An Interactive Tool for Designing Quadrotor Camera Shots

Niels Joubert* Stanford University
Mike Roberts* Stanford University
Anh Truong Stanford University
Floraine Berthouzoz Adobe Research
Pat Hanrahan Stanford University

Figure 1: *(Left) Our interactive tool for designing quadrotor camera shots. In our tool, users specify camera pose keyframes in a virtual environment using a 3D scene view (a) and a 2D map view (b). Our tool synthesizes a camera trajectory that obeys the physical equations of motion for quadrotors, and interpolates between the user-specified keyframes. Users can preview the resulting shot in the virtual environment, using the playback buttons and scrubber interface to navigate through the shot (c). Users can also control the precise timing of the shot by editing easing curves (d). Users can set the virtual camera's field of view to match their real-world camera (e). Our tool provides the user with visual feedback about the physical feasibility of the resulting trajectory, notifying the user if her intended trajectory violates the physical limits of her quadrotor hardware (f). Once the user is satisfied with her shot, she presses the Start Capture button (g). (Right) Our tool commands a quadrotor camera to execute the resulting trajectory fully autonomously, capturing real video footage that is faithful to the virtual preview. We show frames from our real-world video output, with corresponding frames from the virtual preview shown as small insets (h).*

Abstract

Cameras attached to small quadrotor aircraft are rapidly becoming a ubiquitous tool for cinematographers, enabling dynamic camera movements through 3D environments. Currently, professionals use these cameras by flying quadrotors manually, a process which requires much skill and dexterity. In this paper, we investigate the needs of quadrotor cinematographers, and build a tool to support video capture using quadrotor-based camera systems. We begin by conducting semi-structured interviews with professional photographers and videographers, from which we extract a set of design principles. We present a tool based on these principles for designing and autonomously executing quadrotor-based camera shots. Our tool enables users to: (1) specify shots visually using keyframes; (2) preview the resulting shots in a virtual environment; (3) precisely control the timing of shots using easing curves; and (4) capture the resulting shots in the real world with a single button click using commercially available quadrotors. We evaluate our tool in a user study with novice and expert cinematographers. We show that our tool makes it possible for novices and experts to design compelling and challenging shots, and capture them fully autonomously.

Keywords: robotics, quadrotors, camera animation

1 Introduction

It is now possible to mount a high-resolution camera on a quadrotor aerial vehicle, and create beautiful aerial cinematography. Quadrotors have become particularly popular because of their maneuverability, small size, and low cost. Unfortunately, flying quadrotors is difficult, even for expert users. Typically, users control quadrotors with hand-held joysticks, which requires manual dexterity and practice. Flying a quadrotor with a camera mounted to it is even more challenging, because both the quadrotor and camera must be simul-

*Niels Joubert and Mike Roberts contributed equally to this work.

taneously controlled. Quadrotors can also be flown in autonomous mode, where users design flight paths by specifying waypoints in an offline tool. However, existing flight planning tools are not designed for cinematography: they do not allow users to edit the visual composition of their shot; they do not allow users to preview what their shot will look like; they do not give users precise control over the timing of their shot; and they allow users to create shots that do not respect the physical limits of their quadrotor hardware, which can cause the quadrotor to deviate significantly from the intended trajectory, or even crash.

In this paper, we introduce an interactive tool for designing quadrotor camera shots. Our tool assists users before capture, and assumes full control during capture. In doing so, our tool enables novices and experts to capture high-quality aerial footage. To inform the design of our tool, we conducted formative interviews with professional quadrotor photographers and videographers, and we accompanied them on professional quadrotor shoots. From this study, we extracted a set of design principles for building useful quadrotor camera shot planning tools. Our interactive interface (see Figure 1) instantiates these principles by: (1) allowing users to specify shots visually in a realistic 3D GOOGLE EARTH environment; (2) providing a virtual preview of the entire shot; (3) providing users with precise control over the timing of the shot; and (4) notifying users if their intended shot violates the physical limits of their quadrotor hardware. Together, these features enable cinematographers to quickly design compelling and challenging shots, focusing on their artistic intent rather than the specific controls of the aircraft.

To build our tool, we rely on a physical quadrotor camera model, in which a rigid body quadrotor is attached to a camera mounted on a gimbal. We analyze the dynamics of our model, and show that camera trajectories must be C^4 continuous in order to obey the physical equations of motion for quadrotors. With this requirement in mind, we derive an algorithm for synthesizing C^4 continuous camera trajectories from user-specified keyframes and easing curves. This algorithm enables users to design shots visually, and gives users precise control over the timing of their shot. We then derive an algorithm to compute the control signals required for a quadrotor and gimbal to follow any C^4 continuous camera trajectory. This algorithm enables our tool to provide the user with visually accurate shot previews, and visual feedback about the physical feasibility of camera trajectories.

We use our tool to generate a variety of quadrotor camera shots. We show a shot captured using our tool in Figure 1. We evaluate our tool in a user study with four cinematographers. Two of our users are expert quadrotor pilots, and the other two had almost no quadrotor experience. All of our users appreciated how easy it was to design compelling and challenging shots using our tool. Novices stated that our tool would empower them to shoot high-quality aerial footage, a skill otherwise inaccessible to them, and experts stated that our tool would improve and extend their existing workflow.

2 Related Work

Designing Trajectories for Physical Cameras The DJI GROUND STATION [DJI 2015b] and the APM MISSION PLANNER [APM 2015] systems allow users to design quadrotor camera trajectories by placing waypoints on a 2D map. The QGROUNDCONTROL system [Meier et al. 2012] allows users to design quadrotor camera trajectories by placing waypoints in a 3D scene. However, these tools do not allow users to edit the visual composition of shots, do not provide a virtual preview, do not provide precise timing control, and do not provide feasibility feedback. Tools for designing physical camera trajectories that support these cinematography-oriented features have been developed, such as the BOT & DOLLY IRIS camera control system[1]. However, these tools are not applicable to quadrotor cameras. In our tool, we enable cinematography-oriented features (visual editing, virtual preview, precise timing control, feasibility feedback) in a tool for quadrotor cameras, and thus more effectively assist quadrotor cinematographers.

The 3D ROBOTICS SOLO [3D Robotics 2015] and DJI GO [DJI 2015a] systems allow users to interactively modify quadrotor camera shots during flight. These systems allow users to control the orientation of the camera and speed of the quadrotor as it flies between pre-defined waypoints [3D Robotics 2015], or in a circular orbit around a point of interest [DJI 2015a]. Whereas these systems can be used to *modify* shots as they are being executed, our system can also be used to precisely *design* shots before they are executed. Moreover, the 3D ROBOTICS SOLO and DJI GO systems have autonomous flight modes that will track a moving target object, whereas our system can be used to design shots where there is no particular target object.

Designing Trajectories for Virtual Cameras Designing trajectories for virtual cameras is a classical problem in computer animation. See the comprehensive survey by Christie et al. [2008]. We discuss directly related work not included in this survey here. Oskam et al. [2009] and Hsu et al. [2013] generate camera trajectories by solving a discrete optimization problem on a graph representation of a scene. Both of these methods refine the resulting discrete trajectory, either by using an iterative smoothing procedure [Oskam et al. 2009], or by solving a continuous optimization problem [Hsu et al. 2013]. Existing methods for synthesizing virtual camera trajectories guarantee C^1 or C^2 continuity. However, we demonstrate in Section 8 that camera trajectories must be C^4 continuous in order to obey the physical equations of motion for quadrotors. With this requirement in mind, our tool synthesizes C^4 camera trajectories.

Designing Trajectories for Quadrotors Our method for synthesizing quadrotor camera trajectories is similar to the trajectory synthesis methods introduced by Mellinger and Kumar [2011] and Richter et al. [2013]. These methods make the observation that there exists a *reduced state space* in which all smooth trajectories are guaranteed to obey the physical equations of motion for quadrotors. Based on this observation, they synthesize trajectories by optimizing piecewise polynomials in the reduced state space. We use a similar approach, but adapt it to quadrotor cinematography.

Quadrotors Equipped with Robotic Arms Our quadrotor camera model builds on a growing literature describing physical models for quadrotors equipped with robotic arms [Lippiello and Ruggiero 2012; Kim et al. 2013; Yang and Lee 2014; Ruggiero et al. 2015]. This literature is closely related to our work, in the sense that the camera in our quadrotor camera model can be thought of as a very short, very lightweight, single link, fully actuated robotic arm. Whereas existing approaches focus on designing feedback control policies to follow given trajectories, our approach focuses on synthesizing these trajectories subject to high-level user constraints.

Object Tracking using Quadrotors Computer vision algorithms and feedback control policies have been developed to track moving target objects using quadrotors equipped with cameras [Teuliere et al. 2011]. These approaches could be immediately applied to quadrotor cinematography. However, existing approaches *react* to moving target objects by optimizing the *position* of the quadrotor. In contrast, we globally optimize the entire *trajectory* of the quadro-

[1] BOT & DOLLY is now defunct, and the specifications for the IRIS camera control system are no longer publicly available.

tor, and we do not assume the presence of a particular target object.

Quadrotors in Computer Graphics Quadrotors have very recently been applied to problems in computer graphics. Srikanth et al. [2014] introduce a feedback control policy for maneuvering a quadrotor with a non-orientable light attached to it. Their control policy positions the quadrotor relative to a target object, so as to achieve a particular lighting effect when viewed from a stationary camera positioned elsewhere in the scene. As in our work, Srikanth et al. computationally control quadrotors to achieve an aesthetic visual objective. However, they optimize the *position* of a light in a scene, whereas we optimize the *trajectory* of a camera *through* a scene.

3 Design Principles

In order to design more effective tools for quadrotor camera control, we began by analyzing manuals on cinematography [Mascelli 1965; Arijon 1976; Katz 1991], as well as conducting formative interviews. We interviewed six professional photographers and videographers. Their level of expertise with quadrotor cameras ranged from novice to expert. We accompanied two of the quadrotor experts to professional quadrotor shoots. All participants primarily fly quadrotors manually, but have used existing trajectory planning tools. Each interview lasted approximately an hour. We asked them 30–40 questions pertaining to their setup, their preparations before capture, their workflow during capture, their post-processing steps, and their wish list for quadrotor cinematography. From this study, we extracted a set of design principles for building effective quadrotor camera planning tools.

Allow Users to Design Shots Visually All participants were primarily concerned with the visual contents of a shot. For this reason, when flying the quadrotor manually, they relied heavily on a real-time video feed from the camera to decide whether the current shot captures their artistic intent. Therefore, an effective tool for planning quadrotor camera trajectories should allow users to design shots visually.

Produce Visually Accurate Shot Previews Tools for designing camera trajectories should provide a preview of the entire shot. This preview needs to be visually accurate. In other words, the frames from the preview shot need to be as visually similar as possible to the real captured frames. Guaranteeing visual accuracy is challenging, because the physical dynamics of quadrotors impose constraints on the kinds of camera paths that can be executed. If a shot planning tool does not consider these dynamics when synthesizing camera paths, the quadrotor can deviate significantly from the intended shot during capture, reducing the accuracy of the visual preview. Therefore, an effective tool should consider the physical dynamics of quadrotors when synthesizing trajectories, in order to create visually accurate shot previews.

Give Users Precise Timing Control Several participants expressed how critical it is to be able to control the timing of a shot. Indeed, controlling the timing of a shot enables users to specify ease-in and ease-out behavior, which is important in cinematography [Arijon 1976; Lasseter 1987]. Therefore, an effective tool should allow users to precisely control the shot's visual progression over time.

Consider Physical Hardware Limits Quadrotor cameras have inherent physical limits, such as limited maximum thrust, limited maximum velocity, and a limited range of joint angles that are achievable on the camera gimbal. Attempting to fly a trajectory that does not respect these physical limits can cause the quadrotor to deviate significantly from the intended trajectory, or even crash. Indeed, several participants reported destroying equipment in accidents where they misjudged the safety of their camera trajectory or the abilities of their hardware. Therefore, it is crucial for an effective tool to consider the physical limits of the aircraft.

Provide Users with Spatial Awareness Participants often reasoned about the path a camera takes through space. For example, some participants verbally describe shots by saying *"move from here to there while keeping this in view"* or *"circle around a point"*. Moreover, users are concerned with the quadrotor's safety around obstacles. Therefore, an effective tool should provide a virtual environment that is accurately aligned to the real shot location, in order to provide users with meaningful spatial awareness.

Support Rapid Iteration and Provide Repeatability Cinematographers often perform multiple *takes* of the same shot [Mascelli 1965]. Between takes, they tweak elements of the scene until they achieve their artistic vision. In support of this workflow, participants expressed the need for tools that support iteration and repeatability with quadrotors. In outdoor environments where lighting and weather conditions can change rapidly and greatly affect the quality of a shot, it is important for users to be able to repeat the same shot multiple times. In addition, an effective tool should allow users to rapidly iterate, supporting the creative process of exploring and designing shots.

4 User Interface

We reify the design principles described in Section 3 into an interactive tool for planning and capturing quadrotor camera shots (Figure 1). In our tool, the user specifies camera pose keyframes at specific times in a virtual environment. Our tool synthesizes a camera trajectory that obeys the physical equations of motion for quadrotors, and interpolates between the user-specified keyframes.

In our tool, a camera pose *keyframe* consists of a *look-at* position, and a *look-from* position. Our tool interpolates these vectors separately to synthesize a camera pose *trajectory*. For simplicity, we always set the camera's *up* vector equal to the world-frame *up* vector. If artistic control of the camera's *up* vector is desired, our keyframe representation could be straightforwardly modified to include an *up* vector.

Editing the Visual Content and Timing of Shots Our tool provides a 3D view of a virtual scene using GOOGLE EARTH (Figure 1a). The user can set keyframes in this view by moving the virtual camera using a trackball interface. This interface enables the user to design shots visually. Our tool also provides a 2D map view of the scene using GOOGLE MAPS (Figure 1b). The user can set keyframes in this view by dragging look-from and look-at markers around the 2D map.

The user can add, edit, and delete keyframes using the 3D scene view and the 2D map view. These views are linked: edits in one view instantly update the other view. Whenever a keyframe is added, edited, or deleted, our tool synthesizes a new camera trajectory in real-time. Our tool draws the camera trajectory on the 2D map view as a curve, and in the 3D scene view as a rollercoaster-style track, to support spatial awareness.

The user can also change the total duration of her shot, and navigate through time using a scrubber interface (Figure 1c). To set a keyframe at a specific time, the user scrubs to that moment in time and edits the camera pose, as described above. When the user clicks

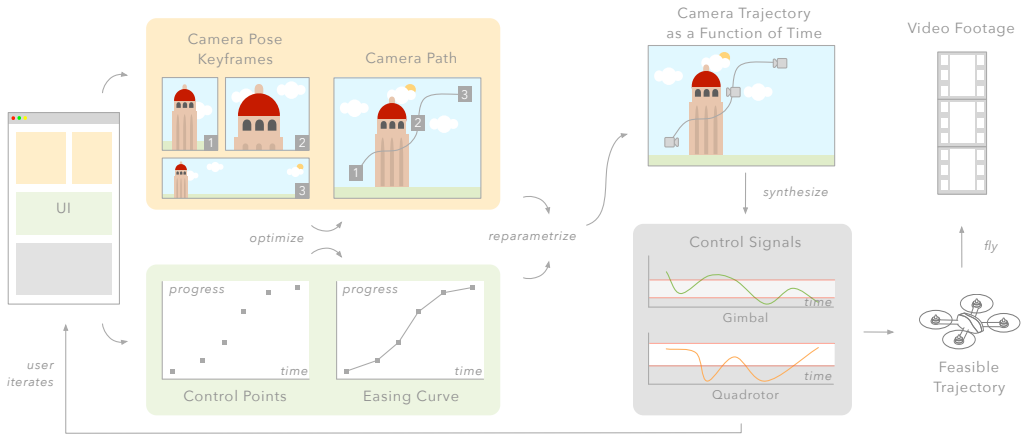


Figure 2: *Overview of the major technical components of our system. We begin with two user-specified inputs: (1) camera pose keyframes in a virtual environment (e.g.,* GOOGLE EARTH*); and (2) a sequence of easing curve control points. From these inputs, we compute a smooth camera path and a smooth easing curve. We optimize the smoothness of the camera path and easing curve in a way that obeys the physical equations of motion for quadrotors. We re-paramterize the camera path, according to the easing curve, to produce a camera trajectory as a function of time. We synthesize the control signals required for a quadrotor and gimbal to follow the camera trajectory. We plot these control signals in our user interface, providing the user with visual feedback about the physical feasibility of the resulting trajectory. The user can edit the resulting trajectory by editing camera pose keyframes and easing curve control points. Once the user is satisfied with the trajectory, we command a quadrotor camera to execute the trajectory fully autonomously, capturing real video footage.*

the *Play* button or moves the scrubber, our tool instantly plays back a preview of the shot. This functionality, when combined with our strategy for reasoning about the physical feasibility of camera trajectories, allows the user to accurately preview her shot, and supports rapid iteration.

The user can edit distinct easing curves for look-at and look-from position trajectories (Figure 1d). The user can add, edit, and delete control points on these easing curves. Editing these easing curves enables the user to precisely control the timing control of her shot.

Fixing Physically Infeasible Shots Our tool synthesizes camera trajectories that are guaranteed to obey the physical equations of motion for quadrotors. However, the user can specify shots in our tool that exceed the physical limits of her quadrotor hardware. For example, the user might specify two keyframes so close together in time, but so far apart in space, that her quadrotor cannot fly fast enough to capture the shot. Our tool provides the user with visual feedback about the physical feasibility of her trajectory, notifying the user if her intended trajectory violates the physical limits of her hardware.

Every time the user edits her shot, our tool re-calculates dynamic and kinematic quantities of interest along the camera trajectory in real-time (e.g., gimbal joint angles, velocities, and thrust forces). Our tool plots these quantities on a set of *feasibility plots* (Figure 1f). In each plot, our tool shows the physical limits of the quantity with two horizontal red lines. If any dynamic or kinematic quantity exceeds these physical limits, our tool highlights the corresponding feasibility plot. Our tool also highlights any infeasible regions directly in the 3D scene view (Figure 1a), the 2D map view (Figure 1b), and on the easing curves (Figure 1d). In each of these views, our tool colors each point along the trajectory according to the magnitude of the feasibility violations that occur at that point. Based on this visual feedback, the user can adapt her shot to the physical limits of her hardware.

Capturing Real Video Footage At any time during the design process, the user can save her shot. Once the user is pleased with her shot, she can take a laptop running our tool, and her quadrotor, to the approximate real-world starting location of her shot. The user can initiate an automatic capture session by clicking the *Start Capture* button (Figure 1g). Once the user clicks this button, our tool commands a quadrotor camera to execute the user-specified shot fully autonomously, capturing real video footage.

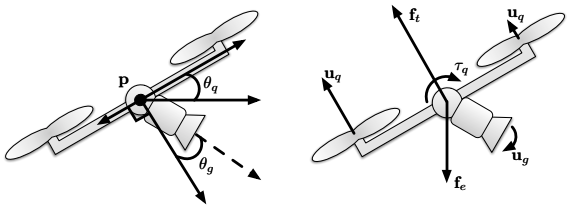
5 Technical Overview

We provide an overview of the major technical components of our system in Figure 2. At the core of our system is a physical quadrotor camera model, in which a rigid body quadrotor is attached to a camera mounted on a gimbal (Section 6). In this model, the quadrotor and the gimbal are physically coupled, which enables us to consider their motion jointly.

We analyze the dynamics of our model, and show that camera trajectories must be C^4 continuous in order to obey the physical equations of motion for quadrotors. With this requirement in mind, we derive an algorithm for synthesizing C^4 continuous camera trajectories from user-specified keyframes and easing curves (Section 7). This algorithm enables users to design shots visually, and gives users precise control over the timing of their shot. At a high level, our approach is to optimize the smoothness of the camera trajectory by solving a constrained quadratic minimization problem that guarantees C^4 continuity.

We then derive an algorithm to compute the control signals required for a quadrotor and gimbal to follow any C^4 continuous camera trajectory (Section 8). This algorithm enables our tool to provide the user with visually accurate shot previews, and visual feedback about the physical feasibility of camera trajectories. At a high level, our approach is to compute a trajectory through our quadrotor camera's state space that places the gimbal at the same world frame pose as the camera we are trying to follow at all times. We use this state space trajectory to solve for the quadrotor and gimbal control signals.

Our algorithm for synthesizing camera trajectories is guaranteed to produce trajectories that obey the physical equations of motion for quadrotors. However, our algorithm might produce trajectories that exceed the physical limits of a particular real-world quadrotor. As



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

Figure 3: *Overview of our quadrotor camera model, shown in 2D for simplicity. (a) Degrees of freedom. We model the physical state of a quadrotor camera with the following degrees of freedom: the position of the quadrotor in the world frame,* $\mathbf{p}$*; the orientation of the quadrotor in the world frame,* θ_q*; and the orientation of the gimbal in the body frame of the quadrotor,* θ_g*. Note that the orientation of the gimbal is defined relative to the orientation of the quadrotor. (b) Forces and torques. We maneuver the quadrotor by applying thrust control at the propellors,* $\mathbf{u}_q$*. This generates a net thrust force* $\mathbf{f}_t$*, and a net torque* τ_q*, 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. We orient the camera by applying a torque control at the gimbal,* $\mathbf{u}_g$*. Note that thrust is always aligned with the quadrotor's local up direction.*

discussed in Section 4, our strategy for handling these physically infeasible trajectories is interactive.

Once the user is satisfied with her camera trajectory, we command a quadrotor camera to execute the trajectory fully autonomously, capturing real video footage (Section 9). At a high level, we use the camera trajectory computed in Section 7 to drive a feedback controller running on a real-world quadrotor. This feedback controller compensates for unexpected disturbances, unmodeled forces, and sensor noise, without having to explicitly re-compute the camera trajectory. We execute the user's intended camera trajectory by sampling the position and velocity of look-at and look-from points along the trajectory, and transmitting these quantities to the quadrotor. Strictly speaking, we could attempt to execute the camera trajectories computed in Section 7, without going to the extra trouble of computing control signals in Section 8. However, computing control signals enables our tool to provide visual feedback about the physical feasibility of trajectories, which is an important safety feature. Moreover, computing control signals enables our tool to *certify* the accuracy of visual shot previews, since the visual preview will be accurate only if the trajectory is physically feasible.

6 A Quadrotor Camera Model

In this section, we introduce our physical quadrotor camera model, in which a rigid body quadrotor is attached to a camera mounted on a gimbal. We model the gimbal as a ball-and-socket joint that is rigidly attached to the quadrotor's center of mass. We provide an overview of our model in Figure 3.

Our model assumes that the quadrotor can be maneuvered by applying thrust forces at the propellers, and that the camera can be oriented by applying a torque to a ball-and-socket joint at the quadrotor's center of mass. We refer to these forces and torques as *control inputs*, since we apply them to control the physical state of the quadrotor camera. Our goal in this section is to express the equations of motion that relate the physical state of the quadrotor to the control inputs.

Degrees of Freedom and Control Inputs We denote all the degrees of freedom in our quadrotor camera model with the vector $\mathbf{q}$. This 9-dimensional vector includes the position and orientation of the quadrotor in the world frame, as well as the orientation of the camera in the body frame of the quadrotor. We use Euler angles to represent the orientation of the quadrotor and the orientation of the camera. We denote all the control inputs in our model with the vector $\mathbf{u}$. This 7-dimensional vector includes the upward thrust forces applied at each of the quadrotor's four propellers, as well as the torque applied at the gimbal.

Physical Limits We assume that we have limited control authority over our quadrotor camera model, and that our quadrotor camera model can only access a box-shaped region of its state space. This allows us to model several common physical limitations of existing quadrotor camera systems: (1) propellers can only generate bounded thrust; (2) quadrotors have maximum speeds imposed by their internal flight control software; and (3) gimbals can only be oriented within a particular frustum. We refer to constraints on $\mathbf{q}$ and $\dot{\mathbf{q}}$ as *state constraints*. We refer to constraints on $\mathbf{u}$ as *actuator limit constraints*.

Relating the Quadrotor Camera State to the Control Inputs We relate the physical state of the quadrotor camera to the control inputs as follows,

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

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 state constraints and actuator limit constraints of our system. This equation fully determines the evolution of our quadrotor camera model over time. Tedrake [2014] refers to the form of this as *manipulator form*. The matrices in this equation, known as the *manipulator matrices*, can be obtained by augmenting the quadrotor dynamics model presented by Mellinger and Kumar [2011] to include a fully actuated 3 degree-of-freedom gimbal. We include a concise definition for these matrices in Appendix A, and a more detailed derivation in the supplementary material.

7 Synthesizing Virtual Camera Trajectories

In this section, we consider the problem of synthesizing a camera trajectory from a sequence of user-specified camera pose keyframes and easing curve control points. At a high level, our approach is to smoothly interpolate our camera pose keyframes to produce a camera path. Likewise, we smoothly interpolate our easing curve control points to produce an easing curve. We optimize the smoothness of these curves by solving a constrained quadratic minimization problem that guarantees C^4 continuity. We justify this continuity requirement explicitly in Section 8.

We follow the standard practice in computer graphics [Parent 2007] of decoupling the spatial and temporal specification of camera motion: the *camera path* defines *where* the camera should go, but does not define *when* the camera should go there. In order to define a *camera trajectory* as a function of time, we re-parameterize the camera path according to the progression in the easing curve.

Representing Camera Paths and Easing Curves as Piecewise Polynomials Any piecewise polynomial representation of degree 5 or higher has enough free coefficients to enforce C^4 continuity. In the supplementary material, we evaluate alternative polynomial

representations for quadrotor camera paths. In our experience, we found that 7th degree piecewise polynomials produce the smoothest and most reasonably bounded control signals for quadrotors. For this reason, we choose to represent camera paths and easing curves using 7th degree piecewise polynomials.

We represent curves through 3D space with a distinct piecewise polynomial for each dimension. We represent camera pose trajectories with two distinct piecewise polynomial curves through 3D space: one for the *look-from* point, and another for the *look-at* point.

Optimizing the Smoothness of Piecewise Polynomials Constraining a 7th degree piecewise polynomial to be C^4 continuous does not fully determine its coefficients. To choose a particular set of coefficients, our approach is to optimize the overall smoothness of the resulting curve. We describe our approach for optimizing the smoothness of our curves in this subsection.

Suppose we are given $k+1$ scalar keyframe values, $v_{0:k}$, placed at the scalar parameter values, $u_{0:k}$. We would like to find k distinct polynomial segments that stitch together to produce a C^4 continuous curve that exactly interpolates our keyframes, and we would like the resulting curve to be as smooth as possible. Our approach here is similar to the quadrotor trajectory synthesis approach of Mellinger and Kumar [2011].

Stating our problem formally, let $\mathbf{c}$ be the vector of all the polynomial coefficients for all the distinct polynomial segments. Let $\mathbf{d}_{i,j}$ be the j^{th} derivative of the piecewise polynomial curve p with respect to the scalar parameter u at keyframe i. Let $\mathbf{d}$ be the vector of all such derivatives. We would like to find the optimal set of coefficients and derivatives, $\mathbf{c}^*$ and $\mathbf{d}^*$ respectively, as follows,

$$\mathbf{c}^*, \mathbf{d}^* = \arg\min_{\mathbf{c},\mathbf{d}} \sum_{i=0}^{k-1} \int_0^1 \left(\frac{d^4}{d\bar{u}_i^4} p_i \right)^2 d\bar{u}_i$$

$$\text{subject to} \quad p_i(0) = v_i \qquad p_i(1) = v_{i+1}$$
$$\frac{d^j}{d\bar{u}_i^j} p_i(0) = w_i^j \mathbf{d}_{i,j} \qquad \frac{d^j}{d\bar{u}_i^j} p_i(1) = w_i^j \mathbf{d}_{i+1,j} \tag{2}$$

where p_i is the i^{th} polynomial segment; $\bar{u}_i = \frac{u-u_i}{u_{i+1}-u_i} \in [0,1]$ is a normalized scalar parameter used to evaluate p_i; $j \in \{1,2,3,4\}$ is an index that refers to the various derivatives of our polynomial segments; and $w_i = u_{i+1} - u_i$ is the width of the i^{th} polynomial segment in non-normalized parameter space.

The objective function in this optimization problem attempts to make the resulting curve as smooth as possible. The equality constraints in this optimization problem ensure that our keyframes are correctly interpolated, and that the derivatives of adjacent polynomial segments match, taking into account that some segments are wider than others in non-normalized parameter space. In the supplementary material, we evaluate alternative objective functions. In our experience, we found that minimizing the 4th derivative of our polynomials produced the smoothest and most reasonably bounded control signals for quadrotors. For this reason, we choose to minimize the 4th derivative of our polynomials.

We can express the optimization problem in equation (2) as a constrained quadratic minimization problem as follows,

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{x}^T \mathbf{Q} \mathbf{x} \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{3}$$

where $\mathbf{x}$ is the concatenated vector of our coefficients and derivatives; $\mathbf{Q}$ is the symmetric positive definite matrix obtained by expanding the expression $\int_0^1 \left(\frac{d^4}{d\bar{u}_i^4} p_i \right)^2 d\bar{u}_i$ from equation (2); $\mathbf{A}$ is the matrix and $\mathbf{b}$ is the vector that can be obtained by expressing the equality constraints from equation (2) in matrix form. The problem in equation (3) can be solved by solving the following linear system,

$$\begin{bmatrix} 2\mathbf{Q} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix} \tag{4}$$

where λ is the Lagrange multiplier variable obtained by transforming equation (3) into unconstrained form [Boyd and Vandenberghe 2004].

When solving the constrained quadratic minimization problem in this section, we found that spacing our camera pose keyframes in non-normalized parameter space according to a *chordal parameterization* [Yuksel et al. 2011] helped to produce well-behaved smooth camera paths. To that end, we also constrained the 1st derivatives at the endpoints of our camera path as we would for Natural Cubic Splines [Bartels et al. 1987].

Re-parameterizing Camera Paths as Functions of Time At this point, we have defined a *camera path* through space, and an *easing curve* that defines the progress of the camera over time. In order to define a *camera trajectory* as a function of time, we re-parameterize the path according to the progression given in the easing curve using standard numerical techniques [Guenter and Parent 1990].

Our camera path is C^4 continuous with respect to u, and our easing curve is C^4 continuous with respect to time. Therefore our camera trajectory will be C^4 continuous with respect to time after this re-parameterization step.

8 Synthesizing State Space Trajectories and Control Trajectories

In this section, we consider the problem of synthesizing a *state space trajectory* and corresponding *control trajectory* that will command our quadrotor and gimbal to follow a given *virtual camera trajectory* in the world frame. At a high level, our approach is to compute a trajectory through our quadrotor camera's state space, that places the gimbal at the same world frame pose as the virtual camera we are trying to follow at all times. We then substitute this *state space trajectory* into equation (1) to solve for the corresponding *control trajectory*. Note that the quadrotor's orientation is partially determined by its direction of acceleration (see Listing 1). Therefore, we must use the available degrees of freedom in the gimbal, to align the orientation of the gimbal with the orientation of the virtual camera we are trying to follow.

Computing a State Space Trajectory In this subsection, we compute a state space trajectory for our quadrotor camera as a function of a given virtual camera trajectory. We assume that the virtual camera trajectory has been discretized into a sequence of $T+1$ camera poses evenly spaced in time. We also assume that the virtual camera trajectory is C^4 continuous. We justify this continuity requirement explicitly at the end of this section.

We begin by numerically computing the linear acceleration of the virtual camera along the trajectory using finite differences. At each moment in time along the trajectory, we solve for the degrees of freedom in our quadrotor camera model as follows,

1. Set the position of the quadrotor equal to the position of the virtual camera.
2. Compute the orientation of the quadrotor based on the acceleration and orientation of the virtual camera (see Listing 1). In this step, we align the quadrotor's orientation to its direction

of acceleration. This approach guarantees that the quadrotor's orientation is always consistent with equation (1). Or stated more precisely, that the state space trajectory we compute in this section, when substituted into equation (1), always yields a left hand side that is in the column space of the matrix $\mathbf{B}$.

Our algorithm here is similar to the algorithm presented by Mellinger and Kumar [2011]. However, we adapt their algorithm to determine the quadrotor's orientation from the virtual camera's orientation (and its direction of acceleration), rather than requiring the quadrotor's yaw angle to be specified explicitly. This is an important practical difference, since it allows users to specify shots visually, rather than having to explicitly specify yaw angles.

3. Compute the orientation of the gimbal in the body frame of the quadrotor, based on the orientation of the virtual camera and quadrotor in the world frame. For this step, we use the relationship $\mathbf{R}_{\mathcal{W},\mathcal{C}} = \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{R}_{\mathcal{Q},\mathcal{G}}$, where $\mathbf{R}_{\mathcal{W},\mathcal{C}}$ is the rotation matrix that represents the orientation of the virtual camera in the world frame; $\mathbf{R}_{\mathcal{W},\mathcal{Q}}$ is the rotation matrix that represents the orientation of the quadrotor in the world frame; and $\mathbf{R}_{\mathcal{Q},\mathcal{G}}$ is the rotation matrix that represents the orientation of the gimbal in the body frame of the quadrotor.

At this point, we have solved for the position and orientation of our quadrotor, as well as the orientation of our gimbal, at every moment in time along the discretely sampled virtual camera trajectory. We compute the Euler angle representations of the quadrotor and gimbal orientations using standard numerical techniques [Diebel 2006]. In doing so, we have solved for the state space trajectory, corresponding to the given virtual camera trajectory.

Uniqueness The state space trajectory we compute above is not unique. There are other state space trajectories that will follow the given virtual camera trajectory. For example, the quadrotor could be at a different yaw angle, and the gimbal could also be at a different orientation to compensate. Among this family of valid state space trajectories, our algorithm computes the state space trajectory that sets the gimbal's yaw angle to zero, while minimizing the magnitude of the gimbal's pitch angle (Listing 1, lines 3-5). This approach means our algorithm can be used without modification on quadrotor cameras with 2 degree-of-freedom gimbals, as well as the 3 degree-of-freedom gimbal we assume in our model.

Computing a Control Trajectory In this subsection, we compute a control trajectory $\mathbf{u}_{0:T}$, as a function of our state space trajectory $\mathbf{q}_{0:T}$. We begin by computing the 1st and 2nd derivatives of our state space trajectory, $\dot{\mathbf{q}}_{0:T}$ and $\ddot{\mathbf{q}}_{0:T}$ respectively, using finite differences. We compute our control trajectory by repeatedly substituting $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ into equation (1), and solving for $\mathbf{u}$, at each moment in time along the discretely sampled state space trajectory. We use the Moore-Penrose pseudoinverse of $\mathbf{B}$ to invert equation (1), which in this case, is guaranteed to yield an exact unique solution for $\mathbf{u}$. This is because we explicitly constructed $\mathbf{q}_{0:T}$ 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. We include a proof that $\mathbf{B}$ is always full column rank in the supplementary material.

C^4 Continuity A virtual camera trajectory must be at least C^4 continuous with respect to time if we hope to synthesize a control trajectory to follow it. At a high level, this continuity requirement arises from the fact that a quadrotor can only apply thrust forces along its local *up* axis. Indeed, we see in Listing 1 (lines 1–3) that we use the 2nd derivative of the virtual camera position $\ddot{\mathbf{p}}_c$ to solve for the quadrotor's orientation degrees of freedom. Moreover, we see in equation (1) that we use the 2nd derivative of the quadrotor's degree-of-freedom vector $\ddot{\mathbf{q}}$ to solve for the control input $\mathbf{u}$. Therefore, the control input $\mathbf{u}$ is a function of the 4th derivative of the virtual camera trajectory. If a virtual camera trajectory is not at least C^4 continuous, then the control input will not be well-defined across the trajectory. This continuity requirement is also noted by Mellinger and Kumar [2011].

Input:

- Acceleration of the virtual camera in the world frame, $\ddot{\mathbf{p}}_c$.
- Virtual camera's local $\mathbf{x}$ axis (i.e., the look-at vector) in the world frame, $\mathbf{x}_c$.
- External force in the world frame, $\mathbf{f}_e$.
- Mass of the quadrotor camera, m.

Output:

- Rotation matrix representing the quadrotor's orientation in the world frame, $\mathbf{R}_{\mathcal{W},\mathcal{Q}}$.

1: $\mathbf{f} \leftarrow m\ddot{\mathbf{p}}_c$
2: $\mathbf{f}_t \leftarrow \mathbf{f} - \mathbf{f}_e$
3: $\mathbf{y}_q \leftarrow$ normalized $\mathbf{f}_t$
4: $\mathbf{z}_q \leftarrow$ normalized $\mathbf{y}_q \times \mathbf{x}_c$
5: $\mathbf{x}_q \leftarrow$ normalized $\mathbf{z}_q \times \mathbf{y}_q$
6: $\mathbf{R}_{\mathcal{W},\mathcal{Q}} \leftarrow$ the rotation matrix defined by the axes $\mathbf{x}_q, \mathbf{y}_q, \mathbf{z}_q$

Listing 1: *Computing the orientation of the quadrotor in the world frame. We begin by substituting linear acceleration and mass into Newton's Second Law to solve for net force (line 1). We make the observation that we can always decompose the net force acting on our quadrotor into a thrust force and an external force, where the external force models effects like gravity, wind, and drag. With this observation in mind, we solve for thrust force (line 2). We make the observation that our quadrotor model can only generate thrust forces along its local* $\mathbf{y}$ *axis. With this observation in mind, we normalize the thrust force and set the quadrotor's local* $\mathbf{y}$ *axis equal to the normalized thrust force vector (line 3). This approach guarantees that the quadrotor's orientation is always consistent with equation (1). Or stated more precisely, that the state space trajectory we compute in Section 8, when substituted into equation (1), always yields a left hand side that is in the column space of the matrix* $\mathbf{B}$*. In our algorithm, the quadrotor's local* $\mathbf{y}$ *axis, in combination with the virtual camera's local* $\mathbf{x}$ *axis, uniquely determines the orientation of the quadrotor (lines 4–6).*

Unbounded Control Inputs The state space trajectory $\mathbf{q}_{0:T}$ we compute in this section is guaranteed to satisfy the equations of motion given in equation (1). In other words, there exists some control trajectory $\mathbf{u}_{0:T}$ that will follow $\mathbf{q}_{0:T}$. However, the control inputs required to follow $\mathbf{q}_{0:T}$ might exceed the physical limits of a particular real-world quadrotor. In general, it is not guaranteed that $\mathbf{u}_{0:T}$ and $\mathbf{q}_{0:T}$ will satisfy the actuator limit constraints and state constraints given in equation (1). We must take extra care to ensure that $\mathbf{q}_{0:T}$ and $\mathbf{u}_{0:T}$ satisfy these constraints. We address this issue interactively in our user interface, as described in Section 4.

9 Real-Time Control System and Hardware Platform

In this section, we describe the real-time control system and hardware platform we use to execute camera trajectories autonomously and capture real video footage.

Real-Time Control System We show a block diagram of our real-time control system in Figure 4. We build our real-time control system on top of the open source ARDUPILOT autopilot software [APM 2015]. The ARDUPILOT software runs on board the quadro-

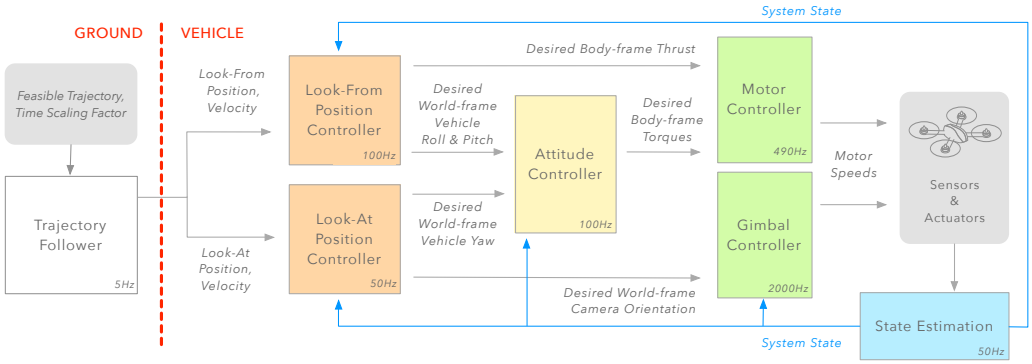


Figure 4: *Block diagram of our real-time control system for executing camera trajectories. On a ground station (left), our trajectory follower (white) samples the camera trajectory, transmitting the sampled position and velocity of look-at and look-from points to the quadrotor. Our trajectory follower allows the user to optionally adjust a time scaling factor, to execute the trajectory faster or slower. On board the quadrotor (right), the higher-level look-from and look-at position controllers interface with a lower-level attitude controller (yellow) and motor controller (green), similar to those described by Kumar and Michael [2012].*

tor, and provides a hierarchical feedback controller for following camera trajectories, similar to the controller described by Kumar and Michael [2012]. The ARDUPILOT feedback controller takes as input the position and velocity of look-at and look-from points along a camera trajectory. Our real-time control system runs on a ground station. Our system executes the user's intended camera trajectory by sampling the position and velocity of look-at and look-from points along the trajectory, and transmitting these quantities to the quadrotor.

Time Scaling and Safety While the camera trajectory is being executed, our real-time control system allows the user to optionally adjust a time scaling factor. By default, our system samples the camera trajectory uniformly in time. If the user adjusts the time scaling factor, our system applies a linear scaling to the time step used to determine the next sampling location along the trajectory. Using our time scaling functionality, we implement a *full stop* command, which is an important safety feature. Setting the time scaling factor to 0 pauses the quadrotor at its current position. This allows the user to abort capture at any time, and helps to avoid crashes.

Hardware Platform Our hardware platform consists of an 3D ROBOTICS IRIS+ quadrotor [3D Robotics 2014] running the open source ARDUPILOT autopilot software [APM 2015] on a PIXHAWK autopilot computer [Meier et al. 2012]. We equip our quadrotor with a 2-axis gimbal and a GOPRO HERO 4 BLACK camera. At the time of publication, this hardware setup is priced at $1300, and is representative of an entry-level quadrotor for aerial cinematography.

System Identification We determined the system parameters used in our quadrotor camera model, which are specific to our hardware, partially through direct measurement and partially through published engineering specifications. We used a dynamometer to measure the maximum force and torque our rotors could generate, and estimated the moment of inertia from the quadrotor's mass and shape. We used the maximum lean angles, maximum velocities, and maximum accelerations published by the ARDUPILOT community [APM 2015].

10 Evaluation and Discussion

In this section, we describe the user study we conducted to evaluate our tool, and discuss our key findings.

User Study We performed a user study aiming to understand whether our tool enables the creation of shots that would be challenging to capture otherwise. We recruited two expert cinematographers, and two novice cinematographers with computer graphics experience. Both of our expert cinematographers had extensive experience manually flying quadrotors for cinematography.

After demonstrating the capabilities of our tool in a 30 minute tutorial, we gave all four participants identical tasks. We first tasked them with creating one shot featuring the 285 foot tall Hoover Tower (i.e., the *instructed shot*). The tower was selected for its striking appearance and large scale, providing an opportunity for interesting shots that are well-suited for quadrotor cinematography. We also tasked participants with creating a second shot of their own choosing (i.e., the *freeform shot*). We instructed them to create and refine shots that are cinematically interesting, and within the physical limits of our quadrotor hardware, as visualized in our tool. They had 90 minutes to create these shots, during which we were available to answer questions. Our tool saved a log and screen recording of each session. Afterwards, they accompanied us to capture their shots, watched the resulting videos, and filled out an exit questionnaire.

All four participants successfully completed the two tasks. We show the shots from our users in Figure 5, and henceforth we refer to the shots using the lettering in this figure. The participants' shots included a wide variety of camera motions. None of the shots violated any of the kinematic or dynamic limits shown on the feasibility plots in our tool. We were able to successfully capture all eight shots. We encourage readers to watch our supplementary video, where we show these real-world shots, and the virtual previews from our tool, in a series of side-by-side comparisons.

Novices and Experts Successfully Designed Challenging Shots We asked the expert cinematographers to describe what elements were challenging about the shots they created, if they were to capture them with existing approaches for quadrotor cinematography. Each expert identified camera motions in their shots that would either take many attempts, or would have to be modified to be less challenging. We identified similar camera motions in the novice shots (see Figure 6). We summarize the similarities between novice and expert shots as follows,

- Expert shot (c) required continuous re-orientation of the camera relative to the flight path, with the look-from trajectory in red arcing away from a fixed look-at point. We found a similar arcing motion around a fixed look-at point in novice shots (g)

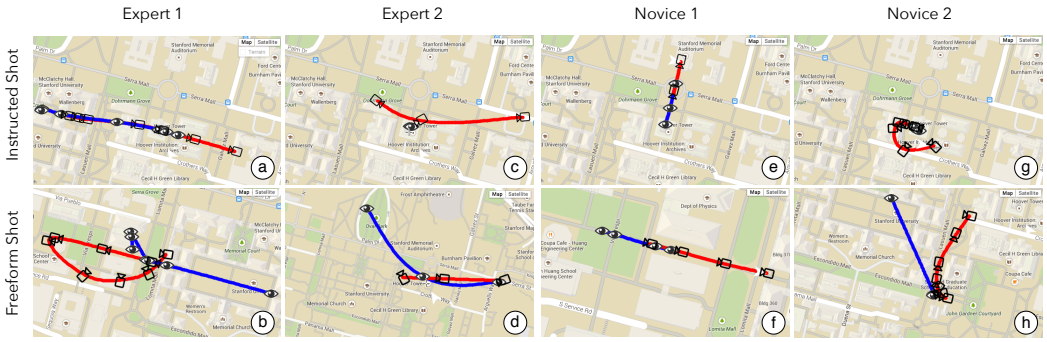


Figure 5: *Camera shots created by the two experts (left) and two novices (right) in our user study. The look-from and look-at trajectories for each shot are shown in red and blue respectively. The shots created by our participants contain a wide variety of camera motions.*

Figure 6: *Novices and experts successfully designed shots with challenging camera motions using our tool. The expert shot (top) is especially challenging to execute manually, since it requires smoothly changing the camera orientation to look down at Hoover Tower exactly as the quadrotor flies over it. The novice shot (bottom) contains a similar camera motion.*

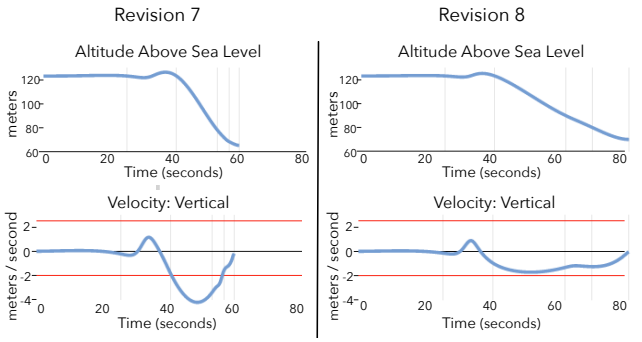


Figure 7: *Participants were able to modify infeasible shots into feasible shots using the visual feedback we provide in our tool. After his 7th revision, Expert 1 found that his shot was infeasible (left). He edited both the altitude and timing of 3 keyframes to create a feasible shot as his 8th revision (right). Horizontal red lines indicate physical limits of our quadrotor hardware.*

and (h). This camera motion is difficult to execute manually, because it requires continuously and precisely re-orienting the camera during flight.

- Expert shot (a) required flying straight towards a point over a long distance, which we also found in novice shot (f). This camera motion is difficult to execute manually, since small initial errors in the direction of flight have to be corrected, leading to visual artifacts in the resulting video.
- Expert shot (a) required smoothly adjusting the rate of camera re-orientation, to end at a specific orientation at a specific time. We found this camera motion in novice shot (a). We show these two shots in Figure 6. This camera motion is especially difficult to execute manually. The camera must translate towards a point while tilting down, so that the end of the tilting motion exactly coincides with being above the tower, all while approaching the tower in a straight line. The expert that designed shot (a) remarked that executing such a shot manually would require approximately 20 attempts.

This finding suggests that users can successfully design compelling shots with challenging camera motions using our tool, regardless of their level of expertise with quadrotors.

Previewing Shots Visually was Useful Our exit survey asked participants to identify the most useful feature of our user interface, and rank the features in our user interface on a 5-point scale from *not useful* to *indispensable*. Three of the four participants identified the ability to visually preview their shot as being the most useful, and all users rated this feature as a 4 or higher. Indeed, Figure 1 shows that our visual preview accurately estimates the appearance of recorded video footage. This finding validates our approach of enabling users to design shots visually, and highlights the importance of ensuring physical feasibility during the design process.

Controlling the Timing of Shots was Useful All participants used the easing curves to refine the timing of their shots. Participants used the easing curves to modify the pacing of their shots, and to fix feasibility violations. In all shots, participants adjusted the default easing curve control points. Of the eight shots created, six featured additional control points added by the participant. This finding validates our approach of enabling users to precisely control the timing of their shots.

Visual Feasibility Feedback was Useful, Although Some Participants Would Have Preferred an Automatic Solution All of our participants responded to the visual feasibility feedback in our tool. Users successfully modified their shots until they were within the physical limits of our hardware, as shown on the feasibility plots in our tool. Figure 7 shows Expert 1 modifying both the altitude and timing of three keyframes to stay within the vertical speed limit of our quadrotor. However, participants were divided in their opinions about our feasibility plots. One participant rated it as the best part of the tool. He described it as essential to creating shots at

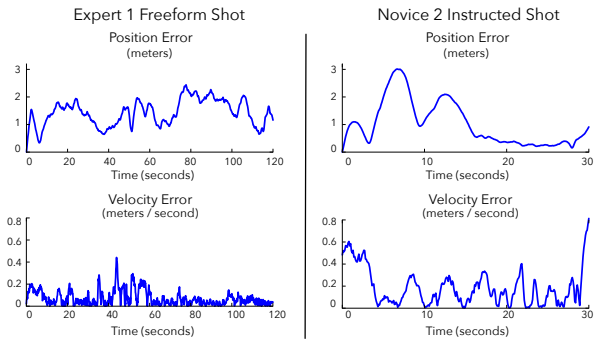


Figure 8: *Position and velocity error of our quadrotor for the longest (left) and shortest (right) shots. The position error is less than 3.01m at all times, and the velocity error is less than 0.80m/s at all times. Note that the horizontal scaling varies varies on the left and right subplots.*

the physical limits of the hardware. Another participant expressed difficulty knowing exactly how to tweak trajectories in response to the visual feedback. This finding suggests that some users would prefer an automatic solution for fixing feasibility issues, while others like precise control over their shots. We believe that developing an automatic solution to fix feasibility violations is an interesting direction for future work.

Accuracy To quantify how well our quadrotor camera system follows trajectories, we compared the intended trajectories created by our users, to the actual trajectories executed by our quadrotor (see Figure 8). The average position error across all shots was 1.12m ($\sigma = 0.57$), and was never greater than 3.01m. The average velocity error across all shots was 0.11m/s ($\sigma = 0.10$), and was never greater than 0.80m/s. In general, our system is limited by the positioning and pointing accuracy of our quadrotor. This limitation makes close-up shots particularly challenging, where small errors in position lead to more noticeable visual errors. However, our participants responded positively when they saw the captured footage for the shots they created. This finding suggests that the level of accuracy achievable with current-generation quadrotor hardware is sufficient to obtain a variety of compelling shots.

Concluding Remarks Overall, all participants were enthusiastic about using our system. Experts appreciated having a powerful tool to visually plan complex trajectories and execute repeatable takes (e.g. Expert 1 remarked *"Normally I fly less ambitious paths to avoid making mistakes!"* and *"I love how I can get the same shot, take after take, day after day!"*). Novices were particularly enthusiastic about being able to capture high-quality video footage with quadrotors without having experience flying them (e.g., Novice 2 remarked, *"I liked how it turned a 'drone flying problem' into a 'drawing a curve in space problem'. I don't know how to fly a drone and don't want to, but I find drawing in 3D very intuitive."*).

11 Conclusions

We introduced a set of design principles for quadrotor shot planning tools. Based on these principles, we built a tool for designing quadrotor camera shots. Specifically, we added four components to current quadrotor mission planning tools: (1) visual shot design; (2) virtual preview; (3) precise timing control; and (4) visual feasibility feedback. Using our tool, both novices and expert users designed compelling shots that would be challenging to create otherwise. We successfully and autonomously captured all shots with reasonable accuracy on a real quadrotor camera platform.

In the future, we believe quadrotors will enable many creative applications beyond the pre-scripted aerial cinematography shown in this paper. Quadrotor cameras might soon be able to autonomously film a person skiing down a mountain, or a pack of wild animals hunting prey. By flying the same trajectory repeatedly, quadrotor cameras could also enable new kinds of highly dynamic time-lapse video footage. Moving beyond the domain of cinematography, quadrotor cameras could help to reconstruct virtual 3D models of the physical world with unprecedented coverage and scale.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. We thank Maxine Lim for her assistance in creating figures, and for designing our project website. We thank Jane E, James Hegarty, Wilmot Li, and Jerry Talton for their valuable feedback on early drafts of this paper. We thank Andrew Tridgell, Randy MacKay, and the ARDUPILOT team for their software support. We thank 3D ROBOTICS for their hardware support. We thank the professional cinematographers we interviewed, as well as our user study participants, for their time and valuable insights. 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, 2014. IRIS+. `http://3drobotics.com/iris/`.

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

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

ARIJON, D. 1976. *Grammar of the Film Language*. Hastings House Publishers.

BARTELS, R. H., BEATTY, J. C., AND BARSKY, B. A. 1987. *An Introduction to Splines for use in Computer Graphics & Geometric Modeling*. Morgan Kaufmann Publishers.

BOYD, S., AND VANDENBERGHE, L. 2004. *Convex Optimization*. Cambridge University Press.

CHRISTIE, M., OLIVIER, P., AND NORMAND, J.-M. 2008. Camera control in computer graphics. *Computer Graphics Forum 27*, 8.

DIEBEL, J., 2006. Representing attitude: Euler angles, unit quaternions, and rotation vectors.

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

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

GUENTER, B., AND PARENT, R. 1990. Motion control: Computing the arc length of parametric curves. *Computer Graphics Applications 10*, 3.

HSU, W.-H., ZHANG, Y., AND MA, K.-L. 2013. A multi-criteria approach to camera motion design for volume data animation. *Transactions on Visualization and Computer Graphics (Proc. SciVis 2013) 19*, 12.

KATZ, S. D. 1991. *Film Directing Shot by Shot*. Butterworth Publishers.

KIM, S., CHOI, S., AND KIM, H. J. 2013. Aerial manipulation using a quadrotor with a two DOF robotic arm. In *Intelligent Robots and Systems (IROS) 2013*.

KUMAR, V., AND MICHAEL, N. 2012. Opportunities and challenges with autonomous micro aerial vehicles. *International Journal of Robotics Research 31*, 11.

LASSETER, J. 1987. Principles of traditional animation applied to 3D computer animation. In *SIGGRAPH 1987*.

LIPPIELLO, V., AND RUGGIERO, F. 2012. Exploiting redundancy in cartesian impedance control of UAVs equipped with a robotic arm. In *Intelligent Robots and Systems (IROS) 2012*.

MASCELLI, J. 1965. *The Five C's of Cinematography*. Silman-James Press.

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*.

OSKAM, T., SUMNER, R. W., THUEREY, N., AND GROSS, M. 2009. Visibility transition planning for dynamic camera control. In *Symposium on Computer Animation (SCA) 2009*.

PARENT, R. 2007. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann Publishers.

RICHTER, C., BRY, A., AND ROY, N. 2013. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *International Symposium of Robotics Research (ISRR) 2013*.

RUGGIERO, F., TRUJILLO, M., CANO, R., ASCORBE, H., VIGURIA, A., PEREZ, C., LIPPIELLO, V., OLLERO, A., AND SICILIANO, B. 2015. A multilayer control for multirotor UAVs equipped with a servo robot arm. In *International Conference on Robotics and Automation (ICRA) 2015*.

SRIKANTH, M., BALA, K., AND DURAND, F. 2014. Computational rim illumination with aerial robots. In *Computational Aesthetics (CAe) 2014*.

TEDRAKE, R., 2014. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for MIT 6.832). `http://people.csail.mit.edu/russt/underactuated/`.

TEULIERE, C., ECK, L., AND MARCHAND, E. 2011. Chasing a moving target from a flying UAV. In *Intelligent Robots and Systems (IROS) 2011*.

YANG, H., AND LEE, D. 2014. Dynamics and control of quadrotor with robotic manipulator. In *International Conference on Robotics and Automation (ICRA) 2014*.

YUKSEL, C., SCHAEFER, S., AND KEYSER, J. 2011. Parameterization and applications of Catmull-Rom curves. *Computer Aided Design 43*, 7.

A Defining the Quadrotor Camera Manipulator Matrices

In this appendix, we define the quadrotor camera manipulator matrices, attempting to be as concise as possible. We include a detailed derivation of these matrices in the supplementary material.

We begin by defining the layout of our degree-of-freedom vector $\mathbf{q}$, and our control vector $\mathbf{u}$, as follows,

$$\mathbf{q} = \begin{bmatrix} \mathbf{p} \\ \mathbf{e}_q \\ \mathbf{e}_g \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} \mathbf{u}_q \\ \mathbf{u}_g \end{bmatrix} \tag{5}$$

where $\mathbf{p}$ is the position of the quadrotor's center of mass; $\mathbf{e}_q$ is the vector of Euler angles representing the quadrotor's orientation in the world frame; $\mathbf{e}_g$ is the vector of Euler angles representing the orientation of the gimbal in the body frame of the quadrotor; $\mathbf{u}_q$ is the thrust control we apply at the quadrotor propellors; and $\mathbf{u}_g$ is the torque control we apply at the gimbal.

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

$$\begin{aligned} \mathbf{H}(\mathbf{q}) &= \begin{bmatrix} m\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{A}_g \end{bmatrix} \\ \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) &= \\ \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\dot{\mathbf{A}}_q - \left(\mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\dot{\mathbf{e}}_q\right)_\times \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \dot{\mathbf{A}}_g \end{bmatrix} \\ \mathbf{G}(\mathbf{q}) &= \begin{bmatrix} -\mathbf{f}_e \\ \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times1} \end{bmatrix} \\ \mathbf{B}(\mathbf{q}) &= \begin{bmatrix} \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f} & \mathbf{0}_{3\times3} \\ \mathbf{M}_\tau & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times4} & \mathbf{I}_{3\times3} \end{bmatrix} \end{aligned} \tag{6}$$

where m is the mass of the quadrotor camera; $\mathbf{I}_q$ is the inertia matrix of the quadrotor camera; $\mathbf{R}_{\mathcal{W},\mathcal{Q}}$ 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; $\mathbf{A}_q$ is the matrix that relates the quadrotor's Euler angle time derivatives to its angular velocity in the world frame; $\mathbf{A}_g$ is the matrix that relates the gimbal's Euler angle time derivatives to its angular velocity in the body frame of the quadrotor; $\mathbf{f}_e$ 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 quadrotor camera manipulator matrices depend on the matrices, $\mathbf{M}_\mathbf{f}$ and $\mathbf{M}_\tau$. We define these matrices as follows,

$$\begin{aligned} \mathbf{M}_\mathbf{f} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{M}_\tau &= \begin{bmatrix} ds_\alpha & ds_\beta & -ds_\beta & -ds_\alpha \\ \gamma & -\gamma & \gamma & -\gamma \\ -dc_\alpha & dc_\beta & dc_\beta & -dc_\alpha \end{bmatrix} \end{aligned} \tag{7}$$

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 camera manipulator matrices, in particular our expressions for $\mathbf{A}_q$ and $\mathbf{A}_g$, depend on our choice of Euler angle conventions. See our derivation in the supplementary material for details.