflect MATLAB as well as C++/Eigen implementations [7]. 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 2: 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^{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 3, 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 3: 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%

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

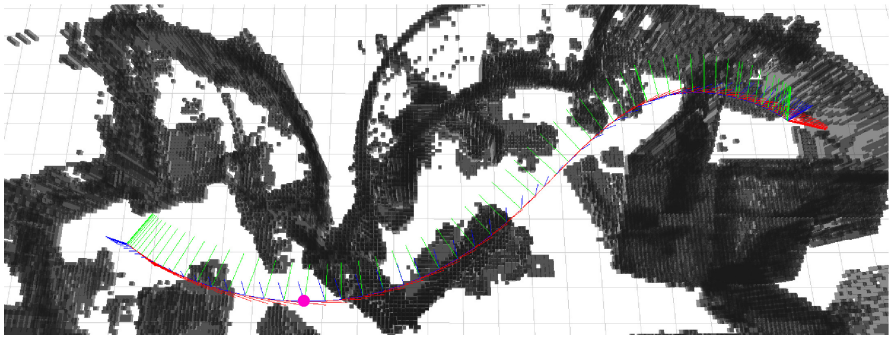


Fig. 7: Automatically generated trajectory through a map of a laboratory environment in the Stata Center, MIT.

4.3 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 trajectories returned by our algorithm are shown in Figures 1 and 7, and were generated in several seconds. These trajectories exhibit roughly $2m$ of altitude variation in order to fly through doorways and navigate over tall shelves and dividing walls. Figure 8 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`.

5 Related Work

The literature on motion planning for robots and vehicles is extensive, considering both simple holonomic systems as well as those with differential constraints. Randomized algorithms such as PRM, RRT and RRT* have enjoyed success due to their simplicity and performance in high-dimensional spaces [14, 17, 12].

Sampling-based algorithms have also been demonstrated for motion planning under differential constraints, which often perform very well when there exist simple analytical techniques for obtaining a steer function from one vertex in state space to the next [16, 13]. However, for general dynamical systems, steering between two states may require iteratively simulating the vehicle dynamics at a significant com-

putational cost [11]. 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 [26]. Nevertheless, sampling-based methods have proven successful in real-world applications to motion planning of vehicles with non-trivial dynamics [15].

Many methods exist for optimizing trajectories between two states of a dynamical system [3], and have been successfully applied to quadrotor control [24]. B-splines [23] and Legendre polynomials [20] have been used to avoid ill-conditioning in trajectory optimization problems, however these options preclude the efficient method presented here. Finally, our method is not limited to quadrotor control, as there exist simple differentially flat representations of fixed-wing aircraft [8] and cars [22] among many other systems.

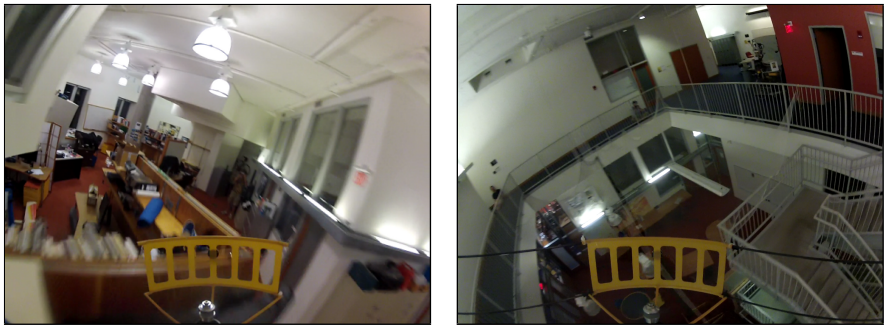


Fig. 8: Onboard video frames from aggressive quadrotor flight up to $8m/s$.

6 Conclusion

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 of global optimality. We observe that in this domain it is infeasible to rely on the limit of infinite sampling to perform optimization, and instead we perform low-dimensional search for route-finding followed by analytical optimization in which the shortest path is translated into a dynamically feasible polynomial trajectory. We then iteratively refine the polynomial trajectory by a time allocation procedure that trades off between time and snap of the path.

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

References

1. P. Abbeel, A. Coates, and A. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *Int. Journal of Robotics Research*, 29(13):1608–1639, 2010.
2. D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1999.
3. J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control and Dynamics*, 21(2):193–207, 1998.
4. A. Bry, A. Bachrach, and N. Roy. State estimation for aggressive flight in GPS-denied environments using onboard sensing. In *Proc. Int. Conf. on Robotics and Automation*, 2012.
5. Cutler, M. and How, J. Actuator constrained trajectory generation and control for variable-pitch quadrotors. In *Proc. AIAA Guidance, Navigation, and Control Conf.*, 2012.
6. N. Faiz, S. Agrawal, and R. Murray. Differentially flat systems with inequality constraints: An approach to real-time feasible trajectory generation. *Journal of Guidance, Control and Dynamics*, 24(2):219–227, 2001.
7. G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.
8. J. Hauser and R. Hindman. Aggressive flight maneuvers. In *Proc. Conf. on Decision and Control*, December 1997.
9. M. Hehn and R. D'Andrea. Quadrocopter trajectory generation and control. In *Int. Federation of Automatic Control, World Congress*, 2011.
10. J.P. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *Control Systems, IEEE*, 28(2):51–64, 2008.
11. J.H. Jeon, S. Karaman, and E. Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*. In *Conf. on Decision and Control*, 2011.
12. S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Proc. Robotics: Science and Systems*, 2010.
13. S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *Conf. on Decision and Control*, 2010.
14. L.E. Kavraki, et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.
15. Y. Kuwata, et al. Real-time motion planning with applications to autonomous urban driving. *Control Systems Technology, IEEE Transactions on*, 17(5):1105–1118, 2009.
16. S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Int. Journal of Robotics Research*, 20(5):378–400, 2001.
17. S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Workshop on Algorithmic Foundations of Robotics*, 2000.
18. T. Lee, M. Leoky, and N.H. McClamroch. Geometric tracking control of a quadrotor uav on se(3). In *Conf. on Decision and Control*, 2010.
19. D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proc. Int. Conf. on Robotics and Automation*, 2011.
20. D. Mellinger, A. Kushleyev, and V. Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *Proc. Int. Conf. on Robotics and Automation*, 2012.
21. D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Proc. Int. Symposium on Experimental Robotics*, 2010.
22. R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. In *Proc. ASME Int. Congress and Exposition*, 1995.
23. 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.
24. R. Ritz, M. Hehn, S. Lupashin, and R. D'Andrea. Quadrocopter performance benchmarking using optimal control. In *Proc. Int. Conf. on Intelligent Robots and Systems*, 2011.
25. S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar. Vision-based state estimation and trajectory control towards aggressive flight with a quadrotor. In *Proc. Robotics: Science and Systems*, 2013.
26. A. Shkolnik, M. Walter, and R. Tedrake. Reachability-guided sampling for planning under differential constraints. In *Proc. Int. Conf. on Robotics and Automation*, 2009.