Generating Dynamically Feasible Trajectories for Quadrotor Cameras

Mike Roberts
Stanford University

Pat Hanrahan
Stanford University

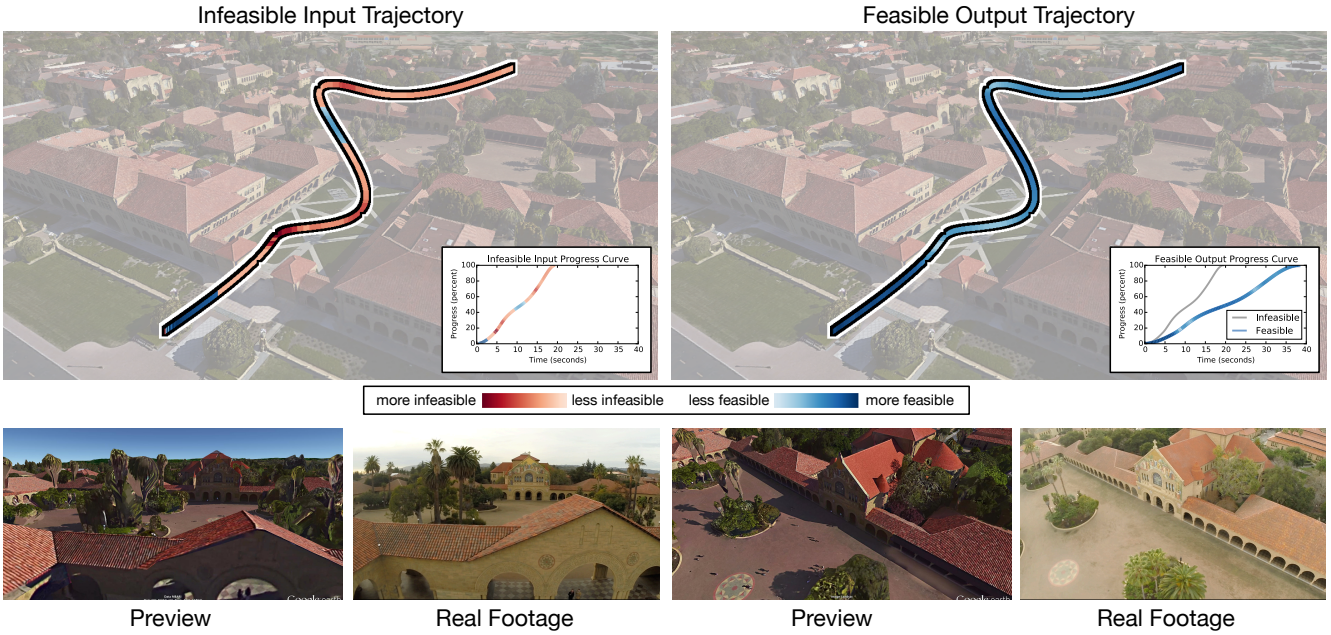


Figure 1: *Our algorithm for generating feasible quadrotor camera trajectories. Our algorithm takes as input a infeasible quadrotor camera trajectory (top row, left) and produces as output a feasible trajectory that is as similar as possible to the input trajectory (top row, right). By design, our algorithm does not change the spatial layout or visual contents of the input trajectory. Instead, our algorithm guarantees the feasibility of the output trajectory by re-timing the input trajectory, perturbing its timing as little as possible while remaining within velocity and control force limits (top row, insets). Our algorithm runs at interactive rates, solving for the feasible trajectory shown above in less than 2 seconds. Infeasible trajectories can be unsafe to fly on real quadrotor cameras, but we can safely fly the feasible trajectories generated by our algorithm, producing real video footage that is faithful to a* GOOGLE EARTH *shot preview (bottom row).*

Abstract

When designing trajectories for quadrotor cameras, it is important that the trajectories respect the dynamics and physical limits of quadrotor hardware. We refer to such trajectories as being *feasible*. In this paper, we introduce a fast and user-friendly algorithm for generating feasible quadrotor camera trajectories. Our algorithm takes as input an infeasible trajectory designed by a user, and produces as output a feasible trajectory that is as similar as possible to the user's input. By design, our algorithm does not change the spatial layout or visual contents of the input trajectory. Instead, our algorithm guarantees the feasibility of the output trajectory by *re-timing* the input trajectory, perturbing its timing as little as possible while remaining within velocity and control force limits. Our choice to perturb the timing of a shot, while leaving the spatial layout and visual contents of the shot intact, leads to a well-behaved non-convex optimization problem that can be solved at interactive rates.

We implement our algorithm in an open-source tool for designing quadrotor camera shots, where we achieve interactive performance across a wide range of camera trajectories. We demonstrate that our algorithm is between 25× and 45× faster than a spacetime constraints approach implemented using a commercially available solver. As we scale to more finely discretized trajectories, this performance gap widens, with our algorithm outperforming spacetime constraints by between 90× and 180×. Finally, we fly 5 feasible trajectories generated by our algorithm on a real quadrotor camera, producing video footage that is faithful to GOOGLE EARTH shot previews, even when the trajectories are at the quadrotor's physical limits.

Keywords: robotics, quadrotors, trajectory optimization

Concepts: •**Computing methodologies** → **Computer graphics; Motion path planning; Computational control theory;**

SIGGRAPH '16 Technical Paper, July 24 - 28, 2016, Anaheim, CA
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925980

1 Introduction

When designing trajectories for quadrotor cameras, it is important that the trajectories respect the dynamics and physical limits of quadrotor hardware. We refer to such trajectories as being *feasible*. If a quadrotor attempts to execute an *infeasible* trajectory, the quadrotor can deviate significantly from the intended trajectory, or even crash. In addition, the virtual camera previews shown in existing shot planning tools will be visually accurate only if a shot is feasible.

Despite the importance of reasoning about feasibility in the design process, existing tools do not provide automatic methods for guaranteeing feasibility. At best, existing tools *notify* users when their trajectories are infeasible, but offer no guidance on how to *modify* these trajectories to make them feasible. It can be challenging for users to manually edit their trajectories to be feasible, while preserving their original artistic intent. Indeed, requiring users to perform this type of manual editing can be quite burdensome, even for experienced users, since it requires users to reason explicitly about the non-linear quadrotor dynamics.

In this paper, we introduce a fast and user-friendly algorithm for generating feasible quadrotor camera trajectories. Our algorithm takes as input an infeasible trajectory designed by a user, and produces as output a feasible trajectory that is as similar as possible to the user's input. By design, our algorithm does not change the spatial layout or visual contents of the input trajectory. Instead, our algorithm guarantees the feasibility of the output trajectory by *re-timing* the input trajectory, perturbing its timing as little as possible while remaining within velocity and control force limits. Using our algorithm, a shot designer can modify a shot to be feasible with a single button click, completely preserving the visual contents of her shot. We demonstrate the behavior of our algorithm in Figure 1.

Our choice to perturb the timing of a shot, while leaving the spatial layout and visual contents of the shot intact, leads to a well-behaved non-convex optimization problem that can be solved at interactive rates. To formulate our problem, we begin by analyzing the non-linear dynamics of a rigid body quadrotor along a fixed path. Based on this analysis, we make two important observations that motivate our approach. First, we observe that the full state of the quadrotor, as well as all necessary control forces, are fully determined by the *progress curve* of the quadrotor along the path. Second, we observe that all sufficiently smooth progress curves satisfy the non-linear quadrotor dynamics.

Based on the two observations above, we explicitly optimize for the progress curve (and hence, the re-timing of the input trajectory) that best agrees with the user's original input, subject to velocity and control force limits. Compared to existing trajectory optimization approaches, our approach has three major advantages. First, our approach requires fewer decision variables. Second, our approach avoids slow-to-converge non-linear equality constraints that encode the system dynamics. Third, our approach always provides an iterative solver with a well-defined search direction to fall back on, when attempting to satisfy inequality constraints. Together, these advantages enable an off-the-shelf solver to make very rapid progress towards an optimal solution, ultimately leading to the interactive performance of our algorithm.

We demonstrate the utility of our algorithm by implementing it in HORUS [Joubert et al. 2015], an open-source tool for designing quadrotor camera shots, where we achieve interactive performance across a wide range of camera trajectories. We also apply our algorithm to a dataset of 8 infeasible camera trajectories, designed by 2 novice and 2 expert quadrotor cinematographers. In all our experiments, our algorithm successfully solves for optimal trajectories in less than 2 seconds, and is between 25× and 45× faster than a spacetime constraints approach implemented using a commercially available solver. As we scale to more finely discretized trajectories, this performance gap widens, with our algorithm outperforming spacetime constraints by between 90× and 180×. Our algorithm is also more accurate than spacetime constraints, predicting the results of 5^{th} order accurate rigid body physics simulations with lower average error. Finally, we fly 5 feasible trajectories generated by our algorithm on a real quadrotor camera, producing video footage that is faithful to GOOGLE EARTH shot previews, even when the trajectories are at the quadrotor's physical limits.

2 Related Work

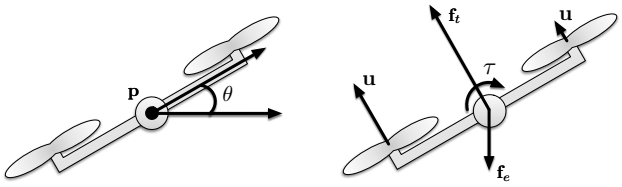
Design Tools for Quadrotor Camera Trajectories Several tools exist for designing quadrotor camera trajectories. These tools allow users to place waypoints on a 2D map [APM 2015; DJI 2015b] or in a 3D virtual environment [Meier et al. 2012; Joubert et al. 2015; Gebhardt et al. 2016]. Other tools allow users to interactively control the speed and orientation of a quadrotor camera as it flies along a pre-determined trajectory [3D Robotics 2015; DJI 2015a; Joubert et al. 2015]. However, most existing tools do not reason explicitly about the quadrotor dynamics, and therefore do not offer any assurance that the user's intended shot is feasible.

The shot planning tool introduced by Joubert et al. [2015] computes velocities and control forces along a user-specified trajectory, notifying the user if her shot is infeasible. However, this tool offers no guidance on how to modify the shot to make it feasible. The tool introduced by Gebhardt et al. [2016] optimizes quadrotor camera trajectories subject to velocity and control force limits, and is therefore capable of generating feasible trajectories. However, this tool uses an approximate linear model of the quadrotor dynamics, which is only accurate for conservative trajectories. In contrast, our algorithm reasons explicitly about the non-linear quadrotor dynamics, and is applicable to aggressive trajectories that are at a quadrotor's physical limits.

Using existing tools, a simple strategy for capturing an infeasible shot would be to uniformly time-stretch the shot until it is feasible, and then time-warp the resulting video footage back to the original timing. However, this strategy is only applicable in completely static scenes. For example, if a person is walking through the scene, then their walking motion will be time-warped, which may be visually jarring.

Trajectory Optimization Methods for Quadrotors The most common approach for optimizing quadrotor trajectories is to generate C^4 continuous splines that minimize some kind of bending energy [Mellinger and Kumar 2011; Bry et al. 2015; Deits and Tedrake 2015; Joubert et al. 2015]. These spline-based trajectories can be adapted to remain within velocity and control force limits by optimizing the time allocated to each spline segment [Mellinger and Kumar 2011; Bry et al. 2015]. However, this approach can result in overly conservative trajectories in cases where a long spline segment is only briefly infeasible. In contrast, our algorithm optimally re-times trajectories at the level of individual samples along a path, rather than at the level of spline segments, resulting in trajectories that more closely match a user's intended timing.

More general non-convex optimization methods have been applied to generate feasible quadrotor trajectories. Similar to our approach in this paper, some of these methods explicitly optimize a progress curve [Cowling et al. 2007; Bouktir et al. 2008; Van Loock et al. 2013]. However, these methods focus on finding the fastest or most fuel-efficient trajectory that reaches a goal, and typically also optimize the spatial layout of the trajectory. In contrast, we treat the spatial layout of the trajectory as fixed, and we focus on finding the



(a) Degrees of Freedom (b) Forces and Torques

Figure 2: *Overview of our quadrotor dynamics model, shown in 2D for simplicity. (a) Degrees of freedom. We model a quadrotor as having a position* $\mathbf{p}$ *and an orientation* θ*. (b) Forces and torques. We maneuver the quadrotor by applying a thrust force at each propellor, denoted with the vector* $\mathbf{u}$*. These thrust forces generate a net thrust force* $\mathbf{f}_t$ *and a net torque* τ *at the quadrotor's center of mass. The only other force acting on the quadrotor is an external force* $\mathbf{f}_e$*, which models effects like gravity, wind, and drag.*

progress curve that most closely matches a user's input.

Trajectory Optimization Methods in Computer Graphics and Robotics The problem of optimizing trajectories for dynamical systems has been studied extensively in the computer graphics literature [Geijtenbeek and Pronost 2012], and in the robotics literature [Betts 1998]. A common approach in both communities is to discretize a trajectory into a sequence of system states, and encode the system dynamics into an optimization problem as a set of equality constraints. This approach is known as *spacetime constraints* in the computer graphics community [Witkins and Kass 1988], and *direct collocation* in the robotics community [Betts 1998]. We refer the reader to Fang and Pollard [2003] and Safonova et al. [2004] for a detailed overview of spacetime constraints approaches in computer graphics. A well-known limitation of spacetime constraints is that it slow to converge for highly non-linear systems, such as quadrotors. In contrast, our approach avoids this slow convergence behavior, due to our analysis of the non-linear quadrotor dynamics along a fixed path.

Trajectory optimization problems along a fixed path have been studied in the computer graphics literature [McCann et al. 2006], and even more extensively in the robotics literature [Shin and McKay 1985; Slotine and Yang 1989; Dahl and Nielsen 1990; Verscheure et al. 2009; Lipp and Boyd 2014]. Existing methods require a path through a system's full *configuration space* as input, and optimize the speed of the system along this path. However, these methods are not applicable to quadrotors. This is because a quadrotor's orientation must match the direction in which it is accelerating. Therefore, it is generally not possible for a quadrotor to travel at different speeds along the same path through configuration space. In contrast, our algorithm is applicable to quadrotors. This is because we only require a path through *camera pose space* as input, and a quadrotor (equipped with an appropriate camera gimbal) is free to travel at different speeds along such a path.

Faulwasser et al. [2014] present a theoretical analysis of fixed path optimization problems for *differentially flat* systems, a general class of dynamical systems to which quadrotors belong. Our analysis of the quadrotor dynamics builds on this previous analysis, by applying it to the problem of matching a user-specified progress curve as closely as possible.

3 Quadrotor Dynamics Model

In this section, we introduce the quadrotor dynamics model we use throughout the paper. Although this model follows previous literature [Mellinger and Kumar 2011; Joubert et al. 2015], we present it here for completeness, and in a unified notation with the rest of the paper. We provide an overview of our model in Figure 2.

Assumptions At a high level, we assume that we must maneuver a rigid body quadrotor equipped with a camera mounted on a gimbal. We assume that the quadrotor and gimbal are *kinematically coupled* [Kondak et al. 2013], in the sense that moving the position of the quadrotor also moves the position of the gimbal. However, we assume that our quadrotor and gimbal are not *dynamically coupled* [Kondak et al. 2013], in the sense that torques acting on the gimbal do not induce reactive torques on the quadrotor. We assume that the gimbal has very large actuator limits.

These assumptions simplify our dynamics model. In particular, our set of assumptions implies that, as long as we can maneuver the quadrotor to the correct position and heading, a 2-axis gimbal can always be oriented to match a desired camera pose. This reasoning allows us to exclude the gimbal configuration and control torques from our dynamics model, resulting in a lower-dimensional and more computationally efficient model than the quadrotor camera model presented by Joubert et al. [2015]. We feel that these assumptions are reasonable, since the cameras mounted on quadrotors tend to be very lightweight relative to the quadrotors themselves. For example, on our quadrotor hardware, the camera and gimbal are roughly $25\times$ lighter than the actual quadrotor.

Degrees of Freedom, Control Forces, and Physical Limits We denote the degrees of freedom in our model with the vector $\mathbf{q}$. This 6-dimensional vector contains the position and orientation of the quadrotor in the world frame. We use Euler angles to represent the orientation of the quadrotor. We refer to $\mathbf{q}$ as the *configuration* of the quadrotor, and we refer to the space of all possible $\mathbf{q}$ values as the quadrotor's *configuration space*. We denote the control forces in our model with the vector $\mathbf{u}$. This 4-dimensional vector contains the magnitude of the upward thrust forces generated at each of the quadrotor's four propellers. We refer to the space of all possible $\mathbf{u}$ values as the quadrotor's *control space*. We assume that our quadrotor can only achieve a limited speed in each dimension, and can only generate limited thrust at each propeller. We express these limits as inequality constraints on $\dot{\mathbf{q}}$ and $\mathbf{u}$. We relate the degrees of freedom in our model to the control forces as follows,

$$\begin{aligned} \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) &= \mathbf{B}(\mathbf{q})\mathbf{u} \\ \text{subject to} \quad \dot{\mathbf{q}}^{\min} \leq \dot{\mathbf{q}} &\leq \dot{\mathbf{q}}^{\max} \\ \mathbf{u}^{\min} \leq \mathbf{u} &\leq \mathbf{u}^{\max} \end{aligned} \tag{1}$$

where the matrix $\mathbf{H}$ models generalized inertia; the matrix $\mathbf{C}$ models generalized velocity-dependent forces like drag; the vector $\mathbf{G}$ models generalized potential forces like gravity; the matrix $\mathbf{B}$ maps from control inputs to generalized forces; and the inequalities represent the velocity limits and control force limits of our system. These matrices can be obtained by expressing the quadrotor dynamics model presented by Mellinger and Kumar [2011] in matrix form [Joubert et al. 2015]. In Appendix A, we include a concise definition for these matrices, which are known as the *manipulator matrices* [Tedrake 2016]. We refer the reader to Joubert et al. [2015] for a more detailed derivation.

4 Solving for Velocities and Control Forces Along a Fixed Trajectory

In this section, we use our model to solve in closed form for the velocities and control forces required to follow a user-specified camera trajectory. As in previous literature [Joubert et al. 2015], we

assume that the user's camera trajectory is C^4 continuous with respect to time.

Our approach in this section roughly follows the exposition by Joubert et al. [2015]. However, we express our approach from a continuous perspective, rather than from a discrete perspective, since we rely on this continuous perspective to develop our analysis and optimization algorithm in Sections 5 and 6.

At a high level, our approach is to solve for a trajectory through configuration space, that places the quadrotor at the same position and heading as the camera at all times. We then substitute this *configuration space trajectory* into equation (1) to solve for the corresponding *control forces*. Although the dynamics generally do not permit us to place the quadrotor at exactly the same orientation as the camera, we can always place it at the same heading. As long as we place the quadrotor at the correct position with the correct heading, a 2-axis gimbal can always be oriented to achieve the desired camera pose.

We provide an algorithm for computing the configuration of a quadrotor along a user-specified camera trajectory in Listing 1. This algorithm implicitly defines a closed form expression for the quadrotor configuration $\mathbf{q}$ in terms of a user-specified camera trajectory. We use the notation $\mathbf{Q}$ to refer to this closed form expression as follows,

$$\mathbf{q} = \mathbf{Q}(\mathbf{c}, \dot{\mathbf{c}}, \ddot{\mathbf{c}}) \tag{2}$$

where $\mathbf{c}$ is a stacked vector that includes the camera position $\mathbf{p}_c$ and the camera look-at vector $\mathbf{x}_c$. The explicit form for $\mathbf{Q}$ can be obtained by proceeding sequentially through Listing 1, symbolically substituting each line into the next. However, the resulting expression for $\mathbf{Q}$ is quite verbose, so we omit it for brevity.

In order to solve for control forces, we need expressions for $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$. We obtain expressions for for $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ by taking the derivative of equation (2) twice with respect to time as follows,

$$\begin{aligned} \dot{\mathbf{q}} &= \frac{d\mathbf{Q}}{dt} &&= \dot{\mathbf{Q}}(\mathbf{c}, \dot{\mathbf{c}}, \ddot{\mathbf{c}}, \dddot{\mathbf{c}}) \\ \ddot{\mathbf{q}} &= \frac{d^2\mathbf{Q}}{dt^2} &&= \ddot{\mathbf{Q}}(\mathbf{c}, \dot{\mathbf{c}}, \ddot{\mathbf{c}}, \dddot{\mathbf{c}}, \ddddot{\mathbf{c}}) \end{aligned} \tag{3}$$

Finally, we solve for the control forces by substituting our expressions for $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ into equation (1) and solving for $\mathbf{u}$ as follows,

$$\mathbf{u} = \mathbf{B}^\star(\mathbf{q}) \left[\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q})\right] \tag{4}$$

where $\mathbf{B}^\star$ is the Moore-Penrose pseudoinverse of $\mathbf{B}$. This approach is guaranteed to yield an exact unique solution for $\mathbf{u}$. This is because we explicitly construct $\mathbf{q}$ to be consistent with the equations of motion for our system, so the left hand side of equation (1) is always in the column space of $\mathbf{B}$, and $\mathbf{B}$ is always full column rank [Joubert et al. 2015]. Since we have closed form expressions for $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$, equation (4) gives us a closed form expression for $\mathbf{u}$. We use the notation $\mathbf{U}$ to refer to this closed form expression as follows,

$$\mathbf{u} = \mathbf{U}(\mathbf{c}, \dot{\mathbf{c}}, \ddot{\mathbf{c}}, \dddot{\mathbf{c}}, \ddddot{\mathbf{c}}) \tag{5}$$

Again, the explicit form for $\mathbf{U}$ is quite verbose, so we omit it for brevity. At this point, we have solved in closed form for the velocities and control forces required to follow a user-specified camera trajectory.

Input:

- The current position, velocity, and acceleration of the camera in the world frame, $\mathbf{p}_c$, $\dot{\mathbf{p}}_c$, $\ddot{\mathbf{p}}_c$.
- The camera's current look-at vector in the world frame, $\mathbf{x}_c$.
- External force in the world frame, $\mathbf{f}_e$.
- Mass of the quadrotor, m.

Output:

- The quadrotor's current configuration, $\mathbf{q}$.

1: $\mathbf{p}_q \leftarrow \mathbf{p}_c$
2: $\mathbf{f} \leftarrow m\ddot{\mathbf{p}}_c$
3: $\mathbf{f}_t \leftarrow \mathbf{f} - \mathbf{f}_e$
4: $\mathbf{y}_q \leftarrow$ normalized $\mathbf{f}_t$
5: $\mathbf{z}_q \leftarrow$ normalized $\mathbf{y}_q \times \mathbf{x}_c$
6: $\mathbf{x}_q \leftarrow$ normalized $\mathbf{z}_q \times \mathbf{y}_q$
7: $\mathbf{R}_{\mathcal{Q},\mathcal{W}} \leftarrow$ the rotation matrix defined by the axes $\mathbf{x}_q, \mathbf{y}_q, \mathbf{z}_q$
8: $\mathbf{e}_q \leftarrow$ the Euler angles corresponding to $\mathbf{R}_{\mathcal{Q},\mathcal{W}}$
9: $\mathbf{q} \leftarrow \begin{bmatrix} \mathbf{p}_q \\ \mathbf{e}_q \end{bmatrix}$

Listing 1: *Computing the configuration of the quadrotor along a user-specified camera trajectory. We begin by setting the quadrotor's position equal to the camera's position (line 1). We substitute linear acceleration and mass into Newton's Second Law to solve for net force (line 2). We decompose the net force acting on our quadrotor into a thrust force and an external force, and we solve for thrust force (line 3). We make the observation that a quadrotor can only generate thrust forces along its local up axis. With this observation in mind, we set the quadrotor's local up axis equal to normalized thrust (line 4). We use the quadrotor's local up axis, and the camera's look-at vector, to determine the quadrotor's orientation (lines 4–8). This approach guarantees that the quadrotor's orientation is always consistent with equation (1).*

5 Solving for Velocities and Control Forces Along a Fixed Path with Variable Timing

In this section, we extend our analysis in Section 4 to the case where only the camera *path* is fixed by the user, but the camera's *timing* can vary. We begin by considering the camera's position and look-at vector as being parameterized by a scalar *path parameter* $s \in [0, 1]$. We emphasize here that s is not time; it is simply a parameter we can sweep from 0 to 1 to trace out our camera path.

In Section 4, we considered the camera position and look-at vector to be explicit functions of time. In this section, we will instead consider these camera parameters to be explicit functions of our path parameter s, and we will consider our path parameter s to be a function of time. We express this *implicit* dependence of our camera parameters on time as follows,

$$\mathbf{c} = \mathbf{c}(s(t)) \tag{6}$$

where t is time. We refer to the function $s(t)$ as the *progress curve* along a path. We assume that our camera parameters are C^4 continuous with respect to our path parameter s.

We factor the time derivatives of $\mathbf{c}$ using the chain rule as follows,

$$\begin{aligned} \dot{\mathbf{c}} &= \mathbf{c}'\dot{s} \\ \ddot{\mathbf{c}} &= \mathbf{c}''\dot{s}^2 + \mathbf{c}'\ddot{s} \\ \dddot{\mathbf{c}} &= \mathbf{c}'''\dot{s}^3 + 3\mathbf{c}''\dot{s}\,\ddot{s} + \mathbf{c}'\dddot{s} \\ \ddddot{\mathbf{c}} &= \mathbf{c}''''\dot{s}^4 + 6\mathbf{c}'''\dot{s}^2\,\ddot{s} + 3\mathbf{c}''\ddot{s}^2 + 4\mathbf{c}''\dot{s}\,\dddot{s} + \mathbf{c}'\ddddot{s} \end{aligned} \tag{7}$$

where we use the notation $\mathbf{c}' = \frac{d\mathbf{c}}{ds}$ to indicate a derivative with respect to our path parameter s, and we use the notation $\dot{s} = \frac{ds}{dt}$ to indicate a derivative with respect to time.

We observe, remarkably, that velocities and control forces are fully determined by the progress curve along a fixed path. To see that this is the case, we substitute equation (7) into our expressions for $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, and $\mathbf{u}$ (i.e., into equations (2), (3), and (5)). We find that we can express the full state of the quadrotor, and all necessary control

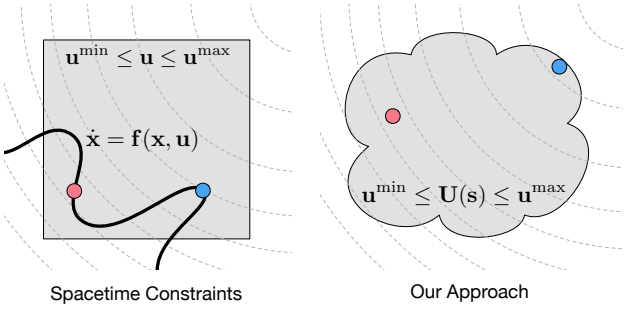


Figure 3: *Illustration of the main difference between spacetime constraints (left) and our approach (right). In spacetime constraints, control force limits are easy to enforce, because they can be encoded as linear inequality constraints (grey shaded region, left). However, quadrotor dynamics are hard to enforce, because they must be encoded as highly non-linear equality constraints (bold curve, left). These non-linear equality constraints force a solver to take many small steps to get from an initial guess (red dot, left) to the optimal solution (blue dot, left). In our approach, control force limits are more challenging to enforce, because they must be encoded as non-linear inequality constraints (grey shaded region, right). However, our approach enforces the quadrotor dynamics implicitly, without requiring additional equality constraints. Our approach enables a solver to make very rapid progress from an initial guess (red dot, right) to the optimal solution (blue dot, right).*

forces, in terms of the functions $\dot{s}(t)$, $\ddot{s}(t)$, $\dddot{s}(t)$, $\ddddot{s}(t)$, which can vary, and the functions $\mathbf{c}(s)$, $\mathbf{c}'(s)$, $\mathbf{c}''(s)$, $\mathbf{c}'''(s)$, $\mathbf{c}''''(s)$, which are fully determined by the fixed path.

We also observe that a progress curve must be C^4 continuous with respect to time in order for it to satisfy the quadrotor dynamics in equation (1). This is because our expression for $\mathbf{u}$ depends on $\ddddot{\mathbf{c}}$, and $\ddddot{\mathbf{c}}$ depends on $\ddddot{s}$.

Together, these observations motivate the design of our optimization algorithm, described in Section 6.

6 Progress Curve Optimization

In Section 5, we observed that the velocities and control forces required to fly a quadrotor along a fixed path, are fully determined by a progress curve along the path. In this section, we optimize a progress curve to match a user-specified input progress curve as closely as possible, subject to velocity and control force limits. We illustrate the main difference between spacetime constraints and our approach in Figure 3.

At a high level, we assume that we are given as input a user-specified camera path and progress curve. We discretize the camera path into a sequence of sample points, and we treat these sample points as fixed. At each sample point, we solve for the progress curve *time derivatives* that best agree with the user's input progress curve. During this optimization procedure, we explicitly enforce C^4 continuity of our progress curve, as well as velocity and control force limits. In a simple post-processing step, we recover our output progress curve from the time derivatives. After this post-processing step, our output progress curve can be safely flown on a real quadrotor.

Enforcing C^4 Continuity In our discrete problem, we must take care to enforce C^4 continuity of our progress curve with respect to time. Stating this continuity constraint formally, let $\mathbf{s}_i$ be the first 4 time derivatives of our progress curve at sample point i. Let v_i be the 5th time derivative of our progress curve at sample point i. Let dt_i be the time delta between sample points i and $i+1$. We enforce C^4 continuity of our progress curve as follows,

$$\mathbf{s}_{i+1} = \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}v_i)dt_i$$

$$\begin{aligned} &\text{subject to} \quad v^{\min} \leq v_i \leq v^{\max} \\ &\text{where} \quad \mathbf{M} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{N} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned} \tag{8}$$

Mathematically speaking, this constraint does correctly enforce C^4 continuity. However, since we intend to *indirectly* optimize our progress curve by optimizing its time derivatives, we do not expect to have explicit access at optimization time to dt_i. Therefore, we substitute $dt_i = \frac{ds_i}{\dot{s}_i}$ into equation (8) to arrive at the following equality constraint,

$$\mathbf{s}_{i+1} = \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}v_i)\frac{ds_i}{\dot{s}_i} \tag{9}$$

In equation (8), $v^{\min}$ and $v^{\max}$ control the extent to which $\dddot{s}$ is allowed to vary from one sample point to the next, while still considering our progress curve to be C^4 continuous. In our implementation, we set $v^{\min}$ and $v^{\max}$ heuristically, based on the minimum and maximum derivatives we observe in the input progress curve. See Appendix B for details.

Enforcing Forward Progress In our discrete problem, we must take care to ensure that the quadrotor always makes forward progress along the path. Or stated more precisely, we must enforce the constraint that $\dot{s} > 0$. Again, since we indirectly optimize our progress curve by optimizing its time derivatives, this constraint ensures that our optimized progress curve always reaches the end of the path.

Full Optimization Problem Stating our optimization problem formally, let $\mathbf{S}$ be the concatenated vector of all $\mathbf{s}_i$ values along the path. Similarly, let $\mathbf{V}$ be the concatenated vector of all v_i values along the path. Let $\dot{s}_i^{\text{ref}}$ be the 1st time derivative of the user's input progress curve at sample point i. We would like to find the the optimal set of progress curve time derivatives $\mathbf{S}^*$ and $\mathbf{V}^*$ as follows,

$$\begin{aligned} \mathbf{S}^*, \mathbf{V}^* &= \arg\min_{\mathbf{S},\mathbf{V}} \sum_i (\dot{s}_i - \dot{s}_i^{\text{ref}})^2 \\ \text{subject to} \quad \mathbf{s}_{i+1} &= \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}v_i)\frac{ds_i}{\dot{s}_i} \\ v^{\min} \leq v_i \leq v^{\max} &\qquad \dot{\mathbf{q}}^{\min} \leq \dot{\mathbf{Q}}(\mathbf{s}_i) \leq \dot{\mathbf{q}}^{\max} \\ \dot{s}_i > 0 &\qquad \mathbf{u}^{\min} \leq \mathbf{U}(\mathbf{s}_i) \leq \mathbf{u}^{\max} \end{aligned} \tag{10}$$

The objective function in this optimization problem attempts to match the optimized progress curve with the input progress curve as closely as possible. The equality constraints, and the inequality constraints on v_i, enforce C^4 continuity of the progress curve. The inequality constraint on $\dot{s}_i$ ensures that the progress curve always reaches the end of the path. The other inequality constraints enforce velocity and control force limits. We use the notation $\dot{\mathbf{Q}}(\mathbf{s}_i)$ and $\mathbf{U}(\mathbf{s}_i)$ to refer to our closed form expressions for velocities and control forces, expressed entirely in terms of our progress curve time derivatives, as described in Section 5. $\mathbf{S}$ and $\mathbf{V}$ are decision variables, everything else is problem data.

Figure 4: *Side-by-side comparison of a* GOOGLE EARTH *shot preview (top row) and real video footage (bottom row) from an aggressive trajectory generated using our algorithm. Our algorithm produces trajectories that can be faithfully captured on a real quadrotor, even when the trajectories are at the quadrotor's physical limits.*

Improving Computational Efficiency To improve computational efficiency, we make two important approximations to the problem in (10).

First, in order for our non-convex solver to achieve acceptable performance, we must compute analytic derivatives of our objective function, and our constraint functions, with respect to our decision variables. We use the symbolic algebra library SymPy to compute these analytic derivatives [SymPy 2014]. However, we found that computing an analytic expression for $\frac{\partial \mathbf{U}}{\partial \mathbf{s}_i}$ was not possible in a reasonable amount of time due to the factorial (in the number of mathematical symbols) complexity of the pseudoinverse term in equation (4). Therefore, we use the following approximation for equation (4),

$$\bar{\mathbf{u}} = \bar{\mathbf{B}}^{\star}\left[\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q})\right] \tag{11}$$

where $\bar{\mathbf{B}}^{\star}$ is a constant (for the purpose of computing analytic derivatives) approximation of $\mathbf{B}^{\star}(\mathbf{q})$.

Second, we would prefer for our equality constraints to be linear, since this would make more of our problem convex, and therefore more computationally efficient [Boyd 2008]. With this reasoning in mind, we replace our *nearly* linear equality constraints with the following linear approximations,

$$\mathbf{s}_{i+1} = \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}\mathbf{v}_i)\frac{ds_i}{\bar{\dot{s}}_i} \tag{12}$$

where $\bar{\dot{s}}_i$ is a constant (for the purpose of computing analytic derivatives) approximation of $\dot{s}_i$.

Although we treat $\bar{\mathbf{B}}^{\star}$ and $\bar{\dot{s}}$ as constant for the purpose of computing analytic derivatives, we iteratively refine the values of $\bar{\mathbf{B}}^{\star}$ and $\bar{\dot{s}}$ as our solver makes progress towards a solution, according to the following update rules,

$$\bar{\mathbf{B}}^{\star[i]} = \mathbf{B}^{\star}(\mathbf{q}^{[i-1]}) \qquad \bar{\dot{s}}^{[i]} = \dot{s}^{[i-1]} \tag{13}$$

where the superscript notation refers to our solver's iteration count. Boyd refers to this strategy as *quasi-linearization* [2008].

To further improve the convergence behavior of our algorithm, at the expense of a modest reduction in accuracy, we simply stop updating $\bar{\dot{s}}$ after a small fixed number of iterations. In all the results shown in this paper, we stop updating $\bar{\dot{s}}$ after 10 iterations. We evaluate the speed and accuracy tradeoffs associated with this choice in Section 7.

Fast Approximate Optimization Problem Based on the approximations described in the previous subsection, we express our fast approximate optimization problem as follows,

$$\begin{aligned} \mathbf{S}^*, \mathbf{V}^* &= \arg\min_{\mathbf{S},\mathbf{V}} \sum_i (\dot{s}_i - \dot{s}_i^{\text{ref}})^2 \\ \text{subject to} \quad & \mathbf{s}_{i+1} = \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}v_i)\frac{ds_i}{\bar{\dot{s}}_i} \\ & v^{\min} \le v_i \le v^{\max} \qquad \dot{\mathbf{q}}^{\min} \le \dot{\mathbf{Q}}(\mathbf{s}_i) \le \dot{\mathbf{q}}^{\max} \\ & \dot{s}_i > 0 \qquad\qquad\quad \mathbf{u}^{\min} \le \bar{\mathbf{U}}(\mathbf{s}_i) \le \mathbf{u}^{\max} \end{aligned} \tag{14}$$

where $\bar{\mathbf{U}}(\mathbf{s}_i)$ is our closed form expression for control forces that includes the approximation from equation (11). The problem in (14) is expressed in a standard form that can be given directly to an off-the-shelf solver. In our implementation, we solve the problem in (14) using the commercially available non-convex solver SNOPT [Gill et al. 2002].

Initialization The optimization problem in (14) is non-convex, and is therefore sensitive to initialization. We initialize our solver by uniformly time stretching the input progress curve until it becomes feasible. Next, we compute the time derivatives of the uniformly stretched progress curve, and we use these time derivatives to initialize our solver. When computing these time derivatives, we take care to compute numerical derivatives using the same forward differencing scheme as in equation (8). In our experience, this initialization strategy noticeably improves the convergence behavior of our solver, compared to initializing with the original infeasible input progress curve. We speculate that this improvement occurs because our constraint gradients are relatively well-behaved near hover conditions, but become increasingly oscillatory and non-convex far away from hover conditions. Therefore, initializing with an overly conservative feasible rajectory, rather than an overly aggressive infeasible trajectory, allows our solver to take advantage of more globally meaningful gradient information.

7 Evaluation and Discussion

In this section, we qualitatively evaluate our algorithm, we describe our dataset of infeasible quadrotor camera trajectories and the experiments we conducted to quantitatively evaluate our algorithm, and we discuss our key results.

In the experiments in this section, we evaluate our algorithm, a modified version of our algorithm where we update $\bar{\dot{s}}$ for 50 iterations (instead of our usual 10 iterations), and a spacetime constraints approach. We describe the exact spacetime constraints problem formulation we use in our experiments in Appendix C. We

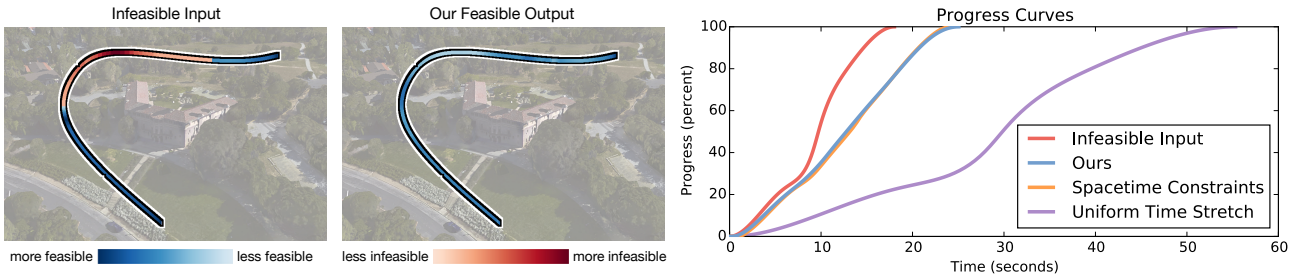


Figure 5: *An aggressive infeasible trajectory (far left), the feasible output trajectory produced by our algorithm (near left), and the feasible progress curves produced by our algorithm and spacetime constraints for this trajectory (right). As a baseline, we include the progress curve obtained by uniformly time stretching the input trajectory until it becomes feasible. Our algorithm and spacetime constraints produce similar progress curves, and both methods perturb the timing of the input trajectory much less than uniform time stretching.*

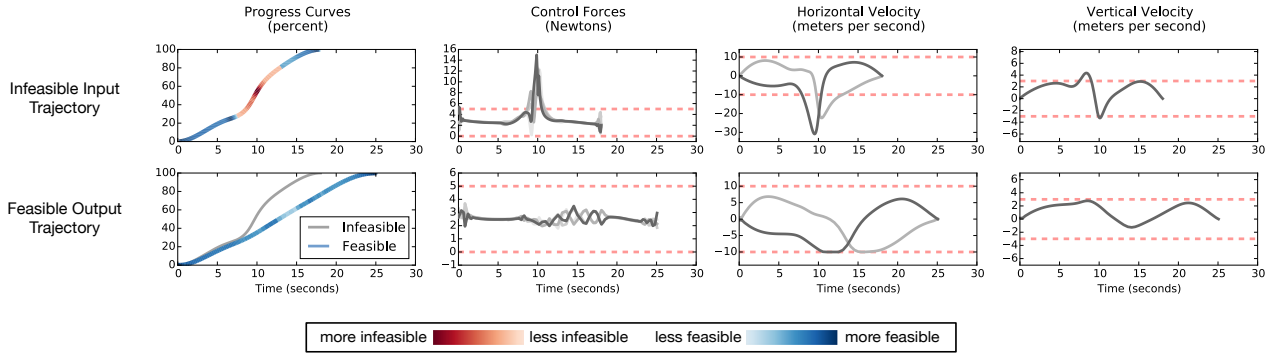


Figure 6: *Our algorithm perturbs the timing of the infeasible input trajectory shown in Figure 5 as little as possible, while remaining within our quadrotor's physical limits. To demonstrate this behavior, we plot the infeasible input (top) and feasible output (bottom) progress curves, control force curves, and velocity curves. For reference, we plot the infeasible input progress curve in grey underneath the feasible output progress curve. We indicate control force and velocity limits with horizontal dotted lines. The feasible trajectory produced by our algorithm is at our quadrotor's physical limits for sustained periods, but never exceeds these physical limits.*

discretize each trajectory in our experiments at a moderate resolution of 100 time samples, unless otherwise noted.

Comparing Shot Previews and Real Video Footage In Figure 4, we show a side-by-side comparison of a GOOGLE EARTH shot preview and real video footage from an aggressive trajectory generated using our algorithm. To capture this video footage, we use the quadrotor hardware platform described by Joubert et al. [2015].

Qualitative Comparison with Spacetime Constraints In Figure 5, we show progress curves generated by our algorithm and spacetime constraints for an aggressive infeasible trajectory. We observe that the progress curves produced by both algorithms are very similar. We note that our algorithm and spacetime constraints are solving slightly different problems, so we do not expect the solutions to be identical. In Figure 6, we show that our algorithm remains within the prescribed velocity and control force limits, when generating the output trajectory shown in Figure 5.

Infeasible Trajectory Dataset We based our dataset of infeasible quadrotor camera trajectories on raw data from the user study previously conducted by Joubert et al. [2015]. In this study, participants were tasked with creating cinematically interesting shots using HORUS, an open source tool for quadrotor camera shot planning. This study included 2 expert cinematographers, and 2 novice cinematographers with computer graphics experience.

HORUS notifies users when their shots are infeasible, but requires users to manually edit their shots to make them feasible. All of the shots from the previous study were infeasible at some point during the user's editing session.

HORUS saves the complete revision history of a shot as it is being edited. To compile our dataset, we simply found an infeasible revision for each shot in this previous study. For the purpose of data collection, we assumed that revisions closer to the end of an editing session were more faithful to the user's artistic intent. Therefore, as we compiled our dataset, we preferred revisions that were closer to end of an editing session. From this data collection procedure, we obtained 8 infeasible shots. The feasibility violations in these shots ranged from moderate (e.g., briefly violating velocity limits by less than 10%) to pronounced (e.g., violating velocity or control force limits by more than 50% for sustained periods). We show the infeasible trajectories in our dataset in Figure 7.

Computational Performance To evaluate computational performance, we compared the running times of our algorithm and spacetime constraints on our dataset of infeasible trajectories. We show the results from these experiments in Figure 8.

In the interest of making fair comparisons, we took care to structure the implementations of our algorithm and spacetime constraints as similarly as possible. The following implementation details apply to both implementations. We solve the non-convex optimization problems arising in both approaches using the commercially avail-

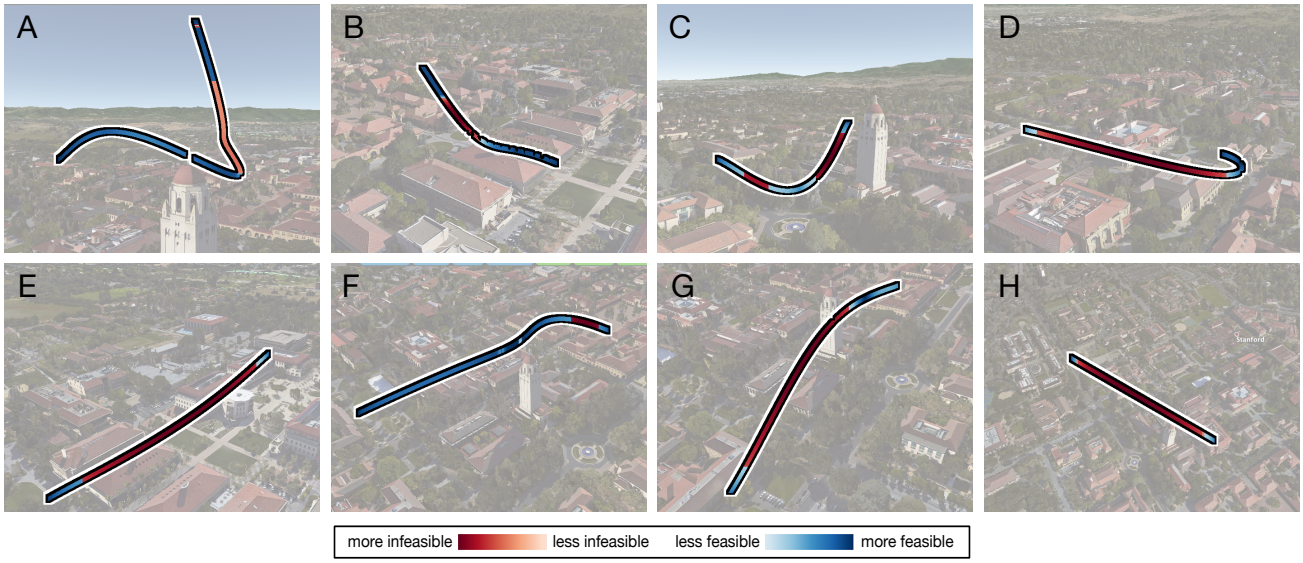


Figure 7: *Our dataset of infeasible quadrotor camera trajectories. We show the trajectories in their spatial context, and color them according to how severely they violate our quadrotor's physical limits. We use the letters in this figure to refer to individual trajectories throughout the paper.*

able solver SNOPT [Gill et al. 2002]. We initialize both approaches using the initialization strategy described in Section 6. We call SNOPT using a Python wrapper provided by the authors, and we use all the default SNOPT error tolerances and parameters in all our experiments. We express our objective function and constraint functions symbolically using the open-source symbolic algebra library SymPy [SymPy 2014]. We use SymPy to automatically generate all necessary gradient expressions. Finally, we use SymPy to generate efficient C code, which we call from Python, to evaluate the objective function, all constraint functions, and all necessary gradients. We perform all our experiments on a Late 2013 Macbook Pro with a 2.6 GHz Intel Core i7 processor and 16GB of memory.

Convergence Behavior To evaluate the convergence behavior of our algorithm, we examined how the objective value in our optimization problem decreases as SNOPT makes progress towards an optimal solution for a trajectory in our dataset. Similarly, we examined how the equality constraint functions, which are zero when the equality constraints are satisfied exactly, decrease as SNOPT makes progress. Again, in all our experiments, we call SNOPT using all the default error tolerances and parameters. We show the results from these experiments in Figure 9.

Accuracy To evaluate accuracy, we conducted an experiment using the state space and control trajectories produced by our algorithm. We simulated the control trajectories using a 5^{th} order accurate rigid body physics simulator, and we measured how well the results of the simulation matched the state space trajectories produced by our algorithm. We repeated this experiment with spacetime constraints, the modified version of our algorithm, and the original infeasible input trajectory simulated without control force limits. We show results from these experiments in Figure 10. Due to the relatively large time steps involved in these simulations, we applied LQR feedback control [Tedrake 2016] in order to prevent the simulations from diverging. We used identical LQR parameters in all our experiments. We do not allow the LQR feedback controller to exceed the quadrotor's control force limits, except when simulating the infeasible input trajectories.

Dimensionality At each time sample, spacetime constraints requires at least 6 scalar decision variables for the configuration of the quadrotor, and 4 scalar decision variables for the quadrotor control forces. In contrast, our formulation requires 5 scalar decision variables at each time sample. Both formulations lead to the same block bi-diagonal structure in the constraint Jacobian. Therefore, compared to spacetime constraints, our formulation reduces the required number of decision variables by at least 50%, while preserving the efficient sparsity pattern in the constraint Jacobian.

Limitations By design, our algorithm will make an input trajectory feasible by perturbing its timing, but will not modify its spatial layout. Therefore, our algorithm is not directly applicable in scenarios where the precise timing of the trajectory must be maintained. In these scenarios, a spacetime constraints approach would be more appropriate. That being said, we believe there is a broad class of usage scenarios in cinematography, journalism, and architecture, where re-timing an infeasible trajectory is reasonable behavior. Therefore, we do not believe this limitation is overly burdensome.

8 Conclusions

We analyzed the dynamics of a quadrotor along a fixed path, and we found that the quadrotor's velocities and control forces are fully determined by its progress curve along the path. This insight lead us to a fast and user-friendly algorithm for generating feasible quadrotor camera trajectories. We implemented our algorithm in an open source tool for designing quadrotor camera shots, and we ran performance benchmarks on a dataset of infeasible quadrotor camera trajectories. We found that our approach is between $25\times$ and $180\times$ faster than spacetime constraints. We successfully captured real video footage using the trajectories generated by our algorithm.

In the future, we believe the ideas in this paper could become part of the standard toolbox for quadrotor trajectory planning. By moving along networks of fixed paths, quadrotor cameras could film highly dynamic scenes with stronger safety guarantees. By considering other objective functions along fixed paths, quadrotors could perform a wide range of tasks more efficiently and more safely.

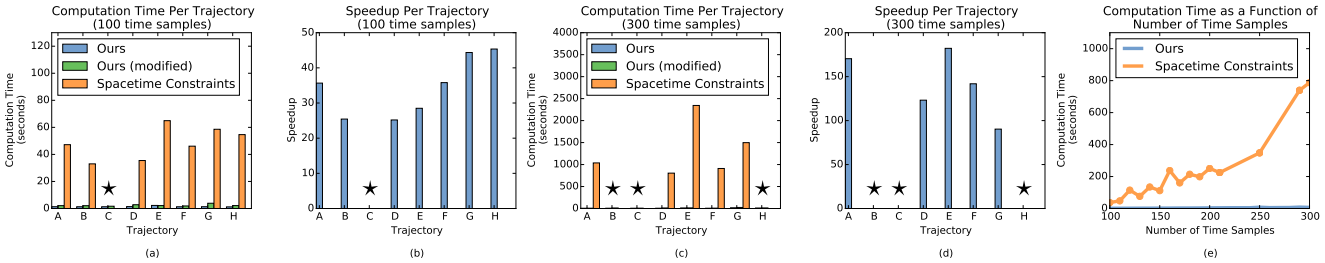


Figure 8: *Computational performance and scaling behavior of our algorithm on our dataset of infeasible trajectories. When we solve for trajectories discretized at a moderate resolution of 100 time samples, our algorithm runs in less than 2 seconds, and is between 25× and 45× faster than spacetime constraints (a and b). As we scale to more finely discretized trajectories, this performance gap widens, with our algorithm outperforming spacetime constraints by between 90× and 180× (c and d). We indicate trajectories where spacetime constraints failed to find a solution with a ⋆. We show the scaling behavior of our algorithm and spacetime constraints for trajectory* D*, which is the trajectory where spacetime constraints performed the best (e). When scaling beyond 200 time samples, spacetime constraints did not consistently find a feasible solution for trajectory* D*. We indicate where spacetime constraints successfully found a solution for trajectory* D *with colored dots. As a baseline, we include timing results for the modification of our algorithm described in Section 7. On plots a, c, and e, lower is better.*

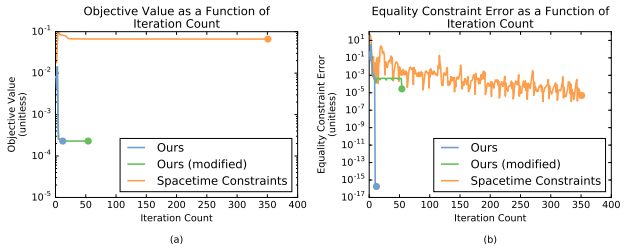


Figure 9: *Convergence behavior of our algorithm on trajectory* B *from our dataset of infeasible trajectories, discretized into 100 time samples. This is the trajectory where spacetime constraints converged the fastest. Our algorithm, the modification of our algorithm described in Section 7, and spacetime constraints all converge rapidly to an optimal objective value (a). However, because the equality constraints in our formulation are nearly linear, our algorithm converges to a solution that satisfies the equality constraints much faster than spacetime constraints (b). We measure equality constraint error by summing the L2 norm of each vector-valued equality constraint function, which is zero when the equality constraint is satisfied exactly. The y axes on these plots use a log scale. Lower is better.*

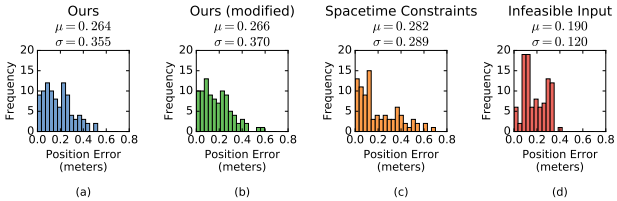


Figure 10: *Accuracy of our algorithm in predicting the results of a rigid body physics simulation. We simulate the control trajectory produced by our algorithm using a 5^{th} order accurate rigid body simulator, and we measure how well the results of the simulation match the state space trajectory produced by our algorithm (a). We repeat this experiment using the modification of our algorithm described in Section 7 (b), spacetime constraints (c), and the original infeasible input trajectory simulated without control limits (d). We plot the results for trajectory* G *in our dataset of infeasible trajectories, which was the trajectory where our algorithm performed the worst. We provide mean error (μ) and standard deviation (σ) values above each plot. Even for this worst-case trajectory, our algorithm is more accurate than spacetime constraints, with slightly lower mean error. For this particular trajectory, our unmodified algorithm is also slightly more accurate than our modified algorithm. Having more histogram mass further to the left is better.*

Acknowledgements

We thank the anonymous reviewers for their valuable feedback; Niels Joubert for his assistance in collecting video footage; Anh Truong for her support in modifying the HORUS user interface; Mackenzie Leake for her assistance in preparing the supplementary video; Jane E for narrating the supplementary video; Ross Daly for his assistance in preparing figures; Enzo Busseti for his assistance with an early version of the optimization problem in this paper; Stephen Boyd and Ron Fedkiw for their helpful and lively discussions; and 3D ROBOTICS for their generous hardware support. This work was supported in part by a NSERC Alexander Graham Bell Canada Graduate Scholarship. Finally, we dedicate this paper in loving memory of our dear friend and valued collaborator, Floraine Berthouzoz.

References

3D ROBOTICS, 2015. Solo. http://3drobotics.com/solo.

APM, 2015. APM Autopilot Suite. http://ardupilot.com.

BETTS, J. T. 1998. A survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics 21*, 2.

BOUKTIR, Y., HADDAD, M., AND CHETTIBI, T. 2008. A prototype of an autonomous controller for a quadrotor UAV. In *Mediterranean Conference on Control and Automation 2008*.

BOYD, S., 2008. Convex optimization II (course notes for Stanford EE364b). http://stanford.edu/class/ee364b.

BRY, A., RICHTER, C., BACHRACH, A., AND ROY, N. 2015. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *International Journal of Robotics Research 34*, 7.

COWLING, I. D., YAKIMENKO, O. A., WHIDBORNE, J. F., AND COOKE, A. K. 2007. A prototype of an autonomous controller

for a quadrotor UAV. In *European Conference on Control (ECC) 2007*.

DAHL, O., AND NIELSEN, L. 1990. Torque-limited path following by on-line trajectory time scaling. *Transactions on Robotics and Automation 6*, 5.

DEITS, R., AND TEDRAKE, R. 2015. Efficient mixed-integer planning for UAVs in cluttered environments. In *International Conference on Robotics and Automation (ICRA) 2015*.

DJI, 2015. DJI Go. http://www.dji.com/product/goapp.

DJI, 2015. DJI Ground Station. http://www.dji.com/product/pc-ground-station.

FANG, A. C., AND POLLARD, N. S. 2003. Efficient synthesis of physically valid human motion. *Transactions on Graphics (Proc. SIGGRAPH 2003) 22*, 3.

FAULWASSER, T., HAGENMEYER, V., AND FINDEISENC, R. 2014. Constrained reachability and trajectory generation for flat systems. *Automatica 50*, 4.

GEBHARDT, C., HEPP, B., NAGELI, T., STEVSIC, S., AND HILLIGES, O. 2016. Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals. In *CHI 2016*.

GEIJTENBEEK, T., AND PRONOST, N. 2012. Interactive character animation using simulated physics: A state-of-the-art review. *Computer Graphics Forum 31*, 8.

GILL, P. E., MURRAY, W., AND SAUNDERS, M. A. 2002. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization 12*, 4.

JOUBERT, N., ROBERTS, M., TRUONG, A., BERTHOUZOZ, F., AND HANRAHAN, P. 2015. An interactive tool for designing quadrotor camera shots. *Transactions on Graphics (Proc. SIGGRAPH Asia 2015) 34*, 6.

KONDAK, K., KRIEGER, K., ALBU-SCHAEFFER, A., SCHWARZBACH, M., LAIACKER, M., MAZA, I., RODRIGUEZ-CASTANO, A., AND OLLERO, A. 2013. Closed-loop behavior of an autonomous helicopter equipped with a robotic arm for aerial manipulation tasks. *International Journal of Advanced Robotic Systems 10*, 145.

LIPP, T., AND BOYD, S. 2014. Minimum-time speed optimisation over a fixed path. *International Journal of Control 87*, 6.

MCCANN, J., POLLARD, N. S., AND SRINIVASA, S. 2006. Physics-based motion retiming. In *SCA 2006*.

MEIER, L., TANSKANEN, P., HENG, L., LEE, G. H., FRAUNDORFER, F., AND POLLEFEYS, M. 2012. PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots 33*, 1–2.

MELLINGER, D., AND KUMAR, V. 2011. Minimum snap trajectory generation and control for quadrotors. In *International Conference on Robotics and Automation (ICRA) 2011*.

SAFONOVA, A., HODGINS, J. K., AND POLLARD, N. S. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *Transactions on Graphics (Proc. SIGGRAPH 2004) 23*, 3.

SHIN, K. G., AND MCKAY, N. D. 1985. Minimum-time control of robotic manipulators with geometric path constraints. *Transactions on Automatic Control 30*, 6.

SLOTINE, J.-J. E., AND YANG, H. S. 1989. Improving the efficiency of time-optimal path-following algorithms. *Transactions on Robotics and Automation 5*, 1.

SYMPY, 2014. SymPy: Python library for symbolic mathematics. http://www.sympy.org.

TEDRAKE, R., 2016. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for MIT 6.832). http://underactuated.mit.edu.

VAN LOOCK, W., PIPELEERS, G., AND SWEVERS, J. 2013. Time-optimal quadrotor flight. In *European Control Conference (ECC) 2013*.

VERSCHEURE, D., DEMEULENAERE, B., SWEVERS, J., SCHUTTER, J. D., AND DIEHL, M. 2009. Time-optimal path tracking for robots: A convex optimization approach. *Transactions on Automatic Control 54*, 10.

WITKINS, A., AND KASS, M. 1988. Spacetime constraints. In *SIGGRAPH 1988*.

A Quadrotor Manipulator Matrices

In this appendix, we define the quadrotor manipulator matrices, attempting to be as concise as possible. We refer the reader to Joubert et al. [2015] for a more detailed derivation.

We begin by defining the layout of our configuration vector $\mathbf{q}$ as follows,

$$\mathbf{q} = \begin{bmatrix} \mathbf{p}_q \\ \mathbf{e}_q \end{bmatrix} \tag{15}$$

where $\mathbf{p}_q$ is the position of the quadrotor; and $\mathbf{e}_q$ is the vector of Euler angles representing the quadrotor's orientation in the world frame.

We express the manipulator matrices for our quadrotor system as follows,

$$\begin{aligned} \mathbf{H}(\mathbf{q}) &= \begin{bmatrix} m\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}} \end{bmatrix} \\ \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) &= \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\dot{\mathbf{A}} - \left(\mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}\dot{\mathbf{e}}\right)_\times \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A} \end{bmatrix} \\ \mathbf{G}(\mathbf{q}) &= \begin{bmatrix} -\mathbf{f}_e \\ \mathbf{0}_{3\times1} \end{bmatrix} \\ \mathbf{B}(\mathbf{q}) &= \begin{bmatrix} \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f} \\ \mathbf{M}_\tau \end{bmatrix} \end{aligned} \tag{16}$$

where m is the mass of the quadrotor; $\mathbf{I}_q$ is the inertia matrix of the quadrotor; $\mathbf{R}_{\mathcal{W},\mathcal{Q}}$ is the rotation matrix that represents the quadrotor's orientation in the world frame (i.e., the rotation matrix that maps vectors from the body frame of the quadrotor into the world frame); $\mathbf{R}_{\mathcal{Q},\mathcal{W}}$ is the rotation matrix that maps vectors from the world frame into the body frame of the quadrotor; $\mathbf{A}$ is the matrix that relates the quadrotor's Euler angle time derivatives to its angular velocity in the world frame; $\mathbf{f}_e$ is the external force; $\mathbf{M}_\mathbf{f}$ is the matrix that maps the control input at each of the quadrotor's propellers into a net thrust force oriented along the quadrotor's local $\mathbf{y}$ axis; $\mathbf{M}_\tau$ is the matrix that maps the control input at each of the quadrotor's propellers into a net torque acting on the quadrotor in the body frame; $\mathbf{0}_{p\times q}$ is the $p \times q$ zero matrix; $\mathbf{I}_{k\times k}$ is the $k \times k$ identity matrix; and the notation $(\mathbf{a})_\times$ refers to the skew-symmetric matrix, computed as a function of the vector $\mathbf{a}$, such that $(\mathbf{a})_\times \mathbf{b} = \mathbf{a} \times \mathbf{b}$ for all vectors $\mathbf{b}$.

Our expressions for the manipulator matrices depend on the matri-

ces, $\mathbf{M_f}$ and $\mathbf{M}_\tau$. We define these matrices as follows,

$$\mathbf{M_f} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{M}_\tau = \begin{bmatrix} ds_\alpha & ds_\beta & -ds_\beta & -ds_\alpha \\ \gamma & -\gamma & \gamma & -\gamma \\ -dc_\alpha & dc_\beta & dc_\beta & -dc_\alpha \end{bmatrix} \tag{17}$$

where d, α, β, and γ are constants related to the physical design of a quadrotor: d is the distance from the quadrotor's center of mass to its propellers; α is the angle in radians that the quadrotor's front propellers form with the quadrotor's positive $\mathbf{x}$ axis; β is the angle in radians that the quadrotor's rear propellers form with the quadrotor's negative $\mathbf{x}$ axis; γ is the magnitude of the in-plane torque generated by the quadrotor propeller producing 1 unit of upward thrust force; $c_a = \cos a$ and $s_a = \sin a$.

Note that our expressions for the quadrotor manipulator matrices, in particular our expressions for $\mathbf{A}$ and $\dot{\mathbf{A}}$, depend on our choice of Euler angle conventions. We follow the Euler angle conventions described by Joubert et al. [2015]. See the detailed derivation by Joubert et al. [2015] for details.

B Setting $v^{\min}$ and $v^{\max}$

In our implementation, we set $v^{\min}$ and $v^{\max}$ heuristically, based on the minimum and maximum derivatives we observe in the input progress curve. In particular, we set $v^{\min}$ and $v^{\max}$ as follows,

$$\begin{aligned} v^{\min} &= v^{\min}_{\text{ref}} - \lambda_{\text{proportional}}(v^{\max}_{\text{ref}} - v^{\min}_{\text{ref}}) - \lambda_{\text{fixed}} \\ v^{\max} &= v^{\max}_{\text{ref}} + \lambda_{\text{proportional}}(v^{\max}_{\text{ref}} - v^{\min}_{\text{ref}}) + \lambda_{\text{fixed}} \end{aligned} \tag{18}$$

where $v^{\min}_{\text{ref}}$ and $v^{\max}_{\text{ref}}$ are the minimum and maximum 5$^{\text{th}}$ time derivatives of the input progress curve; $\lambda_{\text{proportional}}$ has the effect of padding $v^{\min}$ and $v^{\max}$ proportionally to the range of derivatives observed in input progress curve; and λ_{fixed} pads $v^{\min}$ and $v^{\max}$ by a fixed amount. In our implementation, we set $\lambda_{\text{proportional}} = 0.3$ and $\lambda_{\text{fixed}} = 0.001$. We found that including both proportional and fixed padding terms when setting $v^{\min}$ and $v^{\max}$ improved the overall convergence behavior of our algorithm. This heuristic assumes that the input progress curve is C^4 continuous. We make this assumption for simplicity, although it could be relaxed by making minor modifications to equation (18) above.

C Spacetime Constraints Formulation

We begin by concatenating our configuration vector $\mathbf{q}$ and generalized velocity vector $\dot{\mathbf{q}}$ into a single state vector $\mathbf{x}$ as follows,

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{bmatrix} \tag{19}$$

We formulate the spacetime constraints optimization problem used in our experiments as follows,

$$\mathbf{X}^*, \mathbf{U}^*, \mathbf{T}^* = \underset{\mathbf{X},\mathbf{U},\mathbf{T}}{\arg\min} \sum_i \left[\lambda_{\mathbf{p}} \left\| \mathbf{p}_i - \mathbf{p}_i^{\text{ref}} \right\|_2^2 + \lambda_{dt} \left(dt_i - dt_i^{\text{ref}} \right)^2 \right]$$

$$\begin{aligned} \text{subject to} \quad \mathbf{x}_0 &= \mathbf{x}_0^{\text{ref}} \\ \mathbf{x}_N &= \mathbf{x}_N^{\text{ref}} \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) dt_i \\ \mathbf{x}^{\min} &\le \mathbf{x}_i \le \mathbf{x}^{\max} \\ \mathbf{u}^{\min} &\le \mathbf{u}_i \le \mathbf{u}^{\max} \\ dt^{\min} &\le dt_i \le dt^{\max} \end{aligned} \tag{20}$$

where $\mathbf{X}$ is the concatenated vector of quadrotor states across all time samples; $\mathbf{U}$ is the concatenated vector of control forces across all time samples; $\mathbf{T}$ is the concatenated vector of all time deltas; $\lambda_{\mathbf{p}}$ and λ_{dt} are parameters that trade off the optimizer's preference for matching the spatial layout of the user's input trajectory versus matching the timing of the user's input trajectory; $\mathbf{p}_i$ is the quadrotor position at time sample i; $\mathbf{p}_i^{\text{ref}}$ is the reference position at time sample i obtained from the user's input trajectory; dt_i is the time delta from time sample i to time sample $i+1$; dt_i^{ref} is the reference time delta from time sample i to time sample $i+1$ obtained from the user's input trajectory; $\mathbf{x}_i$ is the quadrotor state at time sample i (note that the state variable $\mathbf{x}_i$ includes the quadrotor's position $\mathbf{p}_i$); $\mathbf{u}_i$ is the control force vector at time sample i; $\mathbf{f}$ is a function that encodes the quadrotor dynamics; $\mathbf{x}_0^{\text{ref}}$ and $\mathbf{x}_N^{\text{ref}}$ are the reference start and end states of the quadrotor obtained from the user's input trajectory; and $\mathbf{x}^{\min}$, $\mathbf{x}^{\max}$, $\mathbf{u}^{\min}$, $\mathbf{u}^{\max}$, $dt^{\min}$, and $dt^{\max}$ are state space limits, control force limits, and time stretching limits imposed on the trajectory. $\mathbf{X}$, $\mathbf{U}$, and $\mathbf{T}$ are decision variables, everything else is problem data.

Our spacetime constraints formulation depends on the function $\mathbf{f}$, which encodes the quadrotor dynamics. We define this function in terms of the manipulator matrices from Appendix A as follows,

$$\mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) = \begin{bmatrix} \dot{\mathbf{q}}_i \\ \ddot{\mathbf{q}}_i \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{q}}_i \\ \mathbf{H}_i^{-1}(\mathbf{B}_i \mathbf{u}_i - \mathbf{C}_i \dot{\mathbf{q}}_i - \mathbf{G}_i) \end{bmatrix} \tag{21}$$

In all our experiments, we set $\lambda_{\mathbf{p}} = 0.01$ and $\lambda_{dt} = 0.0001$. We found that these parameter values yielded the best possible computational performance, while still producing trajectories that closely matched the spatial layout the user's input trajectory.

This spacetime constraints formulation departs from the original formulation by Witkins and Kass [1988], in the sense that the optimizer is free to stretch time. This freedom is required in order to ensure that a feasible solution exists.