TRAJECTORY OPTIMIZATION METHODS FOR DRONE CAMERAS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Mike Roberts
May 2019

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Pat Hanrahan) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Doug James)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Frédo Durand)

Approved for the Stanford University Committee on Graduate Studies

Abstract

Drone cameras are now being deployed in a wide range of applications, including Hollywood filmmaking, search and rescue, wildlife monitoring, and large-scale 3D scanning. However, drones remain difficult to control, both for humans and for computers. Drones are difficult for humans to control, because a human operator must either: express commands using counter-intuitive hand-held joysticks; or painstakingly specify waypoints using primitive mission planning software. Drones are also difficult for computers to control, because an effective control algorithm must carefully reason about the non-linear dynamics and physical limits of the drone, obstacles in the drone's surroundings, and uncertainty in the drone's on-board sensor measurements.

In this dissertation, we introduce a variety of trajectory optimization methods that make it easier for people to use drone cameras. We focus specifically on two different applications: drone cinematography and drone 3D scanning. Throughout this dissertation, we leverage domain knowledge that is specialized to each application, and we reformulate classical trajectory optimization problems in terms of the drone camera's visual output, i.e., in terms of what the drone is *seeing*.

We begin by introducing a software tool for designing and autonomously executing cinematography shots with quadrotor cameras. 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 a commercially available quadrotor camera. To support our tool, we introduce a physical model for quadrotor cameras, and we derive an algorithm for generating camera trajectories that agree with our physical model. 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 shots, and capture them fully autonomously.

To further support our shot planning tool, we introduce a fast and user-friendly algorithm for generating camera trajectories that respect the dynamics and physical limits of quadrotor hardware. We demonstrate that our algorithm is between 25× and 180× faster than a general-purpose trajectory optimization approach. We successfully capture real video footage using the trajectories generated by our algorithm. We show that the resulting videos are faithful to virtual shot previews, even when the trajectories being executed are at our quadrotor's physical limits.

Finally, we turn our attention from cinematography to 3D scanning. We introduce an algorithm to generate drone camera trajectories, such that the imagery acquired during the flight will later produce a high-fidelity 3D model. Our method uses a coarse estimate of the scene geometry to plan camera trajectories that: (1) cover the scene as thoroughly as possible; (2) observe the scene geometry from a diverse set of viewing angles; (3) avoid obstacles; and (4) respect a user-specified flight time budget. Our method relies on a mathematical model of scene coverage that exhibits an intuitive diminishing returns property known as *submodularity*. We leverage this property extensively to design a trajectory planning algorithm that reasons globally about the non-additive coverage reward obtained across a trajectory, jointly with the cost of traveling between views. We evaluate our method by using it to scan three large outdoor scenes, and we perform a quantitative evaluation using a photorealistic video game simulator.

Acknowledgments

This dissertation would have been impossible without the help and support of many people. I am indebted to all of you, and I am delighted to take this opportunity to recognize the positive impact that all of you have had on my life.

I want to begin by thanking the world's greatest advisor, Pat Hanrahan. You took me on as your student even though I didn't know what I was doing. You were so patient with me, and you gave me the time I needed to grow into a good student and an independent researcher. You gave me the freedom to choose my own research directions. You spent countless hours with me discussing research ideas, and you provided a seemingly endless supply of helpful feedback on paper drafts and talks. Your guidance and mentorship have been transformative.

I want to thank Frédo Durand and Doug James for serving on my dissertation reading committee. Frédo's notes on writing became my bible during my graduate studies, and I especially appreciate the advice Doug gave me about choosing research problems back at Harvard. Doug and Frédo also provided very helpful and detailed feedback on an early version of my job talk. I want to thank Ron Fedkiw and Stephen Boyd for serving on my dissertation defense committee. Ron's lab was an intellectual home-away-from-home, and I especially enjoyed all of our lively discussions over the years. I also heard a rumor that Ron rescued my Stanford application from the reject pile – thank you. Stephen's textbooks, classes, online lectures, and office hours were always incredibly educational, both in their substance and style.

I want to thank Hanspeter Pfister. You were the first person to give me a chance to do research outside my home town. This opportunity changed my life. I also want to thank John Owens. You were the first person I ever met at an academic conference, and have been an exemplary role model and continuing source of inspiration ever since. I especially appreciate your guidance on re-applying to Stanford, which undoubtedly changed my life for the better. While you were hosting me for a visit at UC Davis, Shubho Sengupta offered a comfy couch to sleep on, and Andrew Davidson gave me a lift to the airport – thank you.

I want to thank my recommenders: Adam Finkelstein, Pat Hanrahan, Doug James, John Owens, Hanspeter Pfister, and Sudipta Sinha. You all submitted a gazillion recommendation letters on my behalf, and were all so incredibly generous with your time. Discussing research ideas with Adam

was always fun and inspirational. Sudipta introduced me to submodular functions, which had a big impact on this dissertation, and our whiteboard sessions were some of my favorite experiences in graduate school.

During my time at Stanford, I benefited from many other amazing mentors. Floraine Berthouzoz was an incredible role model and collaborator. Our discussions were always fun and inspirational, and our late-night paper editing sessions had a lasting impact on me. I wish I could have shared more of the ideas in this dissertation with you. Russ Tedrake carefully replied to so many of my emails, and his online lectures inspired much of the work in this dissertation. Stefanie Jegelka's tutorials and online lectures also directly inspired the work in this dissertation. Mykel Kochenderfer and Silvio Savarese agreed to participate in my qualifying exam – thank you. Vladlen Koltun and Ge Wang agreed to take me on as a rotation student even though I didn't know what I was doing. Maneesh Agrawala, Michael Bernstein, Kayvon Fatahalian, Doug James, James Landay, and Mac Schwager provided valuable ongoing advice, and were all extremely generous with their time. Wilmot Li provided detailed and helpful feedback on an early paper draft. Working with John Owens and David Luebke at Udacity was a once-in-a-lifetime opportunity. Working closely with Frank Dellaert at Skydio was incredibly fun and educational. Debadeepta Dey, Neel Joshi, Ashish Kapoor, Shital Shah, and Sudipta Sinha at Microsoft Research were all fantastic collaborators.

I couldn't have asked for better friends, colleagues, and labmates at Stanford. Niels Joubert immediately became a close collaborator and an even closer friend. Indeed, many of the ideas in this dissertation were conceived during our memorable late-night coding sessions. Anh Truong and Jane E quickly became trusted confidants and valued collaborators. I especially appreciate all of Anh's fantastic work on our 3D scanning project, and Jane's work to narrate one of our supplementary videos. Gilbert Bernstein, Zach DeVito, James Hegarty, Matt Fisher, Sharon Lin, Jonathan Ragan-Kelley, Daniel Ritchie, Manolis Savva, Lingfeng Yang, and Yi-Ting Yeh were fantastic colleagues and role models. I especially want to thank Zach and Jonathan for putting up with it when I would sleep in their office – you were doing me a bigger favor than you realize. Ross Daly and Mackenzie Leake helped to prepare figures and supplementary materials for a paper at the last possible nanosecond. Maxine Lim created several figures that appear in this dissertation. Ali Alkhatib, Will Crichton, Abe Davis, Jingyi Li, Alex Poms, and Jerry Talton provided a ton of helpful feedback on various talks and paper drafts. I especially appreciated all the conversations, coding sessions, and late-night snacks with Ed Quigley and Winnie Lin. Justin Johnson was the world's greatest group project partner – thank you for being so patient with me. Enzo Busseti and Okke Schrijvers became life-long friends.

I also had the opportunity to work with many wonderful friends, colleagues, and labmates at Harvard. Won-Ki Jeong was willing to work with me even though I didn't know what I was doing. Bjoern Andres, Verena Kaynig-Fittkau, and Amelio Vazquez-Reina inspired me to learn everything I could about optimization. Moritz Baecher, Kevin Dale, Kalyan Sunkavalli, and Justin Thaler became close friends. I especially want to thank Kevin and Kalyan for generously sharing their

accommodations at SIGGRAPH with me – you were doing me a bigger favor than you realize. Johanna Beyer, Nicolas Bonneel, Michelle Borkin, Cris Cecka, Markus Hadwiger, Daniel Haehn, Ray Jones, Bobby Kasthuri, Seymour Knowles-Barley, Jeff Lichtman, Miriah Meyer, Nicolas Pinto, and Amanda Randles were wonderful collaborators and role models.

I appreciate the technical and administrative support I received throughout my graduate studies. Weichao Qiu provided extensive technical support with his incredibly useful UnrealCV Python library. Jim Piavis and Ross Robinson provided valuable expertise as safety pilots, and Don Gillett granted us permission to 3D scan his barn. I am also thankful for the support I received from the exceptional administrative staff in the Stanford Computer Science Department. In particular, Helen Buendicho, Chris Downey, Jam Kiattinant, Andrea Kuduk, Jillian Lentz, Monica Niemiec, and Jay Subramanian have all gone out of their way to help me countless times.

The work in this dissertation was funded by NSERC and a generous grant from Google. 3DRobotics generously donated the drone hardware we used throughout this dissertation, and granted us permission to 3D scan their industrial scene. Thank you.

Moving beyond academia, I am grateful for the people in my personal life that have helped and supported me throughout my graduate career. In Calgary, Chelsea Leishman and the rest of Leishman family always believed in me. Rob Faust remains a life-long friend, and is still the best DJ partner on the planet. In Boston, Eric Baczuk and Simon Scheider were always great workout buddies, and I always enjoyed our endless conversations at Chipotle. When I got to Stanford, I was fortunate to navigate the challenges and adventures of graduate student life alongside Okke Schrijvers. Shannon Hoban and the rest of the Hoban family, Danielle Radin, Nikil and Tara Viswanathan, Ian Winters, and Brooke Wright all helped to make Stanford feel like home. Dinner parties with Greg Benedict and Leah Meth were always welcome study breaks. Cedar House in Berkeley was a home-away-from-home, and Iris Corey, Hannah Marshall, and Armin Samii immediately became close friends. I especially appreciate Armin for hosting me when I was considering attending UC Berkeley for graduate school, and I want to thank Armin and Dave Jackson for lending me their spare hotel room at SIGGRAPH.

Throughout my graduate studies, I had the opportunity to meet many wonderful housemates. In particular, Heather and Zac Altman, Katie Ballenger, Caroline Beaudon, Akhila Bettadapur, Nigel Kooreman, Felipe Pincheira, Jessica Schrouff, Gustavo Schwenkler, Alex and Andreas Zoellner quickly became close friends, and all made the Hackmann House a joy to come home to. Living in the Facebook House was surreal and hilarious thanks to Kai Kuspa. I was extremely fortunate to live in Hawthorne Manor with Aladin Corhodzic, Mike D'Amico, Sherif Kassatly, and Ibrahim Toukan. I experienced so many laughs and wonderful memories with all of you. I especially appreciate Sherif and Ibrahim for lending me a couple of months of rent at one point – thank you. Sherif also did me a huge favor by lending me his old laptop when mine got stolen.

My time at Stanford was undoubtedly more fun because of all the opportunities I had to get in

my musical happy place through DJing. I am thankful for all my friends that made this possible. Armin Samii hosted the best house party of all time, and trusted me to DJ at it. Jack Wang at KZSU trusted me to DJ at Stanford football games soon after we met. Jake Zeller went out of his way to connect me with nightclubs in San Francisco, which ultimately enabled me to open for the Chainsmokers. Working with Ladidi Gharba was a once-in-a-lifetime opportunity. I miss you. Henrik Thorenfeldt trusted me to DJ at the Young Scandinavians Club cabin at Clear Lake. We quickly became close friends, and Clear Lake became a second family. Brian Mac immediately became a close friend and trusted DJ partner.

Finally, I want to thank my mom and dad, Judy and Rodger Roberts, for their kind words of support over the years. And I especially want to thank my brother, Anthony Roberts. Several times throughout my studies, you loaned me a significant amount of money, and these extraordinary acts of trust enabled me to continue pushing forward. I would not have been able to complete this dissertation without you. Thank you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Drone cameras are everywhere. In the last few years, these devices have emerged as a fundamentally new kind of consumer camera. Indeed, the US government forecasts that over 3 million consumer drones will be deployed nationally by 2019 [47]. The rapidly growing popularity of drone cameras can be attributed to their small size, low cost, and unprecedented agility. Drone cameras can maneuver through their environment in a totally freeform way, and there is an exciting and dynamic quality to drone footage that is nearly impossible to obtain otherwise.[1]

In particular, drone cameras are enabling professional filmmakers to tell stories in new ways. For example, drones are being used to film scenes in a growing number of Hollywood movies, including The Wolf of Wall Street, Transformers, and Skyfall [10, 51]. But the potential applications for drone cameras extend far beyond Hollywood. Drone cameras are being used in disaster relief scenarios [95], wildlife monitoring [39], journalism [68], 3D mapping [101], surveying [3], and inspection [6].

But drone cameras are hard to use. It is challenging for novice users to fly simple trajectories without crashing, and it is challenging for experts to obtain the kind of smooth and aesthetically pleasing footage you'd expect to see in a Hollywood movie.

Our ultimate goal in this dissertation is to make drone cameras easier to use.

1.1 Why Are Drone Cameras Hard To Use?

Today, the most common method for practitioners to control drone cameras is to pilot them manually with remote-control joysticks. Joysticks are *expressive*, in the sense that experts can use joysticks to

[1] As an example of particularly compelling drone footage, we encourage the reader to watch this shot of Venice Beach by Robert McIntosh, one of the world's best drone pilots: http://vimeo.com/218839072.

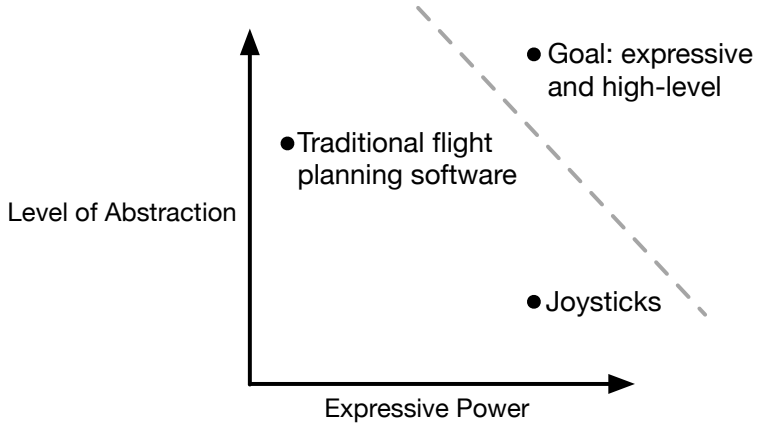


Figure 1.1: Overview of existing user interfaces for drone camera systems. On the one hand, joysticks are highly expressive but too low-level. On the other hand, traditional flight planning software is high-level but not expressive enough. Our goal in this dissertation is to build drone camera systems that are expressive and high-level.

express an incredibly wide range of drone motion. But joysticks can be counter-intuitive for novice users, because the user must translate high-level goals for the drone (e.g., "get low to the ground and go towards that building") into low-level joystick movements (e.g., "pull back on the left joystick and simultaneously sweep towards 12 o'clock on the right joystick). This task is especially demanding because joystick commands typically correspond to drone actions in the *body frame* of the drone (i.e., the coordinate system that is rigidly attached to the drone). For example, pressing up on a joystick will move the drone forward in the body frame. But high-level user goals are typically easier to express in the *world frame* (i.e., the coordinate system that is rigidly attached to the environment).

When filming with a drone, users must also control the orientation of the camera, which is typically mounted on an independently controlled joint known as a *gimbal*. It is especially challenging to control the gimbal while simultaneously maneuvering the drone. With this difficulty in mind, expert pilots often work in teams, with one pilot controlling the drone motion, and another controlling the gimbal.

As an alternative to flying with joysticks, flight planning software has been developed to control a drone by specifying GPS waypoints on a 2D map. The user is responsible for setting waypoints before a flight, and the drone subsequently flies in auto-pilot mode and attemps to follow the waypoints. This type of flight planning software is *high-level*, in the sense that it does not require the user to explicitly specify moment-to-moment navigation commands for the drone. But existing flight planning software is not expressive enough. There are many interesting maneuvers that a drone could safely execute, that are difficult or impossible to express in existing flight planning software.

Figure 1.1 summarizes the landscape of existing user interfaces for drone cameras. On the one hand, joysticks are highly expressive but too low-level. On the other hand, existing flight

planning software is high-level but not expressive enough. In order for drone cameras to become widely adopted by non-expert pilots (e.g., journalists, fire fighters, biologists, real-estate agents, indie filmmakers, etc.), we need user interfaces that are both expressive and high-level. In other words, we need interfaces that expose a wide range of drone maneuvers to the user, while also making it easy to specify high-level goals and desired behaviors.

1.2 Why Do We Need Trajectory Optimization?

It is fundamentally challenging to build user interfaces for drone cameras that are expressive and high-level. As our user interfaces move away from explicitly specifying motor torques, computing the final intended drone trajectory from the user's input becomes increasingly *ambiguous*. The underlying computational problems become increasingly *ill-posed*. If we want to build high-level interfaces that are useful for non-experts, we must somehow decide how to interpret the ambiguous input we will get from the user.

The big idea in this dissertation is to use the principles of *trajectory optimization* to recover a user's intended drone trajectory, given some high-level description of the user's preferences and constraints.

In this dissertation, we will be formulating trajectory optimization problems, where we want to somehow choose a sequence of drone actions to minimize some cost function subject to some constraints, e.g.,

$$\begin{aligned} &\text{choose drone actions } \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_T \\ &\text{to minimize a cost function } \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \\ &\text{subject to user constraints and drone constraints} \end{aligned}$$

where $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ is the sequence of drone states corresponding to the sequence of drone actions. The cost functions we choose are going to depend on the particular application domain, and will vary as we progress through the dissertation. Our cost functions are going to encode things like user preferences, as well as any prior knowledge we have about a particular application domain. Our constraints are going to encode things like strict user requirements, as well as our drone's physical limits.

Trajectory optimization problems of this form are not new, and the corresponding solution methods have been widely applied to drones [82]. But they have not been widely applied to drone cameras. In this dissertation, we will apply the principles of trajectory optimization to drone cameras in new ways. We will use domain knowledge that is specialized to a particular drone camera task, and

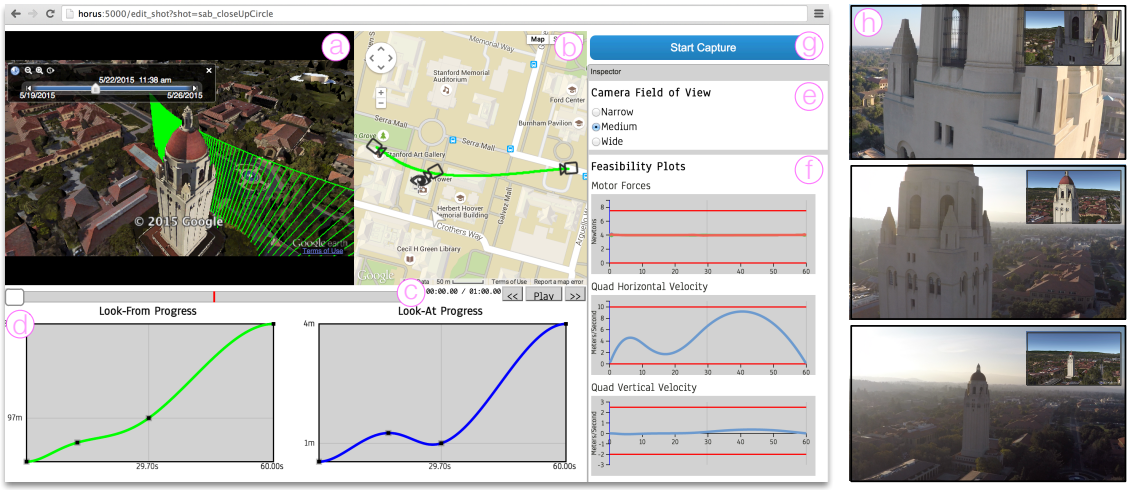


Figure 1.2: (Left) Our interactive tool for designing quadrotor camera shots. In our tool, users specify camera pose keyframes in a virtual environment using a 3D scene view (a) and a 2D map view (b). Our tool synthesizes a camera trajectory that obeys the physical equations of motion for quadrotors, 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). We provide an overview of our tool at http://youtu.be/Ds-KF4ZSnAo.

we will reformulate classical trajectory optimization problems in terms of the drone camera's visual output, i.e., in terms of what the drone is *seeing*. We will apply this methodology in two distinct application domains: drone cinematography and drone 3D scanning. In doing so, we will take several steps towards our ultimate goal of making drone cameras easier to use and more expressive.

1.3 Dissertation Overview and Summary of Contributions

In this dissertation, we make the following technical contributions.

- In Chapter 2, we introduce a software tool for designing and autonomously executing cinematography shots with quadrotor cameras (see Figure 1.2). Our tool enables users to: (1) specify shots visually using keyframes; (2) preview the resulting shots in a virtual environment;

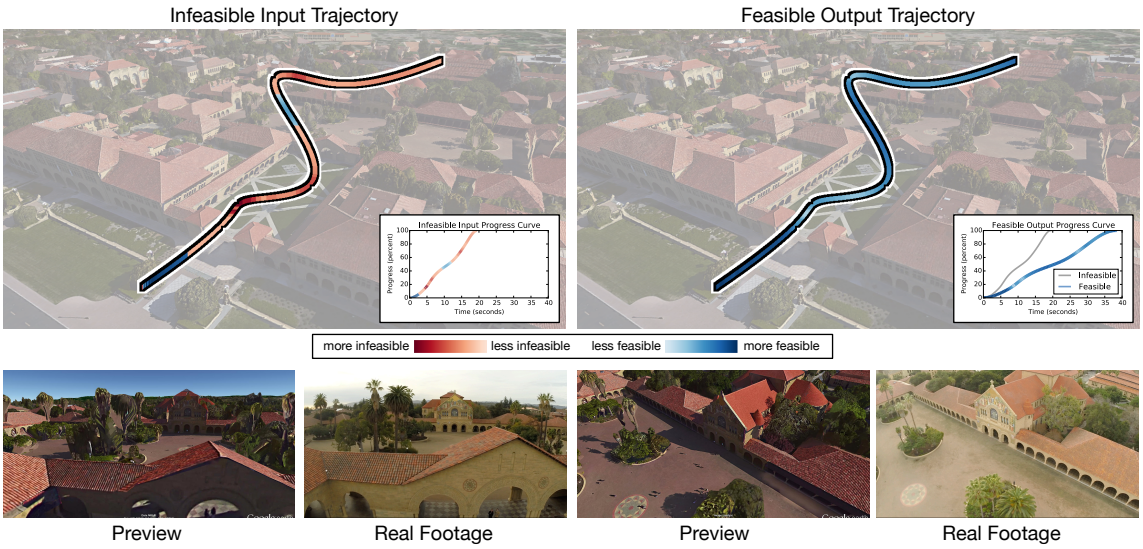


Figure 1.3: Our algorithm for generating feasible quadrotor camera trajectories. Our algorithm takes as input an infeasible quadrotor camera trajectory (top row, left) and produces as output a feasible trajectory that is as similar as possible to the input trajectory (top row, right). By design, our 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). We provide an overview of our algorithm at http://youtu.be/zhik1D2o6Fc.

(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 a commercially available quadrotor camera. 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 shots, and capture them fully autonomously.

- To support our shot planning tool, we introduce a physical model for quadrotor cameras, 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 also derive an algorithm to compute

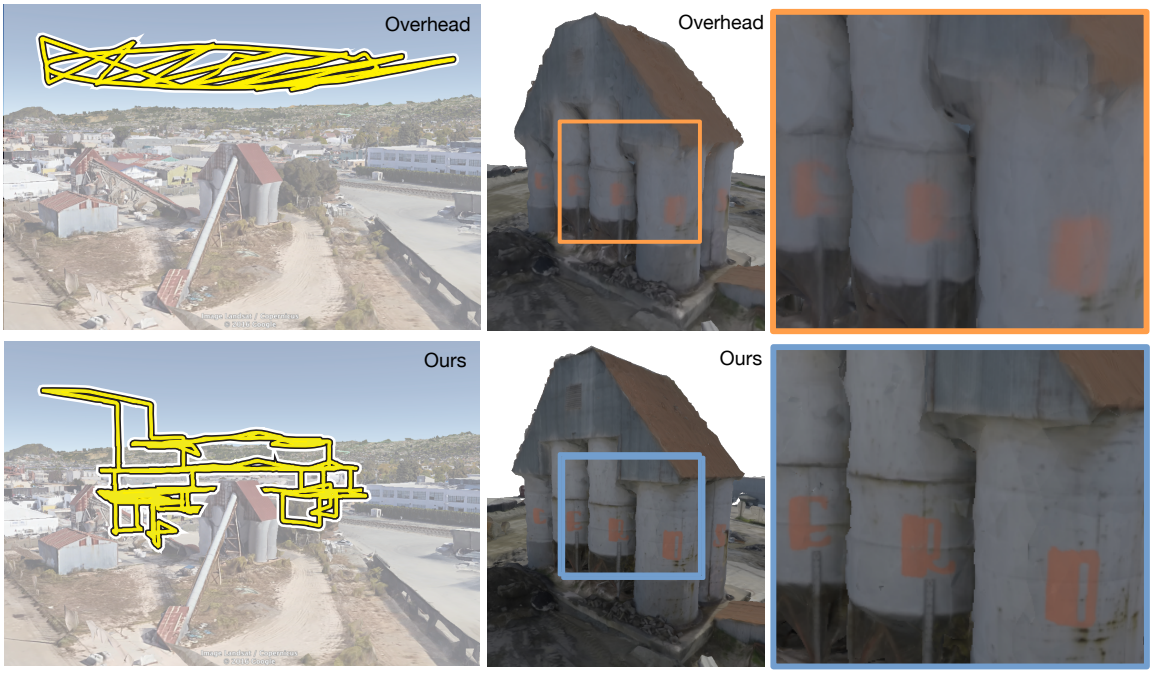


Figure 1.4: 3D reconstruction results obtained using our algorithm for generating aerial 3D scanning trajectories, as compared to an overhead trajectory. Left column: Google Earth visualizations of the trajectories. Middle and right columns: results obtained by flying a drone along each trajectory, capturing images, and feeding the images to multi-view stereo software. Our trajectories lead to noticeably more detailed 3D reconstructions than overhead trajectories. In all our experiments, we control for the flight time, battery consumption, number of images, and quality settings used in the 3D reconstruction. We provide an overview of our algorithm at http://youtu.be/89fFmfVZSO8. This figure appears in [106] and is reprinted with permission. © 2017 IEEE

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.

- In Chapter 3, to further support our shot planning tool, we introduce a fast and user-friendly algorithm for generating camera trajectories that respect the dynamics and physical limits of quadrotor hardware (see Figure 1.3). We refer to such trajectories as being *feasible*. Our algorithm takes as input an infeasible camera 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

An Interactive Tool for Designing Quadrotor Camera Shots (Chapter 2)	
Project page	http://stanford-gfx.github.io/Horus
Overview video	http://youtu.be/Ds-KF4ZSnAo
Source code	http://github.com/stanford-gfx/Horus
	http://mikeroberts3000.github.io/flashlight
Generating Dynamically Feasible Trajectories for Quadrotor Cameras (Chapter 3)	
Project page	http://graphics.stanford.edu/papers/feasible_trajectories
Overview video	http://youtu.be/zhik1D2o6Fc
Source code	http://mikeroberts3000.github.io/flashlight
Submodular Trajectory Optimization for Aerial 3D Scanning (Chapter 4)	
Project page	http://graphics.stanford.edu/papers/aerial_scanning
Overview video	http://youtu.be/89fFmfVZSO8
Source code	https://github.com/stanford-gfx/scanplan

Table 1.1: Project pages, overview videos, and source code for each dissertation chapter.

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 demonstrate that our algorithm is between 25× and 45× faster than a general-purpose trajectory optimization approach implemented using a commercially available solver. As we scale to more finely discretized trajectories, this performance gap widens, with our algorithm outperforming general-purpose trajectory optimization by between 90× and 180×. We successfully capture real video footage using the trajectories generated by our algorithm. We show that the resulting videos are faithful to virtual shot previews, even when the trajectories being executed are at our quadrotor's physical limits.

- In Chapter 4, we introduce an algorithm to generate drone camera trajectories, such that the imagery acquired during the flight will later produce a high-fidelity 3D model (see Figure 1.4). Our method uses a coarse estimate of the scene geometry to plan camera trajectories that: (1) cover the scene as thoroughly as possible; (2) observe the scene geometry from a diverse set of viewing angles; (3) avoid obstacles; and (4) respect a user-specified flight time budget. Our method relies on a mathematical model of scene coverage that exhibits an intuitive diminishing returns property known as *submodularity*. We leverage this property extensively to design a trajectory planning algorithm that reasons globally about the non-additive coverage reward obtained across a trajectory, jointly with the cost of traveling between views. We evaluate our method by using it to scan three large outdoor scenes, and we perform a quantitative evaluation using a photorealistic video game simulator.[2]

- In Chapter 5, we summarize our findings and discuss future research directions inspired by the preceding chapters.

Together, these contributions form a comprehensive toolbox of trajectory optimization methods for

[2]This passage appears in [106] and is reprinted with permission. © 2017 IEEE

drone cameras. In Table 1.1, we provide links to the project page, overview video, and open-source implementation for each chapter. The overview videos are intended to be viewed before diving into each chapter. This work appears in a series of previously published papers [72, 106, 107]. The intended scope of the mathematical notation we introduce is local to each chapter, and related work is discussed within each chapter.

Chapter 2

An Interactive Tool for Designing Quadrotor Camera Shots

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 simultaneously 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 chapter, 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 1.2) 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 1.2. 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.

2.1 Related Work

Designing Trajectories for Physical Cameras The DJI Ground Station [38] and the APM Mission Planner [7] systems allow users to design quadrotor camera trajectories by placing waypoints on a 2D map. The QGroundControl system [93] 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 [2] and DJI Go [37] 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 [2], or in a circular orbit around

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

a point of interest [37]. 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. [25]. We discuss directly related work not included in this survey here. Oskam et al. [98] and Hsu et al. [67] 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 [98], or by solving a continuous optimization problem [67]. Existing methods for synthesizing virtual camera trajectories guarantee C^1 or C^2 continuity. However, we demonstrate in Section 2.7 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 [94] and Richter et al. [105]. 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 [75, 86, 108, 143]. 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 [125]. 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 quadrotor, 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. [120] 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.

2.2 Design Principles

In order to design more effective tools for quadrotor camera control, we began by analyzing manuals on cinematography [9, 73, 88], 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 [9, 83]. 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 [88]. 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.

2.3 User Interface

We reify the design principles described in Section 2.2 into an interactive tool for planning and capturing quadrotor camera shots (see Figure 1.2). 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 1.2a). 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 1.2b). 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 1.2c). 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 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 1.2d). 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 1.2f). 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 1.2a), the 2D map view (Figure 1.2b), and on the easing curves (Figure 1.2d). 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.

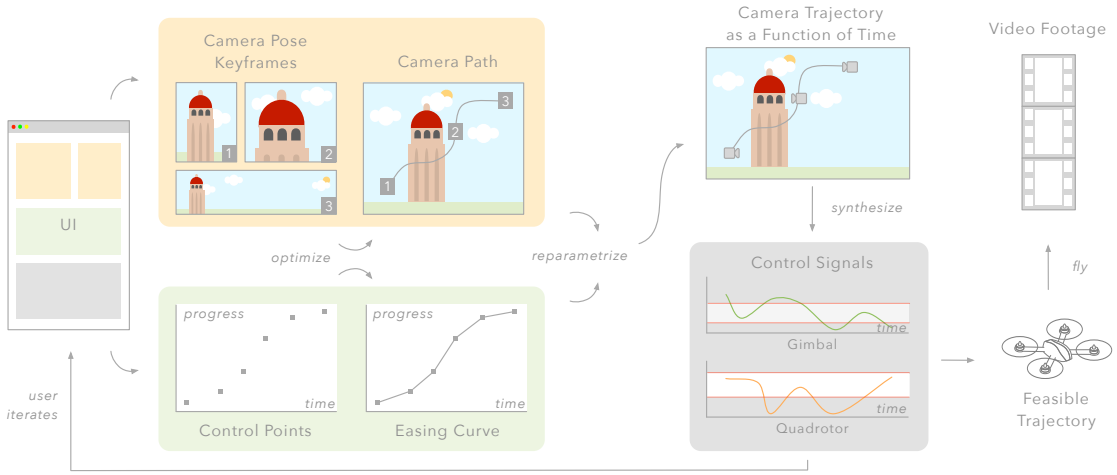


Figure 2.1: Overview of the major technical components of our system. We begin with two user-specified inputs: (1) camera pose keyframes in a virtual environment (e.g., GOOGLE EARTH); and (2) a sequence of easing curve control points. From these inputs, we compute a smooth camera path and a smooth easing 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.

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 1.2g). Once the user clicks this button, our tool commands a quadrotor camera to execute the user-specified shot fully autonomously, capturing real video footage.

2.4 Technical Overview

We provide an overview of the major technical components of our system in Figure 2.1. 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 2.5). 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 2.6). 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 2.7). 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 discussed in Section 2.3, 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 2.8). At a high level, we use the camera trajectory computed in Section 2.6 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 2.6, without going to the extra trouble of computing control signals in Section 2.7. 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.

In Chapter 3, we will *computationally* generate trajectories that adhere to the physical limits of a quadrotor. But for now, we rely on the user to adjust her shot if it violates these limits.

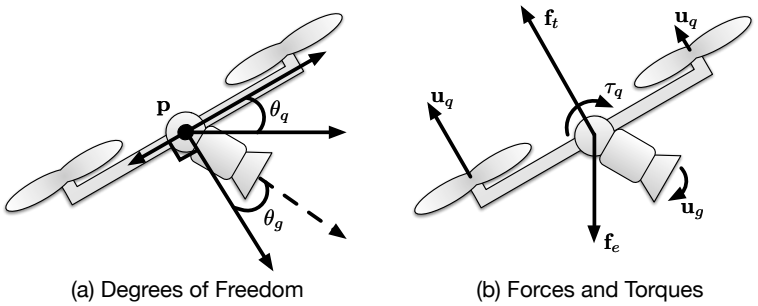


Figure 2.2: Overview of our quadrotor camera model, shown in 2D for simplicity. (a) Degrees of freedom. We model the physical state of a quadrotor camera with the following degrees of freedom: the position of the quadrotor in the world frame, $\mathbf{p}$; the orientation of the quadrotor in the world 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.

2.5 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 2.2.

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,

$$\begin{aligned} \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) &= \mathbf{B}(\mathbf{q})\mathbf{u} \\ \text{subject to} \quad \mathbf{u}^{\min} &\leq \mathbf{u} \leq \mathbf{u}^{\max} \\ \mathbf{q}^{\min} &\leq \mathbf{q} \leq \mathbf{q}^{\max} \\ \dot{\mathbf{q}}^{\min} &\leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}^{\max} \end{aligned} \tag{2.1}$$

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 [124] 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 [94] to include a fully actuated 3 degree-of-freedom gimbal. We include a concise definition for these matrices in Section 2.10.1, and a more detailed derivation in Section 2.10.2.

2.6 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 2.7.

We follow the standard practice in computer graphics [99] 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 Section 2.10.3, we evaluate alternative polynomial representations for quadrotor camera paths. In our experience, we found that 7^{th} 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 7^{th} 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 7^{th} 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 [94].

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,

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

$$\begin{aligned} \text{subject to} \quad & p_i(0) = v_i \qquad\qquad p_i(1) = v_{i+1} \\ & \frac{d^j}{d\bar{u}_i^j} p_i(0) = w_i^j \mathbf{d}_{i,j} \quad \frac{d^j}{d\bar{u}_i^j} p_i(1) = w_i^j \mathbf{d}_{i+1,j} \end{aligned} \tag{2.2}$$

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 Section 2.10.3, we evaluate alternative objective functions. In our experience, we found that minimizing the 4^{th} derivative of our polynomials produced the smoothest and most reasonably bounded control signals for quadrotors. For this reason, we choose to minimize the 4^{th} derivative of our polynomials.

We can express the optimization problem in equation (2.2) as a constrained quadratic minimization problem as follows,

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{x}^T \mathbf{Q} \mathbf{x} \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.3}$$

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 (2.2); $\mathbf{A}$ is the matrix and $\mathbf{b}$ is the vector that can be obtained by expressing the equality constraints from equation (2.2) in matrix form. The problem in equation (2.3) can be solved by solving the following linear system,

$$\begin{bmatrix} 2\mathbf{Q} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix} \tag{2.4}$$

where λ is the Lagrange multiplier variable obtained by transforming equation (2.3) into unconstrained form [16].

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* [144] helped to produce well-behaved smooth camera paths. To that end, we also constrained the 1^{st} derivatives at the endpoints of our camera path as we would for Natural Cubic Splines [11].

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 [57].

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.

2.7 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*

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 (2.1). Or stated more precisely, that the state space trajectory we compute in Section 2.7, when substituted into equation (2.1), always yields a left hand side that is in the column space of the matrix $\mathbf{B}$. In our algorithm, the quadrotor's local $\mathbf{y}$ (up) axis, in combination with the virtual camera's local $\mathbf{x}$ (forward) axis, uniquely determines the orientation of the quadrotor (lines 4–6).

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 (2.1) 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 (2.1). Or stated more precisely, that the state space trajectory we compute in this section, when substituted into equation (2.1), 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 [94]. 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 [36]. 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 1^{st} and 2^{nd} 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 (2.1), 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 (2.1), 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 (2.1) 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 Section 2.10.4.

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 2^{nd} 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 (2.1) that we use the 2^{nd} 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 4^{th} 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 [94].

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 (2.1). 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 (2.1). 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 2.3.

2.8 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 2.3. We build our real-time control system on top of the open source ArduPilot autopilot software [7]. The ArduPilot software runs on board the quadrotor, and provides a hierarchical feedback controller for following camera trajectories, similar to the controller described by Kumar and Michael [82]. 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.

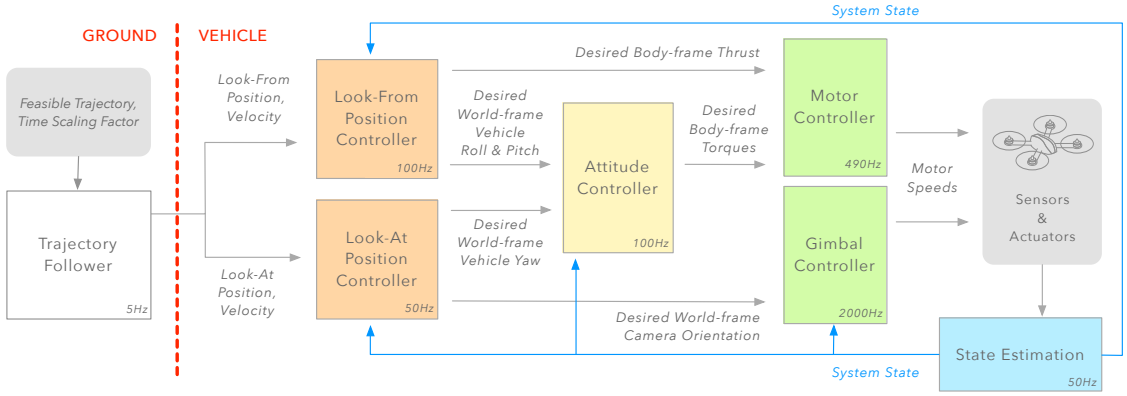


Figure 2.3: Block diagram of our real-time control system for executing camera trajectories. On a ground station (left), our trajectory follower (white) samples the camera trajectory, transmitting the sampled position and velocity of look-at and look-from points to the quadrotor. Our trajectory follower 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 [82].

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 [1] running the open source ArduPilot autopilot software [7] on a Pixhawk autopilot computer [93]. 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

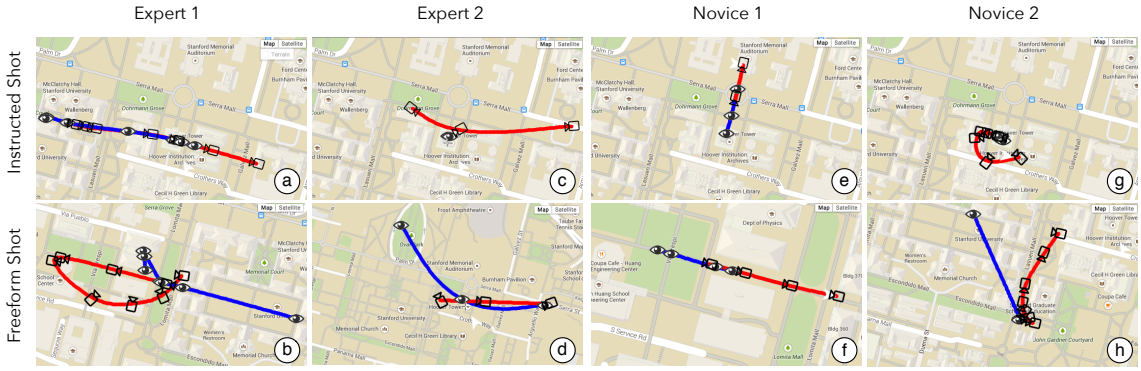


Figure 2.4: Camera shots created by the two experts (left) and two novices (right) in our user study. The look-from and look-at trajectories for each shot are shown in red and blue respectively. The shots created by our participants contain a wide variety of camera motions.

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 [7].

2.9 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 2.5: 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 2.4, 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 2.5). 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) 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.

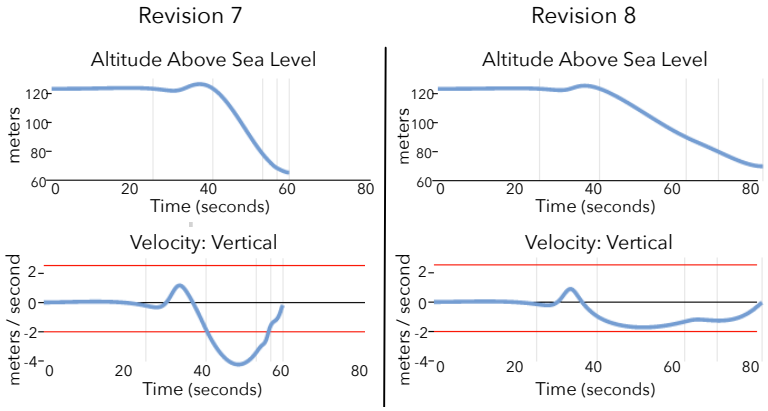


Figure 2.6: Participants were able to modify infeasible shots into feasible shots using the visual feedback we provide in our tool. After his 7^{th} revision, Expert 1 found that his shot was infeasible (left). He edited both the altitude and timing of 3 keyframes to create a feasible shot as his 8^{th} revision (right). Horizontal red lines indicate physical limits of our quadrotor hardware.

- 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 2.5. 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 1.2 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.

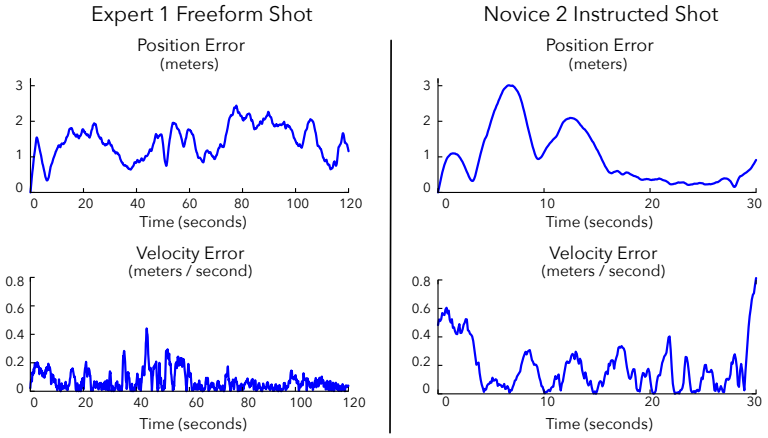


Figure 2.7: Position and velocity error of our quadrotor for the longest (left) and shortest (right) shots. The position error is less than 3.01m at all times, and the velocity error is less than 0.80m/s at all times. Note that the horizontal scaling varies varies on the left and right subplots.

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 2.6 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 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.

In Chapter 3, we will present an automatic algorithm for correcting feasibility violations.

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 2.7). 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."*).

2.10 Appendix

2.10.1 Defining the Quadrotor Camera Manipulator Matrices

In this subsection, we define the quadrotor camera manipulator matrices, attempting to be as concise as possible. We include a detailed derivation of these matrices in Section 2.10.2.

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

$$\mathbf{q} = \begin{bmatrix} \mathbf{p} \\ \mathbf{e}_q \\ \mathbf{e}_g \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} \mathbf{u}_q \\ \mathbf{u}_g \end{bmatrix} \tag{2.5}$$

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,

$$\begin{aligned}
\mathbf{H}(\mathbf{q}) &= \begin{bmatrix} m\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{A}_g \end{bmatrix} \\
\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) &= \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\dot{\mathbf{A}}_q - \left(\mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\dot{\mathbf{e}}_q\right)_\times \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \dot{\mathbf{A}}_g \end{bmatrix} \\
\mathbf{G}(\mathbf{q}) &= \begin{bmatrix} -\mathbf{f}_e \\ \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times1} \end{bmatrix} \\
\mathbf{B}(\mathbf{q}) &= \begin{bmatrix} \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f} & \mathbf{0}_{3\times3} \\ \mathbf{M}_\tau & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times4} & \mathbf{I}_{3\times3} \end{bmatrix}
\end{aligned} \tag{2.6}$$

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,

$$\begin{aligned}
\mathbf{M}_\mathbf{f} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
\mathbf{M}_\tau &= \begin{bmatrix} ds_\alpha & ds_\beta & -ds_\beta & -ds_\alpha \\ \gamma & -\gamma & \gamma & -\gamma \\ -dc_\alpha & dc_\beta & dc_\beta & -dc_\alpha \end{bmatrix}
\end{aligned} \tag{2.7}$$

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 Section 2.10.2 for details.

2.10.2 Deriving the Quadrotor Camera Manipulator Matrices

In this subsection, we derive the manipulator matrices for our quadrotor camera model. At a high level, our strategy will be to relate our quadrotor camera's degrees of freedom to our control inputs.

In the derivation that follows, we assume that our quadrotor and gimbal are *kinematically coupled* [78], 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* [78], in the sense that torques acting on the gimbal do not induce reactive torques on the quadrotor. These assumptions simplify the modeling of our system. Moreover, we feel they are justified, since in the cameras mounted on quadrotors tend to be very lightweight relative to the quadrotors themselves. For example, on our hardware platform, our camera and gimbal are roughly 25× lighter than our quadrotor.

Relating the Quadrotor's Position to the Control Inputs We assume that there are two forces acting on our quadrotor camera: (1) a *thrust force* induced by the quadrotor's propellers; and (2) an *external force* that models any other forces acting on the quadrotor, such as gravity, wind, and drag. Based on this assumption, we use Newton's Second Law to relate the linear acceleration of the quadrotor in the world frame, $\ddot{\mathbf{p}}$, to the control input applied at each of the quadrotor's propellers, $\mathbf{u}_q$, as follows,

$$m\ddot{\mathbf{p}} = \mathbf{f}_e + \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_{\mathbf{f}}\mathbf{u}_q \tag{2.8}$$

where m is the mass of the quadrotor camera; $\mathbf{f}_e$ 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; and $\mathbf{R}_{\mathcal{W},\mathcal{Q}}$ 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). We define $\mathbf{M_f}$ as follows,

$$\mathbf{M_f} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.9}$$

Relating the Quadrotor's Angular Acceleration and Angular Velocity to the Control Inputs We use Euler's Second Law to relate the angular acceleration and angular velocity of the quadrotor in the body frame, $\dot{\omega}_q$ and ω_q respectively, to the control input applied at each of the quadrotor's propellers, $\mathbf{u}_q$, as follows,

$$\mathbf{I}_q\dot{\omega}_q + \omega_q \times \mathbf{I}_q\omega_q = \mathbf{M}_\tau\mathbf{u}_q \tag{2.10}$$

where $\mathbf{I}_q$ is the inertia matrix of the quadrotor camera; and $\mathbf{M}_\tau$ 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. We define $\mathbf{M}_\tau$ as follows,

$$\mathbf{M}_\tau = \begin{bmatrix} ds_\alpha & ds_\beta & -ds_\beta & -ds_\alpha \\ \gamma & -\gamma & \gamma & -\gamma \\ -dc_\alpha & dc_\beta & dc_\beta & -dc_\alpha \end{bmatrix} \tag{2.11}$$

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 it's 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 $s_a = \sin a$. Our definition for $\mathbf{M}_\tau$ assumes: (1) the quadrotor's propellers are co-planar with its center of mass; and (2) a linear relationship between the magnitude of the in-plane torque generated by the quadrotor propeller, and the magnitude of the upward thrust force generated by the propeller.

Relating the Gimbal's Angular Acceleration to the Control Inputs We assume that our 3 degree-of-freedom gimbal is fully actuated, and has very large actuator limits. Since fully actuated systems are *feedback equivalent* [124] to double integrator systems, and we assume that our gimbal has very large actuator limits, we model the gimbal as a double integrator for simplicity. We relate the angular acceleration of the gimbal in the body frame of the quadrotor, $\dot{\omega}_g$ to the *feedback linearized* [124] control input applied at the gimbal, $\mathbf{u}_g$, as follows,

$$\dot{\omega}_g = \mathbf{u}_g \tag{2.12}$$

Relating Euler Angle Time Derivatives to Angular Velocity and Angular Acceleration At this point, we have related the angular velocities and accelerations of our system to our control inputs. However, to derive the complete equations of motion for our system, we need to relate the Euler angle time derivatives of our system to our control inputs. To do this, we must relate Euler angle time derivatives to angular velocities and angular accelerations.

Let $\mathbf{e} = [\theta\ \psi\ \phi]^T$ be a vector of our Euler angles. Let us define the rotation matrix $\mathbf{R}$ in terms of the Euler angles θ, ψ, and ϕ as follows,

$$\begin{aligned}\mathbf{R} &= \mathbf{R}_{\mathbf{y}}^{\psi}\mathbf{R}_{\mathbf{z}}^{\theta}\mathbf{R}_{\mathbf{x}}^{\phi} \\ &= \begin{bmatrix} c_\psi & 0 & s_\psi \\ 0 & 1 & 0 \\ -s_\psi & 0 & c_\psi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\theta & -s_\theta \\ 0 & s_\theta & c_\theta \end{bmatrix} \begin{bmatrix} c_\phi & -s_\phi & 0 \\ s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}\end{aligned} \tag{2.13}$$

where $c_a = \cos a$ and $s_a = \sin a$.

We can straightforwardly compute the time derivative of this expression to get an (admittedly unpleasant) expression for $\dot{\mathbf{R}}$ in terms of our Euler angles and their time derivatives. We omit this step for brevity.

We make the observation that $\dot{\mathbf{R}} = (\omega)_\times \mathbf{R}$, where ω is the angular velocity of a rotating body in the non-rotating frame; 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 $\mathbf{b}$.

From the above expression, we immediately get $\dot{\mathbf{R}}\mathbf{R}^T = (\omega)_\times$. We observe that the entries of $\dot{\mathbf{R}}\mathbf{R}^T$ are linear in $\dot{\psi}$, $\dot{\theta}$, and $\dot{\phi}$. Therefore, there is a matrix $\mathbf{A}$ that relates $\dot{\mathbf{e}}$ to ω as follows,

$$\mathbf{A}\dot{\mathbf{e}} = \omega \tag{2.14}$$

We can take the time derivative of both sides of this expression using the product rule to get the following expression,

$$\dot{\mathbf{A}}\dot{\mathbf{e}} + \mathbf{A}\ddot{\mathbf{e}} = \dot{\omega} \tag{2.15}$$

We define the matrix $\mathbf{A}$ that relates $\dot{\mathbf{e}}$ to ω according to the linear relationship $\mathbf{A}\dot{\mathbf{e}} = \omega$, and its time derivative $\dot{\mathbf{A}}$, as follows,

$$\begin{aligned}\mathbf{A} &= \begin{bmatrix} c_\psi & 0 & s_\psi c_\theta \\ 0 & 1 & -s_\theta \\ -s_\psi & 0 & c_\psi c_\theta \end{bmatrix} \\ \dot{\mathbf{A}} &= \begin{bmatrix} -s_\psi\dot{\psi} & 0 & -s_\psi s_\theta\dot{\theta} + c_\psi c_\theta\dot{\psi} \\ 0 & 0 & -c_\theta\dot{\theta} \\ -c_\psi\dot{\psi} & 0 & -s_\psi c_\theta\dot{\psi} + s_\theta c_\psi\dot{\theta} \end{bmatrix}\end{aligned} \tag{2.16}$$

where $c_a = \cos a$ and $s_a = \sin a$.

Relating the Quadrotor's Orientation to the Control Inputs We can rotate equations (2.14) and (2.15) into the body frame of the quadrotor, and substitute them into equation (2.10), to get the following expression for the quadrotor's rotational dynamics,

$$\mathbf{I}_q(\mathbf{R}_{\mathcal{Q},\mathcal{W}}\dot{\mathbf{A}}_q\dot{\mathbf{e}}_q + \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\ddot{\mathbf{e}}_q) + \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\dot{\mathbf{e}}_q \times \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\dot{\mathbf{e}}_q = \mathbf{M}_\tau\mathbf{u}_q \tag{2.17}$$

where $\mathbf{e}_q$ is the vector of Euler angles representing the quadrotor's orientation in 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; and $\mathbf{A}_q$ is the matrix that relates the quadrotor's Euler angle time derivatives to its angular velocity in the world frame.

Relating the Gimbal's Orientation to the Control Inputs Similarly to our approach in the previous subsection, we can substitute equation (2.15) into equation (2.12) to get the following expression for the gimbal's rotational dynamics,

$$\dot{\mathbf{A}}_g\dot{\mathbf{e}}_g + \mathbf{A}_g\ddot{\mathbf{e}}_g = \mathbf{u}_g \tag{2.18}$$

where $\mathbf{e}_g$ is the vector of Euler angles representing the orientation of the gimbal in the body frame of the quadrotor; and $\mathbf{A}_g$ is the matrix that relates the gimbal's Euler angle time derivatives to its angular velocity in the body frame of the quadrotor.

Defining the Manipulator Matrices We define the layout of our degree-of-freedom vector $\mathbf{q}$, and our control vector $\mathbf{u}$, as follows,

$$\mathbf{q} = \begin{bmatrix} \mathbf{p} \\ \mathbf{e}_q \\ \mathbf{e}_g \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} \mathbf{u}_q \\ \mathbf{u}_g \end{bmatrix} \tag{2.19}$$

Based on this layout, we can express equations (2.8), (2.17), and (2.18) in manipulator form. In doing so, we get the following expressions for our quadrotor camera manipulator matrices,

$$\begin{aligned}
\mathbf{H}(\mathbf{q}) &= \begin{bmatrix} m\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{A}_g \end{bmatrix} \\
\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) &= \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\dot{\mathbf{A}}_q - (\mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\dot{\mathbf{e}}_q)_\times \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \dot{\mathbf{A}}_g \end{bmatrix} \\
\mathbf{G}(\mathbf{q}) &= \begin{bmatrix} -\mathbf{f}_e \\ \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times1} \end{bmatrix} \\
\mathbf{B}(\mathbf{q}) &= \begin{bmatrix} \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f} & \mathbf{0}_{3\times3} \\ \mathbf{M}_\tau & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times4} & \mathbf{I}_{3\times3} \end{bmatrix}
\end{aligned} \tag{2.20}$$

where $\mathbf{0}_{p\times q}$ is the $p \times q$ zero matrix; and $\mathbf{I}_{k\times k}$ is the $k \times k$ identity matrix. In our definition for the manipulator matrices above, we assume that $\mathbf{f}_e$ can depend on $\mathbf{q}$, but cannot depend on $\dot{\mathbf{q}}$. We make this assumption for simplicity, although it could be relaxed by making minor modifications to $\mathbf{C}$ and $\mathbf{G}$ above.

2.10.3 Empirical Justification for Minimizing the 4th Derivative of 7th Degree Polynomials

In this subsection, we evaluate alternative polynomial representations for quadrotor camera paths. We find that minimizing the 4th derivative of 7th degree piecewise polynomials produces the smoothest and most reasonably bounded control signals for quadrotors (see Figures 2.8 and 2.9 in this document).

When evaluating alternative polynomial representations, we chose to limit our experiments to odd degree (i.e., even order) polynomials, since these tend to produce smoother curves than even degree (i.e., odd order) polynomials [56].

2.10.4 Proof that B is Always Full Column Rank

To prove that the matrix $\mathbf{B}$ is full column rank, consider the following matrix,

$$\mathbf{M} = \begin{bmatrix} \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f} \\ \mathbf{M}_\tau \end{bmatrix} \tag{2.21}$$

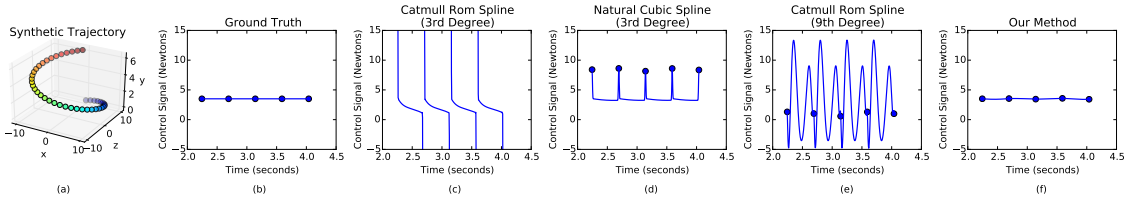


Figure 2.8: Comparison of the quadrotor control signals resulting from different keyframe interpolation methods. We construct a simple synthetic traejectory (a). We use the control signals required for a quadrotor to follow the synthetic trajectory as ground truth (b). We sample the synthetic trajectory with 15 evenly spaced keyframes, and we construct an interpolated trajectory from these keyframes using different interpolation methods. We plot the control signals produced by each interpolation method. We require that the control signals be continuous, and we prefer control signals that are as close as possible to the ground truth. We show the middle of these control signal plots to highlight their periodic behavior without boundary artifacts. For each interpolation method, we show the control signal for the front right propeller, and we indicate the control signal value at each keyframe with a blue dot. 3^{rd} degree Catmull Rom Splines are C^1 continuous, and therefore produce discontinuous control signals (c). Natural Cubic Splines are C^2 continuous, and therefore also produce discontinuous control signals (d). 9^{th} degree Catmull Rom Splines are C^4 continuous, so they produce continuous control signals, but with large periodic excursions (e). Our method is C^4 continuous and produces control signals with minimal excursions (f).

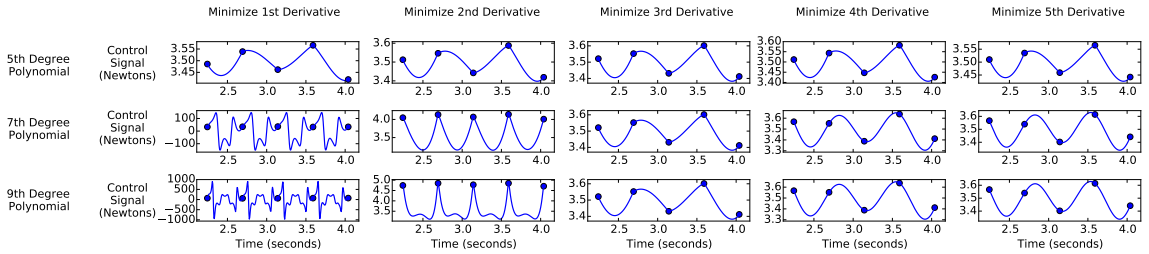


Figure 2.9: Comparison of the quadrotor control signals resulting from different variations of our method when interpolating the synthetic trajectory described in Figure 2.8. We show the middle of these control signal plots to highlight their periodic behavior without boundary artifacts. For each variation of our method, we show the control signal for the front right propeller, and we indicate the control signal value at each keyframe with a blue dot. Note that the vertical scaling varies in each subplot, and is different from the vertical scaling in Figure 2.8. Minimizing the 4^{th} derivative of 7^{th} degree polynomials (also shown, with different vertical scaling, in Figure 2.8f) is the simplest variation of our method that produces smooth and reasonably bounded control signals. Note that C^4 continuous position trajectories are only guaranteed to produce C^0 continuous control signals for quadrotors, which is why some of the control signals in this figure are jagged.

It will suffice to show that $\text{rank}(\mathbf{M}) = 4$. We assume without loss of generality that $0 < \alpha, \beta < \frac{\pi}{2}$ and that $d, \gamma > 0$. We can clearly see that $\text{rank}(\mathbf{M}_\tau) = 3$.

Suppose for the sake of contradiction that $\text{rank}(\mathbf{M}) = 3$. Then we must be able to represent any

row of $\mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M_f}$ as a linear combination of the rows of $\mathbf{M}_\tau$. Let the row $\mathbf{r} = [r\ r\ r\ r]$ be one such row of $\mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M_f}$. Representing $\mathbf{r}$ as a linear combination of the rows in $\mathbf{M}_\tau$, we get the following system of equations,

$$\begin{aligned} r = \ & ids_\alpha & + j\gamma & - kdc_\alpha \\ r = \ & ids_\beta & - j\gamma & + kdc_\beta \\ r = \ & -ids_\beta & + j\gamma & + kdc_\beta \\ r = \ & -ids_\alpha & - j\gamma & - kdc_\alpha \end{aligned} \tag{2.22}$$

for some i, j, k. Equating the first and last of these equations above, we get $ids_\alpha = -j\gamma$. Equating the middle two of these equations above, we get $ids_\beta = j\gamma$. Substituting these two new expressions into the first two equations above, we get $c_\alpha = -c_\beta$. However, this is only possible if α or β is outside the range $(0, \frac{\pi}{2})$. Since we had previously assumed that α and β are both inside the range $(0, \frac{\pi}{2})$, we have arrived at a contradiction. This completes our proof.

Chapter 3

Generating Dynamically Feasible Trajectories for Quadrotor Cameras

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 chapter, 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 1.3.

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, the tool for designing quadrotor camera shots introduced in Chapter 2, 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.

3.1 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 [7, 38] or in a 3D virtual environment [52, 72, 93]. Other tools allow users to interactively control the speed and orientation of a quadrotor camera as it flies along a pre-determined trajectory [2, 37, 72]. 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.

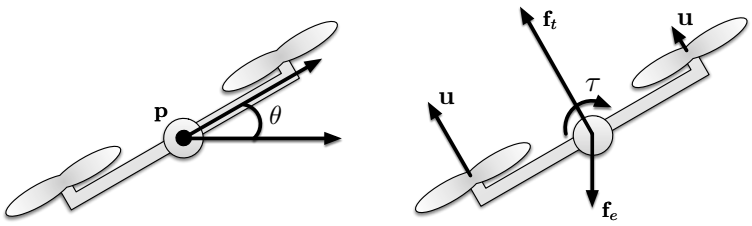
The shot planning tool introduced in Chapter 2 computes velocities and control forces along a user-specified trajectory, notifying the user if her shot is infeasible. However, this tool offers no guidance on how to modify the shot to make it feasible. The tool introduced by Gebhardt et al. [52] optimizes 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 [18, 33, 72, 94]. These spline-based trajectories can be adapted to remain within velocity and control force limits by optimizing the time allocated to each spline segment [18, 94]. 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 chapter, some of these methods explicitly optimize a progress curve [15, 30, 128]. 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 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 [53], and in the robotics literature [12]. 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 [136], and *direct collocation* in the robotics community [12]. We refer the reader to Fang and Pollard [45] and Safonova et al. [109] for a detailed overview of spacetime constraints approaches in computer graphics. A well-known limitation of spacetime



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

Figure 3.1: 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. This model does not include a camera gimbal, and is therefore lower-dimensional than the model in the previous chapter. A lower-dimensional model is sufficient for our purposes in this chapter, because we guarantee that any drone with a two-axis gimbal can execute the camera trajectories produced by our algorithm.

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 [92], and even more extensively in the robotics literature [31, 85, 114, 118, 130]. 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. [46] 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.

3.2 Quadrotor Dynamics Model

In this section, we introduce the quadrotor dynamics model we use throughout this chapter. Although this model is similar to the one in Chapter 2 and follows previous literature [72, 94], we present it here for completeness, and in a unified notation with the rest of the chapter. We provide

an overview of our model in Figure 3.1.

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* [78], 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* [78], 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 introduced in Chapter 2. 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× 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,

$$\begin{aligned} &\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{B}(\mathbf{q})\mathbf{u} \\ \text{subject to} \quad &\dot{\mathbf{q}}^{\min} \leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}^{\max} \\ &\mathbf{u}^{\min} \leq \mathbf{u} \leq \mathbf{u}^{\max} \end{aligned} \tag{3.1}$$

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 [94] in matrix form (see Section 2.10.2 for a more detailed derivation). In Section 3.7.1, we include a concise definition for these matrices, which are known as

the *manipulator matrices* [124].

3.3 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 Chapter 2, 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 in Chapter 2. 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 3.4 and 3.5.

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 (3.1) 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 2. 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,

$$\mathbf{q} = \mathbf{Q}(\mathbf{c}, \dot{\mathbf{c}}, \ddot{\mathbf{c}}) \tag{3.2}$$

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 2, 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 (3.2) twice with respect to time as follows,

$$\begin{aligned} \dot{\mathbf{q}} &= \frac{d\mathbf{Q}}{dt} &&= \dot{\mathbf{Q}}(\mathbf{c}, \dot{\mathbf{c}}, \ddot{\mathbf{c}}, \dddot{\mathbf{c}}) \\ \ddot{\mathbf{q}} &= \frac{d^2\mathbf{Q}}{dt^2} &&= \ddot{\mathbf{Q}}(\mathbf{c}, \dot{\mathbf{c}}, \ddot{\mathbf{c}}, \dddot{\mathbf{c}}, \ddddot{\mathbf{c}}) \end{aligned} \tag{3.3}$$

Finally, we solve for the control forces by substituting our expressions for $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ into equation

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 2: 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 (3.1).

(3.1) and solving for $\mathbf{u}$ as follows,

$$\mathbf{u} = \mathbf{B}^{\star}(\mathbf{q})\left[\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q})\right] \tag{3.4}$$

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 (3.1) is always in the column space of $\mathbf{B}$, and $\mathbf{B}$ is always full column rank [72]. Since we have closed form expressions for $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$, equation (3.4) gives us a closed form expression for $\mathbf{u}$. We use the notation $\mathbf{U}$ to refer to this closed form expression as follows,

$$\mathbf{u} = \mathbf{U}(\mathbf{c}, \dot{\mathbf{c}}, \ddot{\mathbf{c}}, \dddot{\mathbf{c}}, \ddddot{\mathbf{c}}) \tag{3.5}$$

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.

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

In this section, we extend our analysis in Section 3.3 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 3.3, 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,

$$\mathbf{c} = \mathbf{c}(s(t)) \tag{3.6}$$

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,

$$\begin{aligned}
\dot{\mathbf{c}} &= \mathbf{c}'\dot{s} \\
\ddot{\mathbf{c}} &= \mathbf{c}''\dot{s}^2 + \mathbf{c}'\ddot{s} \\
\dddot{\mathbf{c}} &= \mathbf{c}'''\dot{s}^3 + 3\mathbf{c}''\dot{s}\ \ddot{s} + \mathbf{c}'\dddot{s} \\
\ddddot{\mathbf{c}} &= \mathbf{c}''''\dot{s}^4 + 6\mathbf{c}'''\dot{s}^2\ \ddot{s} + 3\mathbf{c}''\ddot{s}^2 + 4\mathbf{c}''\dot{s}\ \dddot{s} + \mathbf{c}'\ddddot{s}
\end{aligned} \tag{3.7}$$

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 (3.7) into our expressions for $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, and $\mathbf{u}$ (i.e., into equations (3.2), (3.3), and (3.5)). We find that we can express the full state of the quadrotor, and all necessary control 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 (3.1). 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 3.5.

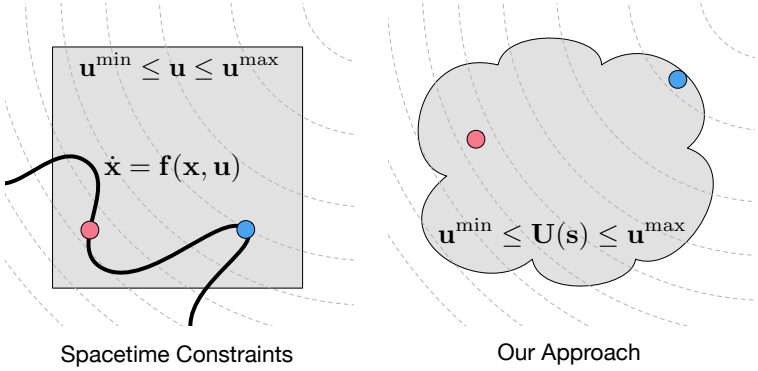


Figure 3.2: Illustration of the main difference between spacetime constraints (left) and our approach (right). In spacetime con- straints, control force limits are easy to enforce, because they can be encoded as linear inequality constraints (grey shaded region, left). However, quadrotor dynamics are 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).

3.5 Progress Curve Optimization

In Section 3.4, 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 3.2.

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 5^{th} 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,

$$\begin{aligned} \mathbf{s}_{i+1} &= \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}v_i)dt_i \\ \text{subject to} \quad & v^{\min} \leq v_i \leq v^{\max} \\ \text{where} \quad \mathbf{M} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{N} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned} \tag{3.8}$$

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 (3.8) to arrive at the following equality constraint,

$$\mathbf{s}_{i+1} = \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}v_i)\frac{ds_i}{\dot{s}_i} \tag{3.9}$$

In equation (3.8), $v^{\min}$ 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 Section 3.7.2 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 1^{st} 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,

$$\begin{aligned} \mathbf{S}^*, \mathbf{V}^* &= \arg\min_{\mathbf{S},\mathbf{V}} \sum_i (\dot{s}_i - \dot{s}_i^{\text{ref}})^2 \\ \text{subject to} \quad & \mathbf{s}_{i+1} = \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}v_i)\frac{ds_i}{\dot{s}_i} \\ v^{\min} \leq v_i \leq v^{\max} \quad & \dot{\mathbf{q}}^{\min} \leq \dot{\mathbf{Q}}(\mathbf{s}_i) \leq \dot{\mathbf{q}}^{\max} \\ \dot{s}_i > 0 \quad & \mathbf{u}^{\min} \leq \mathbf{U}(\mathbf{s}_i) \leq \mathbf{u}^{\max} \end{aligned} \tag{3.10}$$

Figure 3.3: 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.

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 3.4. $\mathbf{S}$ and $\mathbf{V}$ are decision variables, everything else is problem data.

Improving Computational Efficiency To improve computational efficiency, we make two important approximations to the problem in (3.10).

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 [122]. 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 (3.4). Therefore, we use the following approximation for equation (3.4),

$$\bar{\mathbf{u}} = \bar{\mathbf{B}}^{\star} \left[\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \right] \tag{3.11}$$

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 [17]. With this reasoning in mind,

we replace our *nearly* linear equality constraints with the following linear approximations,

$$\mathbf{s}_{i+1} = \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}\mathbf{v}_i)\frac{ds_i}{\bar{\dot{s}}_i} \tag{3.12}$$

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,

$$\bar{\mathbf{B}}^{\star[i]} = \mathbf{B}^\star(\mathbf{q}^{[i-1]}) \qquad\qquad \bar{\dot{s}}^{[i]} = \dot{s}^{[i-1]} \tag{3.13}$$

where the superscript notation refers to our solver's iteration count. Boyd refers to this strategy as *quasi-linearization* [17].

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 chapter, we stop updating $\bar{\dot{s}}$ after 10 iterations. We evaluate the speed and accuracy tradeoffs associated with this choice in Section 3.6.

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

$$\begin{aligned} \mathbf{S}^*, \mathbf{V}^* &= \arg\min_{\mathbf{S},\mathbf{V}} \sum_i (\dot{s}_i - \dot{s}_i^{\text{ref}})^2 \\ \text{subject to} \quad \mathbf{s}_{i+1} &= \mathbf{s}_i + (\mathbf{M}\mathbf{s}_i + \mathbf{N}v_i)\frac{ds_i}{\bar{\dot{s}}_i} \\ v^{\min} \le v_i \le v^{\max} \quad & \dot{\mathbf{q}}^{\min} \le \dot{\mathbf{Q}}(\mathbf{s}_i) \le \dot{\mathbf{q}}^{\max} \\ \dot{s}_i > 0 \quad & \mathbf{u}^{\min} \le \bar{\mathbf{U}}(\mathbf{s}_i) \le \mathbf{u}^{\max} \end{aligned} \tag{3.14}$$

where $\bar{\mathbf{U}}(\mathbf{s}_i)$ is our closed form expression for control forces that includes the approximation from equation (3.11). The problem in (3.14) 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 (3.14) using the commercially available non-convex solver SNOPT [55].

Initialization The optimization problem in (3.14) 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 (3.8). 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 trajectory, rather than an overly aggressive infeasible trajectory, allows our solver to take advantage of more globally meaningful gradient information.

3.6 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 Section 3.7.3. We 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 3.3, 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 in Chapter 2.

Qualitative Comparison with Spacetime Constraints In Figure 3.4, 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 3.5, we show that our algorithm remains within the prescribed velocity and control force limits, when generating the output trajectory shown in Figure 3.4.

Infeasible Trajectory Dataset We based our dataset of infeasible quadrotor camera trajectories on raw data from the user study described in Chapter 2. 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.

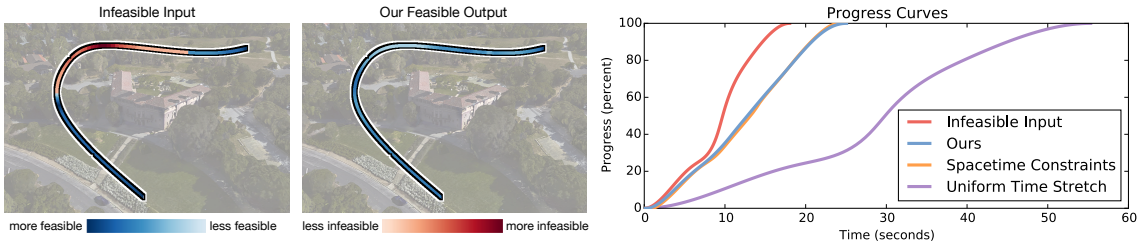


Figure 3.4: An aggressive infeasible trajectory (far left), the feasible output trajectory produced by our algorithm (near left), and the feasible progress curves produced by our algorithm and spacetime constraints for this trajectory (right). As a baseline, we include the progress curve obtained by 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.

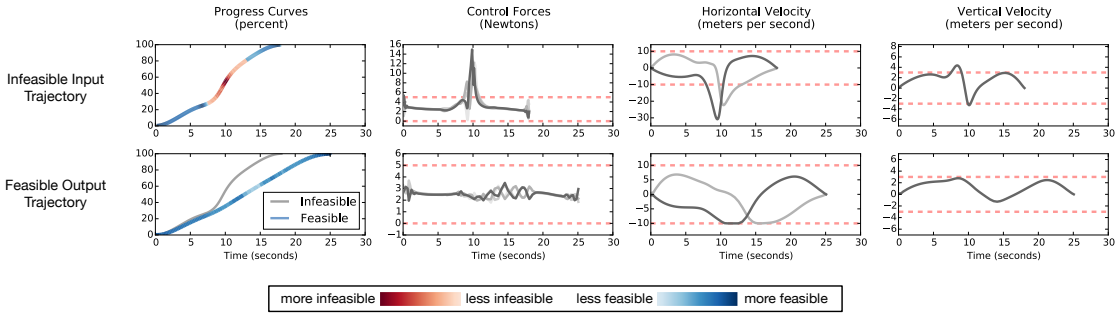


Figure 3.5: Our algorithm perturbs the timing of the infeasible input trajectory shown in Figure 3.4 as little as possible, while remaining within our quadrotor's physical limits. To demonstrate this behavior, we plot the infeasible input (top) and feasible output (bottom) progress curves, control force 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.

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 3.6.

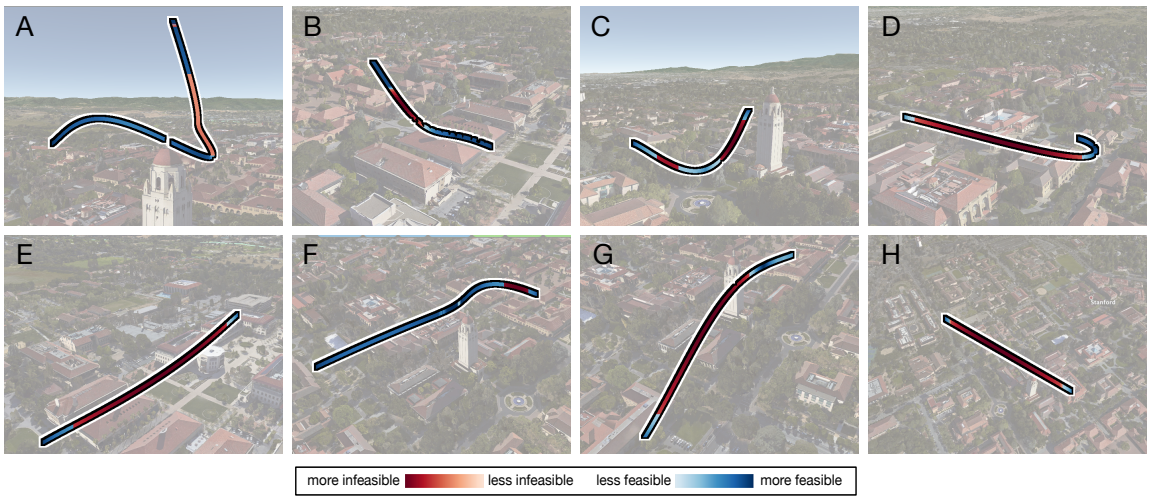


Figure 3.6: Our dataset of infeasible quadrotor camera trajectories. We show the trajectories in their spatial context, and color them according to how severely they violate our quadrotor's physical limits. We use the letters in this figure to refer to individual trajectories throughout this chapter.

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 3.7.

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 available solver SNOPT [55]. We initialize both approaches using the initialization strategy described in Section 3.5. 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 [122]. 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

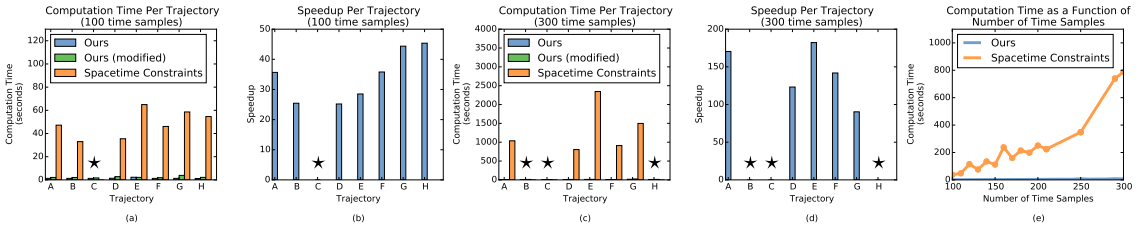


Figure 3.7: Computational performance and scaling behavior of our algorithm on our dataset of infeasible trajectories. When we solve for trajectories discretized at a moderate resolution of 100 time samples, our algorithm runs in less than 2 seconds, and is between 25× and 45× faster than spacetime constraints (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 $\star$. 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 3.6. On plots a, c, and e, lower is better.

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 3.8.

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 3.9. Due to the relatively large time steps involved in these simulations, we applied LQR feedback control [124] 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.

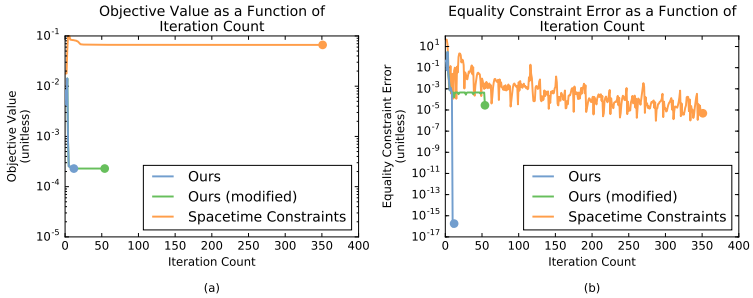


Figure 3.8: Convergence behavior of our algorithm on trajectory B from our dataset of infeasible trajectories, discretized into 100 time samples. This is the trajectory where spacetime constraints converged the fastest. Our algorithm, the modification of our algorithm described in Section 3.6, and spacetime 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 l_2 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.

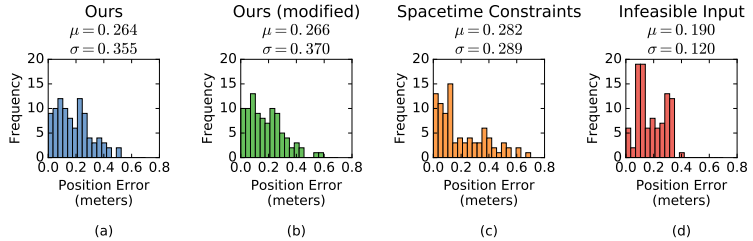


Figure 3.9: Accuracy of our algorithm in predicting the results of a rigid body physics simulation. We simulate the control trajectory produced by our algorithm using a 5^{th} order accurate rigid body simulator, and we measure how well the results of the simulation match the state space trajectory produced by our algorithm (a). We repeat this experiment using the modification of our algorithm described in Section 3.6 (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.

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.

3.7 Appendix

3.7.1 Quadrotor Manipulator Matrices

In this subsection, we define the quadrotor manipulator matrices, attempting to be as concise as possible. We refer the reader to Section 2.10.2 for a more detailed derivation.

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

$$\mathbf{q} = \begin{bmatrix} \mathbf{p}_q \\ \mathbf{e}_q \end{bmatrix} \tag{3.15}$$

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,

$$\begin{aligned} \mathbf{H}(\mathbf{q}) &= \begin{bmatrix} m\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}} \end{bmatrix} \\ \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) &= \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\dot{\mathbf{A}} - \left(\mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}\dot{\mathbf{e}}\right)_\times \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A} \end{bmatrix} \\ \mathbf{G}(\mathbf{q}) &= \begin{bmatrix} -\mathbf{f}_e \\ \mathbf{0}_{3\times1} \end{bmatrix} \\ \mathbf{B}(\mathbf{q}) &= \begin{bmatrix} \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f} \\ \mathbf{M}_\tau \end{bmatrix} \end{aligned} \tag{3.16}$$

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

$$\begin{aligned}\mathbf{M_f} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{M}_\tau &= \begin{bmatrix} ds_\alpha & ds_\beta & -ds_\beta & -ds_\alpha \\ \gamma & -\gamma & \gamma & -\gamma \\ -dc_\alpha & dc_\beta & dc_\beta & -dc_\alpha \end{bmatrix}\end{aligned} \tag{3.17}$$

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 in Chapter 2. See the detailed derivation in Section 2.10.2 for details.

3.7.2 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,

$$\begin{aligned} v^{\min} &= v^{\min}_{\text{ref}} - \lambda_{\text{proportional}}(v^{\max}_{\text{ref}} - v^{\min}_{\text{ref}}) - \lambda_{\text{fixed}} \\ v^{\max} &= v^{\max}_{\text{ref}} + \lambda_{\text{proportional}}(v^{\max}_{\text{ref}} - v^{\min}_{\text{ref}}) + \lambda_{\text{fixed}} \end{aligned} \tag{3.18}$$

where $v^{\min}_{\text{ref}}$ and $v^{\max}_{\text{ref}}$ are the minimum and maximum 5^{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 (3.18) above.

3.7.3 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,

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{bmatrix} \tag{3.19}$$

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

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

$$\begin{aligned} \text{subject to} \quad \mathbf{x}_0 &= \mathbf{x}_0^{\text{ref}} \\ \mathbf{x}_N &= \mathbf{x}_N^{\text{ref}} \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) dt_i \\ \mathbf{x}^{\min} &\leq \mathbf{x}_i \leq \mathbf{x}^{\max} \\ \mathbf{u}^{\min} &\leq \mathbf{u}_i \leq \mathbf{u}^{\max} \\ dt^{\min} &\leq dt_i \leq dt^{\max} \end{aligned} \tag{3.20}$$

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 Section 3.7.1 as follows,

$$\mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) = \begin{bmatrix} \dot{\mathbf{q}}_i \\ \ddot{\mathbf{q}}_i \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{q}}_i \\ \mathbf{H}_i^{-1}(\mathbf{B}_i \mathbf{u}_i - \mathbf{C}_i \dot{\mathbf{q}}_i - \mathbf{G}_i) \end{bmatrix} \tag{3.21}$$

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 [136], in the sense that the optimizer is free to stretch time. This freedom is required in order to ensure that a feasible solution exists.

Chapter 4

Submodular Trajectory Optimization for Aerial 3D Scanning

Small consumer drones equipped with high-resolution cameras are emerging as a powerful tool for large-scale aerial 3D scanning. In order to obtain high-quality 3D reconstructions, a drone must capture images that densely cover the scene. Additionally, 3D reconstruction methods typically require surfaces to be viewed from multiple viewpoints, at an appropriate distance, and with sufficient angular separation (i.e., baseline) between views. Existing autonomous flight planners do not always satisfy these requirements, which can be difficult to reason about, even for a skilled human pilot manually controlling a drone. Furthermore, the limited battery life of consumer drones provides only 10–15 minutes of flight time, making it even more challenging to obtain high-quality 3D reconstructions.

In lieu of manual piloting, commercial flight planning tools generate conservative trajectories (e.g., a lawnmower or orbit pattern at a safe height above the scene) that attempt to cover the scene while respecting flight time budgets [3, 102]. However, because these trajectories are generated with no awareness of the scene geometry, they tend to over-sample some regions (e.g., rooftops), while under-sampling others (e.g., facades, overhangs, and fine details), and therefore sacrifice reconstruction quality.

In this chapter, we introduce a method to automate aerial 3D scanning, by planning good camera trajectories for reconstructing large 3D scenes (see Figure 1.4). Our method relies on a mathematical model that evaluates the usefulness of a camera trajectory for the purpose of 3D scanning. Given a coarse estimate of the scene geometry as input, our model quantifies how well a trajectory covers

the scene, and also quantifies the diversity and appropriateness of views along the trajectory. Using this model for scene coverage, our method generates trajectories that maximize coverage, subject to a travel budget. We bootstrap our method using coarse scene geometry, which we obtain using the imagery acquired from a short initial flight over the scene.

We formulate our trajectory planning task as a reward-collecting graph optimization problem known as *orienteering*, that combines aspects of the traveling salesman and knapsack problems, and is known to be NP-hard [58, 129]. However, unlike the additive rewards in the standard orienteering problem, our rewards are non-additive, and globally coupled through our coverage model. We make the observation that our coverage model exhibits an intuitive diminishing returns property known as *submodularity* [81], and therefore we must solve a *submodular orienteering* problem. Although submodular orienteering is strictly harder than additive orienteering, it exhibits useful structure that can be exploited. We propose a novel transformation of our submodular orienteering problem into an additive orienteering problem, and we solve the additive problem as an integer linear program. We leverage submodularity extensively throughout the derivation of our method, to obtain approximate solutions with strong theoretical guarantees, and dramatically reduce computation times.

We demonstrate the utility of our method by using it to scan three large outdoor scenes: a barn, an office building, and an industrial site. We also quantitatively evaluate our algorithm in a photorealistic video game simulator. In all our experiments, we obtain noticeably higher-quality 3D reconstructions than strong baseline methods.

4.1 Related Work

Aerial 3D Scanning and Mapping High-quality 3D reconstructions of very large scenes can be obtained using offline multi-view stereo algorithms [49] to process images acquired by drones [101]. Real-time mapping algorithms for drones have also been proposed, that take as input either RGBD [62, 87, 95, 121] or RGB [135] images, and produce as output a 3D reconstruction of the scene. These methods are solving a reconstruction problem, and do not, themselves, generate drone trajectories. Several commercially available flight planning tools have been developed to assist with 3D scanning [3, 102]. However, these tools only generate conservative lawnmower and orbit trajectories above the scene. In contrast, our algorithm generates trajectories that cover the scene as thoroughly as possible, ultimately leading to higher-quality 3D reconstructions.

Generating trajectories that *explore* an unknown environment, while building a map of it, is a classical problem in robotics [126]. Exploration algorithms have been proposed for drones based on local search heuristics [131], identifying the frontiers between known and unknown parts of the scene [61, 113], maximizing newly visible parts of the scene [14], maximizing information gain [19, 20], and imitation learning [24]. A closely related problem in robotics is generating trajectories that *cover* a known environment [50]. Several coverage path planning algorithms have been proposed for drones

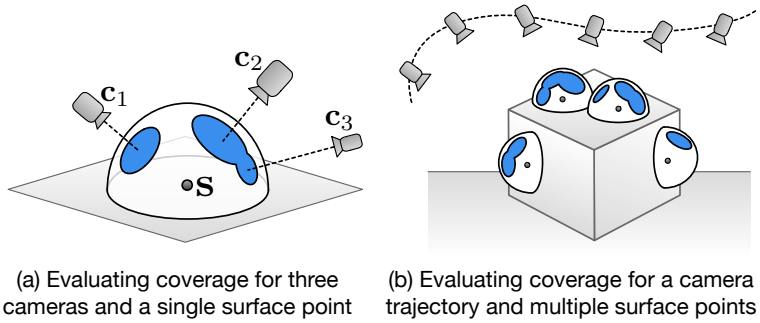
[6, 13, 60, 63]. In an especially similar spirit to our work, Heng et al. propose to reconstruct an unknown environment by executing alternating exploration and coverage trajectories [60]. However, existing strategies for exploration and coverage do not explicitly account for the domain-specific requirements of multi-view stereo algorithms (e.g., observing the scene geometry from a diverse set of viewing angles). Moreover, existing exploration and coverage strategies have not been shown to produce visually pleasing multi-view stereo reconstructions, and are generally not evaluated on multi-view stereo reconstruction tasks. In contrast, our trajectories cover the scene in a way that explicitly accounts for the requirements of multi-view stereo algorithms, and we evaluate the multi-view stereo reconstruction performance of our algorithm directly.

Several path planning algorithms have been proposed for drones, that explicitly attempt to maximize multi-view stereo reconstruction performance [41, 64, 96, 110]. These algorithms are similar in spirit to ours, but adopt a two phase strategy for generating trajectories. In the first phase, these algorithms select a sequence of *next-best-views* to visit, ignoring travel costs. In the second phase, they find an efficient path that connects the previously selected views. In contrast, our algorithm reasons about these two problems – selecting good views and routing between them – jointly in a unified global optimization problem, enabling us to generate more rewarding trajectories, and ultimately higher-quality 3D reconstructions.

View Selection and Path Planning The problem of optimizing the placement (and motion) of sensors to improve performance on a perception task is a classical problem in computer vision and robotics, where it generally goes by the name of *active vision*, e.g., see the comprehensive surveys [23, 111, 123]. We discuss directly related work not included in these surveys here. A variety of active algorithms for 3D scanning with ground-based range scanners have been proposed, that select a sequence of next-best-views [80], and then find an efficient path to connect the views [44, 142]. In a similar spirit to our work, Wang et al. propose a unified optimization problem that selects rewarding views, while softly penalizing travel costs [134]. We adapt these ideas to account for the domain-specific requirements of multi-view stereo algorithms, and we impose a hard travel budget constraint, which is an important safety requirement when designing drone trajectories.

Several algorithms have been proposed to select an appropriate subset of views for multi-view stereo reconstruction [40, 65, 89, 90], and to optimize coverage of a scene [54, 91]. However, these methods do not model travel costs between views. In contrast, we impose a hard constraint on the travel cost of the path formed by the views we select.

Submodular Path Planning Submodularity [81] has been considered in path planning scenarios before, first in the theory community [21, 22], and more recently in the artificial intelligence [115, 116, 145] and robotics [60, 63] communities. The coverage path planning formulation of Heng et al. [60] is similar to ours, in the sense that both formulations use the same technique for approximating coverage [70, 71]. We extend this formulation to account for the domain-specific requirements of



(a) Evaluating coverage for three cameras and a single surface point

(b) Evaluating coverage for a camera trajectory and multiple surface points

Figure 4.1: Our coverage model for quantifying the usefulness of camera trajectories for multi-view stereo reconstruction. More useful trajectories cover more of the hemisphere of viewing angles around surface points. (a) An illustrative example showing coverage of a single surface point with three cameras. Each camera covers a circular disk on a hemisphere around the surface point $\mathbf{s}$, and the total solid angle covered by all the disks determines the total usefulness of the cameras. Note that the angular separation (i.e., baseline) between cameras $\mathbf{c}_2$ and $\mathbf{c}_3$ is small and leads to diminishing returns in their combined usefulness. (b) The usefulness of a camera trajectory, integrated over multiple surface points, is determined by summing the total covered solid angle for each of the individual surface points. Our model naturally encourages diverse observations of the scene geometry, and encodes the eventual diminishing returns of additional observations.

multi-view stereo algorithms, and we evaluate the multi-view stereo reconstruction performance of our algorithm directly.

4.2 Technical Overview

In order to generate scanning trajectories, our algorithm leverages a coarse estimate of the scene geometry. Initially, we do not have any estimate of the scene geometry, so we adopt an *explore-then-exploit* approach.

In the *explore* phase, we fly our drone (i.e., we command our drone to fly autonomously) along a default trajectory at a safe distance above the scene, acquiring a sequence of images as we are flying. We land our drone, and subsequently feed the acquired images to an open-source multi-view stereo pipeline, thereby obtaining a coarse estimate of the scene geometry, and a strictly conservative estimate of the scene's free space. We include a more detailed discussion of our *explore* phase in Section 4.6.1.

In the *exploit* phase, we use this additional information about the scene to plan a scanning trajectory that attempts to maximize the fidelity of the resulting 3D reconstruction. At the core of our planning algorithm, is a coverage model that accounts for the domain-specific requirements of multi-view stereo reconstruction (Section 4.3). Using this model, we generate a scanning trajectory that maximizes scene coverage, while respecting the drone's limited flight time (Section 4.4). We

fly the drone along our scanning trajectory, acquiring another sequence of images. Finally, we land our drone again, and we feed all the images we have acquired to our multi-view stereo pipeline to obtain a detailed 3D reconstruction of the scene.

4.3 Coverage Model for Camera Trajectories

In this section, we model the usefulness of a camera trajectory for multi-view stereo reconstruction, in terms of how well it covers the scene geometry. We provide an overview of our coverage model in Figure 4.1.

In reality, the most useful camera trajectory is the one that yields the highest-quality 3D reconstruction of the scene. However, it is not clear how we would search for such a camera trajectory directly, without resorting to flying candidate trajectories and performing expensive 3D reconstructions for each of them. In contrast, our coverage model only roughly approximates the true usefulness of a camera trajectory. However, as we will see in the following section, our coverage model: (1) is motivated by established best practices for multi-view stereo image acquisition; (2) is easy to evaluate; (3) only requires a coarse estimate of the scene geometry as input; and (4) exhibits submodular structure, which will enable us to efficiently maximize it.

Best Practices for Multi-View Stereo Image Acquisition As a rule of thumb, it is recommended to capture an image every 5–15 degrees around an object, and it is generally accepted that capturing images more densely will eventually lead to diminishing returns in the fidelity of the 3D reconstruction [49]. Similarly, close-up and fronto-parallel views can help to resolve fine geometric details, because these views increase the effective resolution of estimated depth images, and contribute more reliable texture information to the reconstruction [133]. We explicitly encode these best practices for multi-view stereo image acquisition into our coverage model.

Formal Definition Given a candidate camera trajectory and approximate scene geometry as a triangle mesh, our goal is to quantify how well the trajectory covers the scene geometry. We first uniformly sample the camera trajectory to generate a discrete set C, consisting of individual camera poses $\mathbf{c}_{0:I}$. Similarly, we uniformly sample oriented surface points $\mathbf{s}_{0:J}$ from the scene geometry. For each oriented surface point $\mathbf{s}_j$, we define an oriented hemisphere H_j around it. For each surface point $\mathbf{s}_j$ and camera $\mathbf{c}_i$, we define a circular disk D_i^j that covers an angular region of the hemisphere H_j, centered at the location where $\mathbf{c}_i$ projects onto H_j (see Figure 4.1). When the surface point $\mathbf{s}_j$ is not visible from the camera $\mathbf{c}_i$, we define the disk D_i^j to have zero radius, and we truncate the extent of each disk so that it does not extend past the equator of H_j. We define the total covered region of the hemisphere H_j as the union of all the disks that partially cover H_j (see Figure 4.1),

referring to this total covered region as $V_j = \bigcup_{i=0}^{I} D_i^j$. We define our coverage model as follows,

$$f(C) = \sum_{j=0}^{J} \int_{V_j} w_j(\mathbf{h}) d\mathbf{h} \tag{4.1}$$

where the outer summation is over all hemispheres; $\int_{V_j} d\mathbf{h}$ refers to the surface integral over the covered region V_j; and $w_j(\mathbf{h})$ is a non-negative weight function that assigns different reward values for covering different parts of H_j. Our model can be interpreted as quantifying how well a set of cameras covers the scene's *surface light field* [32, 137]. We include a method for efficiently evaluating our coverage model in Section 4.6.2.

To encourage close-up views, we set the radius of D_i^j to decay exponentially as the camera $\mathbf{c}_i$ moves away from the surface point $\mathbf{s}_j$. To encourage fronto-parallel views, we design each function $w_j(\mathbf{h})$ to decay in a cosine-weighted fashion, as the hemisphere location $\mathbf{h}$ moves away from the hemisphere pole. We include our exact formulation for D_i^j and $w_j(\mathbf{h})$ in Section 4.6.3.

Submodularity Roughly speaking, a set function is submodular if the marginal reward for adding an element to the input set always decreases, as more elements are added to the input set [81]. Our coverage model is submodular, because all coverage functions with non-negative weights are submodular [81]. Submodularity is a useful property to identify when attempting to optimize a set function, and is often referred to as the discrete analogue of convexity. We will leverage submodularity extensively in the following section, as we derive our algorithm for generating camera trajectories that maximizing coverage.

4.4 Generating Optimal Camera Trajectories

We provide an overview of our algorithm in Figure 4.2. Our approach is to formulate a reward-collecting optimization problem on a graph. The nodes in the graph represent camera positions, the edges represent Euclidean distances between camera positions, and the rewards are collected by visiting new nodes. The goal is to find a path that collects as much reward as possible, subject to a budget constraint on the total path length. This general problem is known as the *orienteering problem* [58, 129].

A variety of approaches have been proposed to approximately solve the orienteering problem, which is NP-hard. However, these methods are not directly applicable to our problem, because they assume that the rewards on nodes are additive. But the total reward we collect in our problem is determined by our coverage model, which does not exhibit additive structure. Indeed, the marginal reward we collect at a node might be very large, or very small, depending on the entire set of other nodes we visit.

The marginal reward we collect at each node also depends strongly on the orientation of our

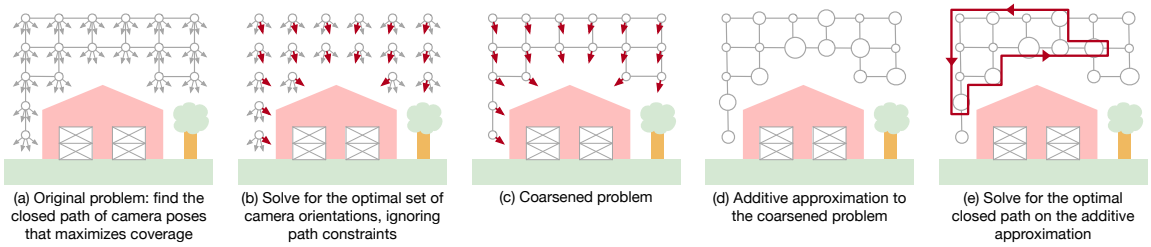


Figure 4.2: Overview of our algorithm for generating camera trajectories that maximize coverage. (a) Our goal is to find the optimal closed path of camera poses through a discrete graph. (b) We begin by solving for the optimal camera orientation at every node in our graph, ignoring path constraints. (c) In doing so, we remove the choice of camera orientation from our problem, coarsening our problem into a more standard form. (d) The solution to the problem in (b) defines an approximation to our coarsened problem, where there is an additive reward for visiting each node. (e) Finally, we solve for the optimal closed path on the additive approximation defined in (d).

camera. In other words, our orienteering problem involves extra choices – how to orient the camera at each visited node – and these choices are globally coupled through our submodular coverage function. Therefore, even existing algorithms for *submodular orienteering* [21, 22, 60, 115, 116, 145] are not directly applicable to our problem, because these algorithms assume there are no extra choices to make at each visited node.

Our strategy will be to apply two successive problem transformations. First, we leverage submodularity to solve for the approximately optimal camera orientation at every node in our graph, ignoring path constraints (Figure 4.2b, Section 4.4.1). In doing so, we remove the choice of camera orientation from our orienteering problem, thereby coarsening it into a more standard form (Figure 4.2c). Second, we leverage submodularity to construct a tight additive approximation of our coverage function (Figure 4.2d, Section 4.4.2). In doing so, we relax our coarsened submodular orienteering problem into a standard additive orienteering problem. We formulate this additive orienteering problem as a compact integer linear program, and solve it approximately using a commercially available solver (Figure 4.2e, Section 4.4.3).

Preprocessing We begin by constructing a discrete set of all the possible camera poses we might include in our path. We refer to this set as our *ground set* of camera poses, C. We construct this set by uniformly sampling a user-defined bounding box that spans the scene, then uniformly sampling a downward-facing unit hemisphere to produce a set of look-at vectors that our drone camera can actuate. We define our ground set as the Cartesian product of these positions and look-at vectors. We construct the graph for our orienteering problem as the grid graph of all the unique camera positions in C, pruned so that it is entirely restricted to the known free space in the scene (see Section 4.2).

Our Submodular Orienteering Problem Let $\mathbf{P} = (\mathbf{p}_0, \mathbf{v}_0), (\mathbf{p}_1, \mathbf{v}_1), \ldots, (\mathbf{p}_q, \mathbf{v}_q)$ be a camera path through our graph, represented as a sequence of camera poses taken from our ground set. We represent each camera pose as a position $\mathbf{p}_i$ and a look-at vector $\mathbf{v}_i$. We would like to find the optimal path $\mathbf{P}^\star$ as follows,

$$\begin{aligned} \mathbf{P}^\star &= \arg\max_{\mathbf{P}} f(C_{\mathbf{P}}) \\ \text{subject to} \quad & l(\mathbf{P}) \leq B \quad \mathbf{p}_0 = \mathbf{p}_q = \mathbf{p}_{\text{root}} \end{aligned} \tag{4.2}$$

where $C_{\mathbf{P}} \subseteq C$ is the set of all the unique camera poses along the path; $l(\mathbf{P})$ is the length of the path; B is a user-defined travel budget; and $\mathbf{p}_{\text{root}}$ is the position where our path must start and end. For safety reasons, we would also like to design trajectories that consume close to, but no more than, some fixed fraction of our drone's battery (e.g., 80% or so). However, constraining battery consumption directly is difficult to express in our orienteering formulation, so we model this constraint indirectly by imposing a budget constraint on path length. We make the observation that our problem is intractable in its current form, because it requires searching over an exponential number of paths through our graph. This observation motivates the following two problem transformations.

4.4.1 Solving for Optimal Camera Orientations

Our goal in this subsection is to solve for the optimal camera orientation at every node in our graph, ignoring path constraints. We achieve this goal with the following relaxation of the problem in equation (4.2). Let $C_S \subseteq C$ be a subset of camera poses from our ground set. We would like to find the optimal subset of camera poses $C_S^\star$ as follows,

$$\begin{aligned} C_S^\star &= \arg\max_{C_S} f(C_S) \\ \text{subject to} \quad & |C_S| = N \quad C_S \in \mathcal{M} \end{aligned} \tag{4.3}$$

where $|C_S|$ is the cardinality of C_S; N is the total number of unique positions in our graph; and the constraint $C_S \in \mathcal{M}$ enforces mutual exclusion, where we are allowed to select at most one camera orientation at each node in our graph. In this relaxed problem, we are attempting to maximize coverage by selecting exactly one camera orientation at each node in our graph. We can interpret such a solution as a coarsened ground set for the problem in equation (4.2), thereby transforming it into a standard submodular orienteering problem.

Because our coverage function is submodular, the problem in equation (4.3) can be solved very efficiently, and to within 50% of global optimality, with a very simple greedy algorithm [81]. Roughly speaking, the greedy algorithm selects camera poses from our ground set in order of marginal reward, taking care to respect the mutual exclusion constraint, until no more elements can be selected. Submodularity can also be exploited to significantly reduce the computation time required by the

greedy algorithm (e.g., from multiple hours to a couple of minutes, for the problems we consider in this chapter) [81]. The approximation guarantee in this subsection relies on the fact that selecting more camera poses never reduces coverage, i.e., our coverage function exhibits a property known as *monotonicity* [81]. We include a more detailed discussion of the greedy algorithm, and provide pseudocode, in Section 4.6.4.

4.4.2 Additive Approximation of Coverage

Our goal in this subsection is to construct an additive approximation of coverage. In other words, we would like to define an additive reward at each node in our graph, that closely approximates our coverage function for arbitrary subsets of visited nodes.

To construct our additive approximation, we draw inspiration from the approach of Iyer et al. [70, 71]. We begin by choosing a permutation of elements in our coarsened ground set. Let $\mathbf{C} = \mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_N$ be our permutation, where $\mathbf{c}_i$ is the i^{th} element of our coarsened ground set in permuted order. Let $C_i = \{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{i-1}\}$ be the subset containing the first i elements of our permutation. We define the additive reward for each element in our permutation as $\tilde{f}_i = f(C_i \cup \mathbf{c}_i) - f(C_i)$. For an arbitrary subset C_S, our additive approximation is simply the sum of additive rewards for each element in C_S. Due to submodularity, this additive approximation is guaranteed to be exact for all subsets C_i, and to underestimate our true coverage function for all other subsets. This guarantee is useful for our purposes, because any solution we get from optimizing our additive approximation will yield an equal or greater reward on our true coverage function.

When choosing a permutation, it is generally advantageous to place camera poses with the greatest marginal reward at the front of our permutation. With this intuition in mind, we form our permutation by sorting the camera poses in our coarsened ground set according to their marginal reward. Fortunately, we have already computed this ordering in Section 4.4.1 using the greedy algorithm. So, we simply reuse this ordering to construct our additive approximation.

4.4.3 Orienteering as an Integer Linear Program

After constructing our additive approximation of coverage, we obtain the following additive orienteering problem,

$$\begin{aligned} \mathbf{P}^\star = \arg\max_{\mathbf{P}} \sum_{C_\mathbf{P}} \tilde{f}_i \\ \text{subject to} \quad l(\mathbf{P}) \leq B \quad \mathbf{p}_0 = \mathbf{p}_q = \mathbf{p}_{\text{root}} \end{aligned} \tag{4.4}$$

where $\tilde{f}_i$ is the additive reward for each unique node along the path $\mathbf{P}$. In its current form, it is still not clear how to solve this problem efficiently, because we must still search over an exponential number of paths through our graph. Fortunately, we can express this problem as a compact integer linear program, using a formulation suggested by Letchford et al. [84]. We transform our undirected

graph into a directed graph, and we define integer variables to represent if nodes are visited and directed edges are traversed. Remarkably, we can constrain the configuration of these integer variables to form only valid paths through our graph, with a compact set of linear constraints. We include a more detailed derivation of this formulation in Section 4.6.5.

Leveraging the formulation suggested by Letchford et al., we convert the problem in equation (4.4) into a standard form that can be given directly to an off-the-shelf solver. We use the modeling language CVXPY [35] to specify our problem, and we use the commercially available Gurobi Optimizer [59] as the back-end solver. Solving integer programming problems to global optimality is NP-hard, and can take a very long time, so we specify a solver time limit of 5 minutes. Gurobi returns the best feasible solution it finds within the time limit, along with a worst-case optimality gap. In our experience, Gurobi consistently converges to a close-to-optimal solution in the allotted time (i.e., typically with an optimality gap of less than 10%). At this point, the resulting orienteering trajectory can be safely and autonomously executed on our drone.

4.5 Evaluation

In all the experiments described in this section, we execute all drone flights at 2 meters per second, with a total travel budget of 960 meters (i.e., an 8 minute flight) unless otherwise noted. All flights generate 1 image every 3.5 meters. Each method has the same travel budget, and generates roughly 275 images. Small variations in the number of generated images are possible, due to differences in how close each method gets to the travel budget. We describe our drone hardware, data acquisition pipeline, and experimental methodology in more detail in Section 4.6.

Real-World Reconstruction Performance We evaluated the real-world reconstruction performance of our algorithm by using it to scan three large outdoor scenes: a barn, an office building, and an industrial site.[1] We show results from these experiments in Figures 1.4 and 4.3, as well as in Section 4.6.6. We compared our reconstruction results to two baseline methods: Overhead and Random.

Overhead. We designed Overhead to generate trajectories that are representative of those produced by existing commercial flight planning software [3, 102]. Overhead generates a single flight at at a safe height above the scene; consisting of an orbit path that always points the camera at the center of the scene; followed by a lawnmower path that always points the camera straight down.

Random. We designed Random to have roughly the same level of scene understanding as our algorithm, except that Random does not optimize our coverage function. We gave Random access

[1] We conducted this experiment with an early implementation of our method that differs slightly from the implementation used in our other experiments. In particular, the graph of camera positions used in this experiment included diagonal edges. We subsequently excluded diagonal edges to enable our integer programming formulation to scale to larger problem instances.

Figure 4.3: Qualitative comparison of the 3D reconstructions obtained from an overhead trajectory, a random trajectory, and our trajectory for two real-world scenes. Our reconstructions contain noticeably fewer visual artifacts than the baseline reconstructions. In all our experiments, we control for the flight time, battery consumption, number of images, and quality settings used in the 3D reconstruction.

to the graph of camera positions generated by our algorithm, which had been pruned according to the free space in the scene. RANDOM generates trajectories by randomly selecting graph nodes to visit and traveling to them via shortest paths, until no more nodes can be visited due to the travel budget. RANDOM always points the camera towards the center of the scene, which is a reasonable strategy for the scenes we consider in this chapter.

During our *explore* phase, we generate an orbit trajectory exactly as we do for OVERHEAD. For the scenes we consider in this chapter, this initial orbit trajectory is always less than 250 meters.

When generating 3D reconstructions, our algorithm and RANDOM have access to the images from our *explore* phase, but OVERHEAD does not. The images in our *explore* phase are nearly identical to the orbit images from OVERHEAD, and would therefore provide OVERHEAD with negligible additional information, so all three methods are directly comparable. We generated 3D reconstructions using the commercially available Pix4Dmapper Pro software [103], configured with maximum quality settings.

Reconstruction Performance on a Synthetic Scene We evaluated our algorithm using a photorealistic video game simulator, which enabled us to measure reconstruction performance relative to known ground truth geometry and appearance. We show results from this experiment in Figure 4.4 and Table 4.1.

Our experimental design here is exactly as described previously, except we acquired images by

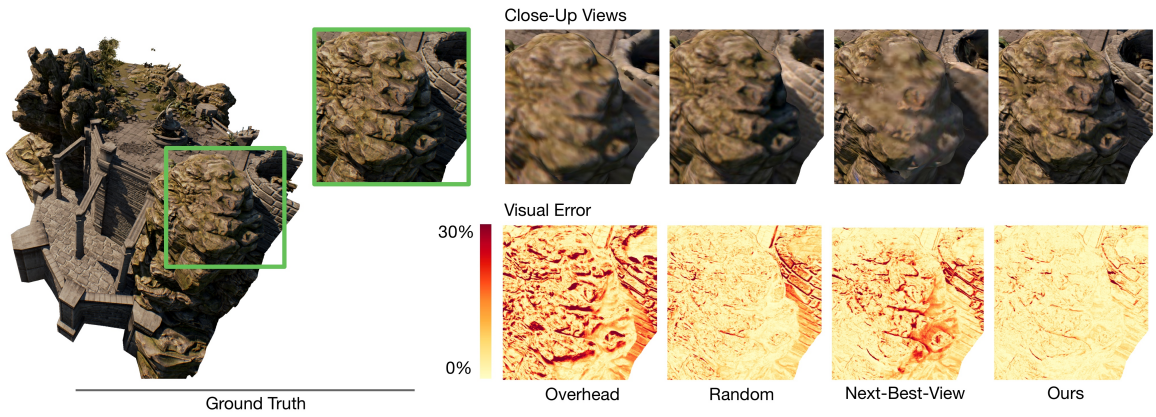


Figure 4.4: Quantitative comparison of the 3D reconstructions obtained from an overhead trajectory, a random trajectory, a next-best-view trajectory, and our trajectory for our synthetic scene. We show close-up renderings of each reconstruction, as well as per-pixel visual error, relative to a ground truth rendering of the scene. Our method leads to quantitatively lower visual error than baseline methods.

Method	Accuracy Error (mm)	Completeness Error (mm)	Visual Error (%)
Overhead	170.2	583.8	7.1
Random	126.5	557.2	4.4
Next-Best-View	122.8	330.7	3.6
Ours	**115.2**	**323.3**	**3.3**

Table 4.1: Quantitative comparison of the 3D reconstructions obtained from an overhead trajectory, a random trajectory, a next-best-view trajectory, and our trajectory for our synthetic scene. For all the columns in this table, lower is better. We report the mean per-pixel visual error across all of our test views, where 100% per-pixel error corresponds to the l_2 norm of the difference between black and white in RGB space. Our method quantitatively outperforms baseline methods, both geometrically (i.e., in terms of accuracy and completeness) and visually.

programmatically maneuvering a virtual camera in the Unreal Engine [43], using the UnrealCV Python library [104]. We also included an additional baseline method, NEXT-BEST-VIEW, that greedily selects nodes according to their marginal submodular reward, and finds an efficient path to connect them using the Approx-TSP algorithm [28] until no more nodes can be added due to the travel budget. This method is intended to be representative of the *next-best-view* planning strategies that occur frequently in the literature [44, 63, 80, 142], including those that have been applied to aerial 3D scanning [41, 64, 96, 110].

We chose the GRASS LANDS environment [42] as our synthetic test scene because it is freely available, has photorealistic lighting and very detailed geometry, and depicts a large outdoor scene that would be well-suited for 3D scanning with a drone.

We evaluated geometric reconstruction quality by measuring *accuracy* and *completeness* relative

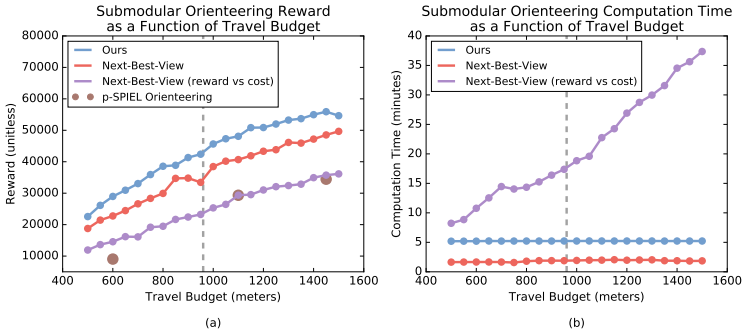


Figure 4.5: Quantitative comparison of submodular orienteering algorithms on our synthetic scene. (a) Submodular reward as a function of travel budget. Our algorithm consistently obtains more reward than other algorithms. All reconstruction results in this chapter were produced with a budget of 960 meters (i.e., 8 minutes at 2 meters per second), shown with a grey dotted line. For this budget, we obtain 20% more reward than next-best-view planning. The p-SPIEL Orienteering algorithm [116] failed to consistently find a solution. (b) Computation time as a function of travel budget. On this plot, lower is better. In terms of computation time, our algorithm is competitive with, but more expensive than, next-best-view planning. We do not show computation times for the p-SPIEL Orienteering algorithm, because it took over 4 hours in all cases where it found a solution.

to a ground truth point cloud [5, 77]. We obtained our ground truth point cloud by rendering reference depth images arranged on an inward-looking sphere around the scene, taking care to manually remove any depth images that were inside objects. We also evaluated visual reconstruction quality by measuring *per-pixel visual error*, relative to ground truth RGB images rendered from the same inward-looking sphere around the scene [132]. When evaluating per-pixel visual error, we took care to only compare pixels that contain geometry from inside the scanning region-of-interest for our scene.

When evaluating geometric quality, we obtained point clouds for each method by running VisualSFM [138, 139, 140, 141], followed by the Multi-View Environment [48], followed by Screened Poisson Surface Reconstruction [74], and finally by uniformly sampling points on the reconstructed triangle mesh surface. When evaluating visual quality, we obtained textured 3D models for each method using the surface texturing algorithm of Waechter et al. [133].

Submodular Orienteering Performance We evaluated the submodular orienteering performance of our algorithm on our synthetic scene. We performed this experiment after we have solved for the optimal camera orientation at every node in our graph, to facilitate the comparison of our algorithm to other submodular orienteering algorithms [116, 145]. We show results from this experiment in Figure 4.5.

In this experiment, we included a baseline method that behaves identically to Next-Best-View, except it greedily selects nodes according to the ratio of marginal reward to marginal cost [145]. We

implemented all algorithms in Python, except for the p-SPIEL Orienteering algorithm [116], where we used the MATLAB implementation provided by the authors. We performed this experiment on a Mid 2015 Macbook Pro with a 2.8 GHz Intel Core i7 processor and 16GB of RAM.

4.6 Appendix

4.6.1 Explore Phase: Estimating the Scene Geometry and Free Space

In this section, we describe our high-level strategy for data acquisition, as well as our multi-view stereo processing pipeline for estimating the scene geometry and free space. Our goals in this section are twofold. First, we would like to obtain a rough estimate of the scene geometry, in the form of a triangle mesh in real-world coordinates. Second, we would like to obtain a strictly conservative estimate of the free space, in the form of an occupancy volume in real-world coordinates. Together, these complimentary scene representations will enable us to plan trajectories that maximize the quality of a 3D reconstruction.

Drone Camera Hardware Our system requires access to a drone camera that can be commanded to fly along pre-specified camera trajectories, given in GPS coordinates (i.e., latitude, longitude, altitude). The drone camera must also take geotagged images of the scene at roughly constant intervals (e.g., one image every few meters). We require geotagged images, in order to establish a correspondence between the physical world and the arbitrary coordinate system of our multi-view stereo reconstruction. In our system, we use the 3D Robotics Solo drone [4] equipped with a GoPro Hero 4 camera.

User Input Our system requires two bounding boxes as input, each of which can easily be drawn by a user on a 2D map (e.g., Google Maps) and extruded vertically. The first bounding box, $\mathbf{b}_s$, specifies the volume that the user wants to scan. The second bounding box, $\mathbf{b}_c$, specifies the volume that the drone is allowed to fly within. We require that $\mathbf{b}_c$ be made sufficiently tall, so that the entire ceiling of $\mathbf{b}_c$ is free of obstacles. This requirement is necessary to give our drone some non-trivial region of free space that it can safely explore, prior to resolving any scene geometry. We do not assume initially that any other space is free.

Initial Camera Trajectory Our system begins by flying the drone in an elliptical orbit trajectory around the ceiling of $\mathbf{b}_c$, pointing the camera at the center of $\mathbf{b}_s$. For the scenes we consider in this chapter, this initial elliptical trajectory consumes roughly 25% of our drone's travel budget.

Dense Multi-View Stereo After we have flown an initial camera trajectory, the first step in our image processing pipeline is to run the structure-from-motion software VisualSFM [138, 139,

140, 141] on the sequence of images acquired by our drone. Our next step is to run the depth map reconstruction step of the Multi-View Environment (MVE) [48]. In our implementation, we set the *scale* parameter of MVE such that the reconstructed depth maps will be at a resolution of at least 512×512 (e.g., if the images we originally capture are 2048×2048, we set the *scale* parameter to 2). For the scenes we consider in this chapter, computing structure-from-motion and dense multi-view stereo takes roughly 15 minutes, and is the dominant cost in our explore phase.

Mapping Between Coordinate Systems We perform all our trajectory planning in real-world coordinates, using the UTM coordinate system [127]. UTM coordinates are similar to GPS coordinates, in the sense that they describe positions on the surface of the Earth. However, unlike GPS coordinates, the UTM coordinate axes are approximately orthogonal, and the default UTM units are meters. Together, these properties make UTM coordinates well-suited for trajectory planning.

In order to use our reconstructed scene geometry for trajectory planning, we must establish a correspondence between the UTM coordinate system and the arbitrary coordinate system of the reconstructed geometry. We estimate this correspondence by considering our sequence of geotagged camera positions (in UTM coordinates), and the sequence of estimated camera positions recovered during our structure-from-motion step (in reconstruction coordinates). We estimate the similarity transform that maps from reconstruction coordinates to UTM coordinates using standard numerical techniques [119].

Obtaining an Oriented Point Cloud and Occupancy Volume We generate an oriented point cloud of the scene geometry, and an occupancy volume of the scene's free space, using our own modified version of MVE. Given a collection of camera poses and corresponding depth images, we obtain our point cloud by projecting each depth image into reconstruction coordinates, and then into UTM coordinates.

We generate an occupancy volume of the free, occupied, and unknown space using a simple space carving algorithm. For every depth observation in every depth image, we project the depth observation and corresponding camera into UTM coordinates. This projection defines a ray that starts at the camera and ends at the depth observation. In our occupancy volume, we mark every interior voxel along this ray as being free, and we mark the last voxel along this ray as being occupied. After having generated our occupancy volume in this way, we create an extra safety buffer around obstacles by dilating the occupied space by 4 meters in every direction. This safety buffer is conservative, in the sense that it is roughly twice the diameter of the largest localization errors we observed on our drone hardware during field testing. We consider all unmarked voxels to be unknown. We store our occupancy volume efficiently in an OctoMap data structure [66].

After this space carving step is complete, we assume that our occupancy volume strictly underestimates the free space in the scene. This assumption is justified by the following three observations. First, MVE aggressively filters outlier depth observations. So, although MVE does not produce a

Scene	Extent of $\mathbf{b}_s$ (m)	Extent of $\mathbf{b}_c$ (m)	Grid Spacing (m)	Number of Grid Nodes
Barn	34×24×28	44×29×58	4.0	1440
Office Building	40×25×37	50×30×47	3.5	1890
Industrial Site	34×25×31	54×30×51	4.0	1456
Grass Lands	48×30×41	78×45×71	4.5	2880

Table 4.2: Grid spacing parameters for each of our scenes.

depth observation at every pixel of every depth image, the observations that MVE does produce tend to be very reliable. Second, we only mark a voxel as being free if there is explicit evidence for doing so (i.e., there is some camera in front of it, that has observed some surface point behind it). Third, we create a large safety buffer around all observed surfaces, to account for any small errors in the MVE depth images.

Surface Reconstruction Given an oriented point cloud of our scene geometry in UTM coordinates, we obtain a water-tight triangle mesh surface by running the Screened Poisson Surface Reconstruction algorithm [74] on the point cloud. In our implementation, we set the *depth* parameter of Screened Poisson Surface Reconstruction to 7, which produces a coarse triangle mesh quickly. To maximize the effective resolution of our surface reconstruction, we clip the input point cloud against the user-specified bounding box $\mathbf{b}_s$. As a post-processing step, we apply the *surface trimming* tool implemented in the Screened Poisson Surface Reconstruction codebase with a *trim* parameter of 7, and we use MeshLab [26] to remove isolated triangle mesh connected components with fewer than 2000 faces. At this point in our pipeline, we have a triangle mesh representation of the scene geometry, as well as an occupancy volume representation of the scene's free space, both in UTM coordinates.

Sampling Details We uniformly sample points on our reconstructed surface using the Poisson Disk Sampling technique [29] implemented in MeshLab [26]. In our implementation, we request that MeshLab return 1500 surface samples. MeshLab is not guaranteed to return exactly this many surface samples, and in practice returns roughly 2000 surface samples.

In our implementation, we sample camera positions in the bounding box $\mathbf{b}_c$ by constructing a uniform grid with a grid spacing that that we specify per scene, ranging from 3.5 to 4.5 meters per grid node. If our specified grid spacing does not align exactly with the borders of $\mathbf{b}_c$, we independently adjust the grid spacing along each axis to be slightly more dense, so as to align with exactly with the borders of $\mathbf{b}_c$. For the scenes we consider in this chapter, this range of grid densities leads to grids containing between 1440 and 2880 grid nodes. We choose our grid spacing to strike a balance between being as dense as possible, while also not leading to too many integer variables in our integer programming formulation. We include the exact grid spacing parameters for each of

our scenes in Table 4.2.

We sample camera orientations by generating uniformly spaced samples on a downward-facing hemisphere. In our implementation, we generate 50 uniformly spaced samples using standard numerical techniques [34].

4.6.2 Efficiently Evaluating Coverage

In this section, we demonstrate how to evaluate coverage efficiently for an arbitrary subset of cameras. It is important that we can evaluate coverage efficiently, because we must evaluate it many times, for many different subsets, in our algorithm for generating scanning trajectories. Our strategy will be to apply a discrete Monte Carlo approximation of coverage that we can evaluate using matrix operations.

Evaluating Coverage using Matrix Operations It is not immediately obvious how to evaluate our coverage model efficiently, due to the unpleasant irregular integration domain in equation (4.1). Our approach begins by replacing this irregular domain with a regular domain, using an indicator function representation to mask out the covered region V_j,

$$f(C) = \sum_{j=0}^{J} \int_{H_j} w_j(\mathbf{h}) v_j(\mathbf{h}) d\mathbf{h} \tag{4.5}$$

where the notation $\int_{H_j} d\mathbf{h}$ refers to a surface integral over the entire hemisphere H_j; and $v_j(\mathbf{h})$ is an indicator function that equals 1 when the hemisphere location $\mathbf{h}$ is in the covered region V_j, and equals 0 otherwise. Next, we approximate our regular hemispherical integral with a discrete Monte Carlo approximation,

$$F(C) = \sum_{j=0}^{J} 2\pi \frac{1}{K} \sum_{k=1}^{K} w_j(\mathbf{h}_k) v_j(\mathbf{h}_k) \tag{4.6}$$

where $F(C) \approx f(C)$ is the discrete approximation of our coverage function; the constant 2π arises because we are integrating over the hemisphere; K is the number of discrete samples on the hemisphere; and k is an index that refers to the discrete hemisphere sample location $\mathbf{h}_k$. In this form, it becomes clear that we can represent our discrete coverage model with the following dot product,

$$F(C) = \mathbf{w}^T \mathbf{v} \tag{4.7}$$

where $\mathbf{w}$ is the stacked vector of all our (normalized) weight function values $2\pi \frac{1}{K} w_j(\mathbf{h}_k)$; and $\mathbf{v}$ is the stacked vector of all our coverage indicator function values $v_j(\mathbf{h}_k)$. We refer to $\mathbf{v}$ as a *coverage indicator vector*. In our implementation, we set $K = 256$, and we generate our hemisphere sample locations $\mathbf{h}_k$ using standard numerical techniques [34].

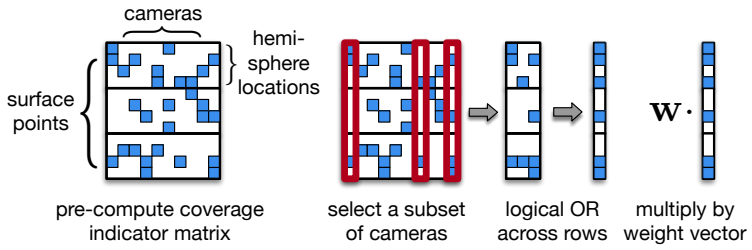


Figure 4.6: Our matrix representation for efficiently evaluating coverage, for arbitrary subsets of cameras. We begin by pre-computing a coverage indicator matrix that represents the independent contribution of each camera to the total coverage. We recover the coverage indicator vector for a particular subset of cameras by selecting the appropriate subset of columns from our matrix, and performing a logical OR operation across the rows of the resulting matrix. We evaluate our coverage model for this subset of cameras by multiplying the coverage indicator vector with a weight vector. We use this efficient matrix representation to evaluate coverage for many different subsets of cameras, in our algorithm for generating scanning trajectories.

Evaluating Coverage for any Subset of Cameras In the previous subsection, we showed how to evaluate our coverage model efficiently for a particular set of cameras. In order to optimize our model, we will need to efficiently evaluate coverage for many different subsets of cameras. In other words, we would like to efficiently evaluate the following expression,

$$F(C_S) = \mathbf{w}^T \mathbf{v}_S \tag{4.8}$$

where $C_S \subseteq C$ is an arbitrary subset of cameras; and $\mathbf{v}_S$ is the coverage indicator vector for the cameras in C_S.

We can efficiently evaluate our model for any subset of cameras $C_S \subseteq C$ by pre-computing a *coverage indicator matrix*, $\mathbf{D}$ (see Figure 4.6). Intuitively speaking, we define the matrix $\mathbf{D}$ to represent the independent contribution of each camera to the total coverage. Each row of $\mathbf{D}$ corresponds to a particular sampling location on a particular hemisphere. Each column of $\mathbf{D}$ corresponds to a particular camera in C. Stating our definition of $\mathbf{D}$ more precisely, we set the entry of $\mathbf{D}$ corresponding to {hemisphere H_j, hemisphere location $\mathbf{h}_k$, camera $\mathbf{c}_i$} equal to 1 if {hemisphere H_j, hemisphere location $\mathbf{h}_k$} is covered by the disk D_i^j, and equal to 0 otherwise.

4.6.3 Designing the Parameters of Our Coverage Model

In this section, we provide the exact parameters we use in our coverage model. In our implementation, we set the radius of each disk D_i^j to decay exponentially as a camera moves away from a surface point. We chose an exponential decay model as a matter of convenience, because it was intuitive for us to specify disk size in terms of a decay half-life. When a surface point is not visible from a

camera, we define the corresponding disk to have zero radius. We determine visibility by raycasting against our coarse triangle mesh estimate of the scene geometry.

Stating our model more precisely, we define each disk as the intersection of a hollow unit sphere, centered at the surface point; and a solid cone that has its apex at the surface point, and is oriented towards the camera. We control the solid angle of each disk by controlling the apex angle of this cone. We define the apex *half angle* θ_i^j of each cone in radians as follows,

$$\theta_i^j = \theta_{\max} 2^{\frac{-\max(t_i^j - t_0, 0)}{t_{\text{half}}}} \tag{4.9}$$

where $\theta_{\max}$ is the largest possible apex half angle in radians; t_0 is the distance in meters from a camera to a surface point, beyond which our half angle doesn't get any larger; t_{half} is the decay half-life of the our half angle in meters; and t_i^j is the distance from camera $\mathbf{c}_i$ to surface point $\mathbf{s}_j$ in meters. In our implementation, we set $\theta_{\max} = \frac{1}{2}\frac{\pi}{180}45$ radians, $t_0 = 4$ meters, and $t_{\text{half}} = 12$ meters.

In our implementation, we set each weight function $w_j(\mathbf{h})$ to have a cosine-weighted falloff as follows,

$$w_j(\mathbf{h}) = \cos \alpha_{\mathbf{h}} \tag{4.10}$$

where $\alpha_{\mathbf{h}}$ is the angle formed by the hemisphere pole and the vector from the hemisphere origin to the hemisphere location $\mathbf{h}$.

4.6.4 The Greedy Algorithm for Maximizing Submodular Functions

In this section, we provide additional details regarding the greedy algorithm for maximizing submodular functions. The problem in equation 4.2 can be solved to within 50% of global optimality with the greedy algorithm in Listing 3. To see that this is the case, we first make the observation that our coverage model is *monotone* (i.e., selecting more camera poses never reduces our coverage score) [81]. We also make the observation that our mutual exclusion constraint is an instance of a more general mathematical object known as a *partition matroid constraint* [81]. It is known that maximizing a monotone submodular function, subject to a cardinality constraint and a partition matroid constraint, can be solved to within 50% of global optimality with the greedy algorithm [81].

The greedy algorithm can be expensive to run on large problems, because it must evaluate the submodular function many times (Listing 3, line 4). Indeed, the greedy algorithm takes several hours to run on the problem instances we consider in this chapter. Fortunately, we can leverage submodularity to avoid a very large fraction of this computational effort. The main insight in this approach, is that the marginal reward for adding an element only ever decreases as more elements are added to the greedy solution, due to submodularity. Therefore, we can maintain a priority queue of marginal rewards, and we can *lazily* update this queue. When a marginal reward in our queue is stale, it can be interpreted as an upper bound on the true marginal reward. This insight can be used to safely skip a large fraction of function evaluations, in an approach known as the *lazy*

Input:

- A ground set of elements C.
- A monotone submodular function $f(C_S)$ to be maximized, where $C_S \subseteq C$.
- A cardinality constraint $|C_S| = N$.
- A mutual exclusion constraint $C_S \in \mathcal{M}$, that defines which elements in C are incompatible.

Output:

- A subset $C_S \subseteq C$ that maximizes f to within 50% of global optimality, subject to the cardinality constraint and mutual exclusion constraint.

```
1: S ← ∅
2: G ← C
3: for i ← 0 to N do
4:     g* ← arg max_{g∈G} f(S ∪ g) − f(S)
5:     I ← all the elements in G that are incompatible with g*
6:     S ← S ∪ g*
7:     G ← G \ {I, g*}
8: C_S ← S
```

Listing 3: The greedy algorithm for maximizing a monotone submodular function, subject to a cardinality constraint and a mutual exclusion constraint. Evaluating the arg max expression on line 4 requires evaluating f once for each element in the set G, leading to many function evaluations.

greedy algorithm [81]. We provide pseudocode for the lazy greedy algorithm, including the necessary modifications to handle our mutual exclusion constraint, in Listing 4. For the problem instances we consider in this chapter, the lazy greedy algorithm reduces computation time by several orders of magnitude.

4.6.5 Detailed Formulation of Orienteering as an Integer Linear Program

In this section, we transform the orienteering problem in Section 4.4.3, into a standard integer linear program. In this derivation, we follow the formulation of Letchford et al. [84]. However, we express the objective and constraints from this formulation in matrix form, which makes the problem easier to express in a high-level modeling language (e.g., CVXPY [35]).

We begin by replacing each undirected edge in our graph with two directed *arcs*, each with the same cost as the original undirected edge. We use the term arc to refer to a directed edge in our modified graph. We define the indicator vector $\mathbf{x}$ to represent whether or not each arc is traversed. We define the indicator vector $\mathbf{y}$ to represent whether or not each node in our graph is visited. We define the vector $\mathbf{g}$ to contain the cumulative costs incurred so far, when beginning to traverse each arc. Finally, we define the constant vector $\mathbf{f}$ to contain the additive reward at each each node, and we define the constant vector $\mathbf{r}$ to contain the instantaneous cost of traversing each arc.

To help coordinate our optimization problem, we also define inbound and outbound *node-arc indicator matrices*, $\mathbf{A}^{\text{in}}$ and $\mathbf{A}^{\text{out}}$ respectively. The rows of these matrices represent the nodes in our graph, and columns represent the arcs. We set the entry of $\mathbf{A}^{\text{in}}$ corresponding to {node n, arc

Input:

- A ground set of elements C.
- A monotone submodular function $f(C_S)$ to be maximized, where $C_S \subseteq C$.
- A cardinality constraint $|C_S| = N$.
- A mutual exclusion constraint $C_S \in \mathcal{M}$, that defines which elements in C are incompatible.

Output:

- A subset $C_S \subseteq C$ that maximizes f to within 50% of global optimality, subject to the cardinality constraint and mutual exclusion constraint.

$S \leftarrow \varnothing$
$G \leftarrow C$
for all $g \in G$ **do**
  $m_g \leftarrow f(S \cup g) - f(S)$
  $u_g \leftarrow$ True
for $i \leftarrow 0$ to N **do**
  $g^\star \leftarrow \arg\max_{g \in G} m_g$
  while not $u_{g^\star}$ **do**
    $m_{g^\star} \leftarrow f(S \cup g^\star) - f(S)$
    $u_{g^\star} \leftarrow$ True
    $g^\star \leftarrow \arg\max_{g \in G} m_g$
  $I \leftarrow$ all the elements in G that are incompatible with $g^\star$
  $S \leftarrow S \cup g^\star$
  $G \leftarrow G \setminus \{I, g^\star\}$
  for all $g \in G$ **do**
    $u_g \leftarrow$ False
$C_S \leftarrow S$

Listing 4: The lazy greedy algorithm for maximizing a monotone submodular function, subject to a cardinality constraint and a mutual exclusion constraint. This algorithm maintains a lazily updated list of marginal rewards, m_g. When a marginal reward in this list is stale (i.e., when $u_g =$ False), the value m_g can be interpreted as an upper bound on the true marginal reward, due to submodularity. This observation can be used to skip a large number of function evaluations. The lazy greedy algorithm drastically reduces the number of times f must be evaluated, as compared to the greedy algorithm.

$m\}$ equal to 1 if arc m is an inbound arc for node n, and equal to 0 otherwise. We define $\mathbf{A}^{\text{out}}$ similarly, but with respect to the outbound arcs. We use the notation $\mathbf{A}_R$ to refer to the row of $\mathbf{A}$ corresponding to the root node (i.e., the node where our path must start and end), and we use the notation $\mathbf{A}_{R'}$ to refer to all the other rows of $\mathbf{A}$.

With this notation in place, we define our integer linear program as follows,

$$\mathbf{x}^\star, \mathbf{y}^\star, \mathbf{g}^\star = \arg\max_{\mathbf{x},\mathbf{y},\mathbf{g}} \mathbf{f}^T\mathbf{y} \tag{4.11a}$$

$$\text{subject to} \quad \mathbf{r}^T\mathbf{x} \leq B \tag{4.11b}$$

$$\begin{aligned} \mathbf{A}^{\text{out}}_R\mathbf{x} &\geq \mathbf{1} & \mathbf{A}^{\text{out}}\mathbf{x} &= \mathbf{A}^{\text{in}}\mathbf{x} \\ \mathbf{A}^{\text{out}}_{R'}\mathbf{x} &\geq \mathbf{y}_{R'} & \mathbf{A}^{\text{out}}_{R'}\mathbf{g} - \mathbf{A}^{\text{in}}_{R'}\mathbf{g} &= \mathbf{A}^{\text{in}}_{R'}\mathbf{T}\mathbf{x} \end{aligned} \tag{4.11c}$$

$$0 \leq \mathbf{g} \leq \mathbf{U}\mathbf{x} \tag{4.11d}$$

$$\mathbf{x} \in \{0,1\}^M \qquad \mathbf{y} \in \{0,1\}^N \qquad \mathbf{g} \in \mathbf{R}^M_+ \tag{4.11e}$$

where the matrix $\mathbf{T} = \mathbf{diag}(\mathbf{r})$ helps us to calculate an intermediate matrix of instantaneous costs for inbound arcs; the matrix $\mathbf{U} = \mathbf{diag}(\mathbf{1}B - \mathbf{r})$ helps us to define the upper bounds for our cumulative cost variable $\mathbf{g}$; and M is the number of arcs in our graph and N is the number of nodes. In this formulation, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{g}$ are decision variables, everything else is problem data. Letchford et al. refer to this integer linear program as a *single-commodity flow* formulation for the orienteering problem [84].

The objective in this optimization problem (4.11a) attempts to maximize the reward we collect, by activating as many nodes as possible. The constraint in (4.11b) enforces our maximum budget on total path length. The constraints in (4.11c) specify that: at least one of the outbound arcs for the root node must be active; the number of active outbound arcs must match the number of active inbound arcs at each node; if an outbound arc is active, then the node at its tail must be active; and the difference in cumulative cost at an outbound arc and a corresponding inbound arc, must match the instantaneous cost of the inbound arc. The constraint in (4.11d) enforces that the cumulative costs incurred so far are less than the total budget. Finally, the constraints in (4.11e) specify that $\mathbf{x}$ and $\mathbf{y}$ are Boolean indicator vectors, and that $\mathbf{g}$ is non-negative and real-valued.

The problem in equation (4.11) is expressed in a standard form that can be given directly to an off-the-shelf solver. Given a solution to the problem in equation (4.11), we recover the sequence of visited nodes by following outbound arcs from the root until there are no more nodes to visit. For each visited node, we look up its corresponding camera pose in our coarsened ground set to obtain a sequence of camera poses.

4.6.6 Evaluation Details

In this section, we provide additional real-world reconstruction results, as well as additional methodological details for our synthetic scene experiments.

Real-World Reconstruction Results We provide high-resolution renderings of our real-world reconstructions in Figure 4.7.

Methodological Details for our Synthetic Scene Experiments When acquiring images for each method, we configured UnrealCV [104] to produce RGB images at a resolution of 512×512.

When generating high-resolution reconstructions, we set the *scale* parameter of MVE [48] to 0. We set the *depth* parameter of the Screened Poisson Surface Reconstruction algorithm [74] to 9, which produces a detailed triangle mesh without overfitting to high-frequency noise in the point clouds estimated by MVE. As in our explore phase, we clipped the input point cloud against the bounding box $\mathbf{b}_s$. As a post-processing step, we applied the *surface trimming* tool implemented in the Screened Poisson Surface Reconstruction codebase with a *trim* parameter of 7, and we used MeshLab [26] to remove isolated triangle mesh connected components with fewer than than 50000 faces. We used the *Gaussian damping* option when running the surface texturing algorithm of Waechter et al. [133].

When collecting ground truth data, we configured UnrealCV to produce RGB and depth images at a resolution of 2048×2048. We generated 256 uniformly distributed views on an inward-looking sphere around our scene using standard numerical techniques [34]. We set the center of our sphere to be the center of the bounding box $\mathbf{b}_s$, and we set its diameter to be 1.5× the maximum dimension of $\mathbf{b}_s$, which in our case was 72 meters. We took care to manually remove images from our set of ground truth views that were inside objects.

When measuring geometric accuracy and completeness, we subsampled our ground truth point cloud to obtain a uniformly sampled point cloud, using the *spatial subsampling* method implemented in CloudCompare [27]. When performing this subsampling operation, we set the minimum space between points to the 95^{th} percentile of the distribution of nearest-neighbor distances in the original ground truth point cloud, which in our case was 0.0133 meters (i.e., prior to subsampling, 95% of nearest-neighbor distances in the original ground truth point cloud were less than 0.0133 meters). We sampled points on each reconstructed triangle mesh using the *mesh sampling* method implemented in CloudCompare [27], where we set the desired sampling density to be $\frac{1}{0.0133^2} = 5653.2308$ points per square meter.

When measuring per-pixel visual error, we departed slightly from the formulation suggested by Waechter et al. [132]. Waechter et al. suggest measuring visual error by computing zero-mean normalized cross-correlation (NCC) over small patches, so as to be robust to low-frequency discrepancies in illumination between the ground truth images and the test images (i.e., the images of a reconstructed 3D model). In our synthetic scene, illumination is controlled perfectly, so we do not need this additional robustness. Moreover, we found that NCC did not successfully localize the noticeable texturing artifacts on baseline reconstructed 3D models (e.g., the texturing artifacts visible in Figure 4.4). For these reasons, we computed per-pixel differences in RGB space, instead of computing NCC.

Because we rendered ground truth views using UnrealCV, we needed to map the reconstructed 3D models into Unreal coordinates before rendering them and comparing them to the ground truth

views. In practice, our procedure for mapping into Unreal coordinates (see Section 4.6.1) is accurate to within a few pixels, but is not perfect. To prevent our visual quality metric from being dominated by very small coordinate system alignment errors, we computed per-pixel differences in the following way. For each ground truth pixel location $\mathbf{z}$, we computed the difference between the ground truth image at $\mathbf{z}$, and each pixel in a 9×9 patch centered at $\mathbf{z}$ in the test image. We took the minimum difference over this patch as the per-pixel difference at $\mathbf{z}$. In our experience, this approach provided a good balance between being robust to small alignment errors, while also effectively localizing noticeable texturing artifacts in the reconstructed 3D models.

Figure 4.7: Qualitative comparison of the 3D reconstructions produced by overhead, random, and our trajectories for three real-world scenes. Our reconstructions contain noticeably fewer artifacts than the baseline reconstructions. In all our experiments, we control for the flight time, battery consumption, number of images, and quality settings used in the 3D reconstruction. Best viewed digitally at high resolution.

Chapter 5

Conclusions

Our ultimate goal in this dissertation was to make drone cameras easier to use and more expressive. To make progress towards this goal, we used the principles of trajectory optimization to express high-level drone behaviors, and to solve for low-level drone actions. We also used domain knowledge to design specialized algorithms that were tailored to particular camera tasks. In other words, we did not attempt to solve general drone intelligence. Instead, we used domain-specific methods to make two targeted creative tasks – drone cinematography and drone 3D scanning – easier for users to specify.

In Chapter 2, we introduced drone cinematography as a research problem. Specifically, we introduced a set of design principles for quadrotor shot planning tools, and based on these principles, we built a tool for designing quadrotor camera shots. We added four components to existing quadrotor mission planning tools: (1) visual shot design; (2) virtual preview; (3) precise timing control; and (4) visual feasibility feedback. To support our tool, we introduced a physical model for quadrotor cameras, and we derived an algorithm for generating camera trajectories that agree with our physical model. 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 Chapter 3, we extended the capabilities of our shot planning tool, enabling it to automatically generate trajectories that respect the physical limits of quadrotor cameras. 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 dynamically feasible trajectories. We implemented our algorithm in our shot planning tool, and we ran performance benchmarks on a dataset of infeasible quadrotor camera trajectories. We found that our approach is between 25× and 180× faster than a spacetime constraints approach. We successfully captured real video footage using the trajectories generated by our algorithm. We showed that the resulting videos are faithful to virtual shot previews, even

when the trajectories being executed are at our quadrotor's physical limits.

In Chapter 4, we turned our attention from cinematography to 3D scanning. We proposed an intuitive coverage model for aerial 3D scanning, and we made the observation that our model is submodular. We leveraged submodularity to develop a computationally efficient method for generating scanning trajectories, that reasons jointly about coverage rewards and travel costs. We evaluated our method by using it to scan three large real-world scenes, and a scene in a photorealistic video game simulator. We found that our method results in quantitatively higher-quality 3D reconstructions than baseline methods, both geometrically and visually.

5.1 Future Work

In the future, consumer robotics will play a critical role in a wide range of creative tasks. We saw concrete examples of this general idea in Chapters 2 and 3, where we used drones to help us create ambitious cinematography shots that would be challenging or impossible to obtain otherwise. But the work in this dissertation is far from the last word on this topic, and there is plenty of exciting work left to do. For example, drones can now autonomously film a person skiing down a mountain[1] [117], and *activity forecasting* methods [76], as well as computational models of human aesthetic preferences [112], might soon be used to plan better-looking shots in real-time. Filming with a team of drones [97] might soon enable new kinds of *bullet time* camera effects [146] when filming action sports in unstructured environments. Computational models of *social saliency* [8, 100] could enable drones to capture outdoor social events autonomously and unintrusively. Advances in drone hardware could enable dramatic new types of *hyperlapse* photography [79].

We also saw examples of robot-enabled creativity support in Chapter 4, where we used drones to generate high-quality virtual models of an environment, e.g., for use in a video game or virtual reality experience. More generally, drones will enable us to capture the physical world with unprecedented coverage and scale. Cooperating teams of drones might soon be used to scan large scenes very quickly. New kinds of drones might be used to scan indoor environments at unprecedented levels of detail. Cooperating drone cameras and drone lights might be used to capture richer material representations.

The idea of using robots to help us create is much bigger than the examples we've seen in this dissertation, because there are so many creative tasks that we would like help with. For example, drone lights are now being used to create swarming 3D displays[2] [69]. However, thinking about this new type of computational display immediately leads us to a host of open questions. How should we specify appropriate goals for a swarming 3D display, and how would we compile an animation

[1] As an example of particularly compelling skiing footage shot with a drone, we encourage the reader to watch this video: http://youtu.be/51xBnjQK3u8. All the footage in this video was captured fully autonomously with no human pilot using the Skydio R1 drone.

[2] The drone swarm display technology developed by Intel can be seen here: http://youtu.be/KhDEEN4gcpI

into a swarm of collision-free drone trajectories?

Moving beyond aerial drones, there are many creative tasks at home that household chore robots might help us with. A particularly fascinating example is cooking. How should we specify goals for a cooking robot, and how might we convert these goals into long-range robot trajectories? Similar open questions arise in a variety of other creative tasks around the house, e.g., assembling and re-arranging furniture.

Perhaps an even greater degree of situational awareness is required outside the home, where we would like help from robots to build on-demand public infrastructure like roads and bridges, e.g., in disaster relief scenarios.

Across a wide range of domains, we want robots to help us *create*. The resulting computational problems are hard, but the language of trajectory optimization gives us the tools we need to make progress. And we should try to make progress now, because the next generation of consumer-facing robots is just around the corner.

Bibliography

[1] 3D Robotics. IRIS+. http://3drobotics.com/iris, 2014.

[2] 3D Robotics. Solo. http://3drobotics.com/solo, 2015.

[3] 3D Robotics. Site Scan. http://3dr.com, 2017.

[4] 3D Robotics. Solo. https://3dr.com/solo-drone, 2017.

[5] Henrik Aanæs, Rasmus Ramsbøl Jensen, George Vogiatzis, Engin Tola, and Anders Bjorholm Dahl. Large-scale data for multiple-view stereopsis. *IJCV*, 120(2), 2016.

[6] Kostas Alexis, Christos Papachristos, Roland Siegwart, and Anthony Tzes. Uniform coverage structural inspection path-planning for micro aerial vehicles. In *IEEE International Symposium on Intelligent Control 2015*.

[7] APM. APM Autopilot Suite. http://ardupilot.com, 2015.

[8] Ido Arev, Hyun Soo Park, Yaser Sheikh, Jessica Hodgins, and Ariel Shamir. Automatic editing of footage from multiple social cameras. *Transactions on Graphics (Proc. SIGGRAPH 2014)*, 33(4), 2014.

[9] Daniel Arijon. *Grammar of the Film Language*. Hastings House Publishers, 1976.

[10] Brooks Barnes. Drone exemptions for Hollywood pave the way for widespread use. New York Times, September 25 2014.

[11] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for Use in Computer Graphics & Geometric Modeling*. Morgan Kaufmann Publishers, 1987.

[12] John T. Betts. A survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2), 1998.

[13] Andreas Bircher, Kostas Alexis, Michael Burri, Philipp Oettershagen, Sammy Omari, Thomas Mantel, and Roland Siegwart. Structural inspection path planning via iterative viewpoint

resampling with application to aerial robotics. In *International Conference on Robotics and Automation (ICRA) 2015*.

[14] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon "next-best-view" planner for 3D exploration. In *International Conference on Robotics and Automation (ICRA) 2016*.

[15] Y. Bouktir, M. Haddad, and T. Chettibi. A prototype of an autonomous controller for a quadrotor UAV. In *Mediterranean Conference on Control and Automation 2008*.

[16] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[17] Steven Boyd. Convex optimization II (course notes for Stanford EE364b). http://stanford.edu/class/ee364b, 2008.

[18] Adam Bry, Charles Richter, Abraham Bachrach, and Nicholas Roy. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *International Journal of Robotics Research*, 34(7), 2015.

[19] Benjamin Charrow, Gregory Kahn, Sachin Patil, Sikang Liu, Ken Goldberg, Pieter Abbeel, Nathan Michael, and Vijay Kumar. Information-theoretic planning with trajectory optimization for dense 3D mapping. In *Robotics: Science and Systems (RSS) 2015*.

[20] Benjamin Charrow, Sikang Liu, Vijay Kumar, and Nathan Michael. Information-theoretic mapping using Cauchy-Schwarz quadratic mutual information. In *International Conference on Robotics and Automation (ICRA) 2015*.

[21] Chandra Chekuri, Nitish Korula, and Martin Pal. Improved algorithms for orienteering and related problems. *Transactions on Algorithms*, 8(3), 2012.

[22] Chandra Chekuri and Martin Pal. A recursive greedy algorithm for walks in directed graphs. In *Foundations of Computer Science (FOCS) 2005*.

[23] Shengyong Chen, Youfu Li, and Ngai Ming Kwok. Active vision in robotic systems: A survey of recent developments. *International Journal of Robotics Research*, 30(11), 2011.

[24] Sanjiban Choudhury, Ashish Kapoor, Gireeja Ranade, and Debadeepta Dey. Learning to gather information via imitation. In *International Conference on Robotics and Automation (ICRA) 2017*.

[25] Marc Christie, Patrick Olivier, and Jean-Marie Normand. Camera control in computer graphics. *Computer Graphics Forum*, 27(8), 2008.

[26] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. MeshLab: An open-source mesh processing tool. In *Sixth Eurographics Italian Chapter Conference 2008*.

[27] CloudCompare. CloudCompare. http://www.cloudcompare.com, 2017.

[28] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. MIT Press, 2009.

[29] Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. Efficient and flexible sampling with blue noise properties of triangular meshes. *Transaction on Visualization and Computer Graphics*, 18(6), 2012.

[30] Ian D. Cowling, Oleg A. Yakimenko, James F. Whidborne, and Alastair K. Cooke. A prototype of an autonomous controller for a quadrotor UAV. In *European Conference on Control (ECC) 2007*.

[31] Ola Dahl and Lars Nielsen. Torque-limited path following by on-line trajectory time scaling. *Transactions on Robotics and Automation*, 6(5), 1990.

[32] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured light fields. *Computer Graphics Forum (Proc. Eurographics 2012)*, 31(2, Part 1), 2012.

[33] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for UAVs in cluttered environments. In *International Conference on Robotics and Automation (ICRA) 2015*.

[34] Alexandre Devert. Spreading points on a disc and on a sphere. http://blog.marmakoide.org/?p=1, 2012.

[35] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83), 2016.

[36] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors, 2006.

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

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

[39] Duke Marine Robotics & Remote Sensing. Highlights from Antarctica research, 2017.

[40] Enrique Dunn and Jan-Michael Frahm. Next best view planning for active model improvement. In *Intelligent Robots and Systems (IROS) 2009*.

[41] Enrique Dunn, Jur van den Berg, and Jan-Michael Frahm. Developing visual sensing strategies through next best view planning. In *Intelligent Robots and Systems (IROS) 2009*.

[42] Epic Games. Infinity Blade: Grass Lands. http://www.unrealengine.com/marketplace/infinity-blade-plain-lands, 2017.

[43] Epic Games. Unreal Engine. http://www.unrealengine.com, 2017.

[44] Xinyi Fan, Linguang Zhang, Benedict Brown, and Szymon Rusinkiewicz. Automated view and path planning for scalable multi-object 3D scanning. *Transactions on Graphics (Proc. SIGGRAPH Asia 2016)*, 35(6), 2016.

[45] Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. *Transactions on Graphics (Proc. SIGGRAPH 2003)*, 22(3), 2003.

[46] Timm Faulwasser, Veit Hagenmeyer, and Rolf Findeisenc. Constrained reachability and trajectory generation for flat systems. *Automatica*, 50(4), 2014.

[47] Federal Aviation Administration. FAA aerospace forecasts (fiscal years 2017 – 2037), 2017.

[48] Simon Fuhrmann, Fabian Langguth, Nils Moehrle, Michael Waechter, and Michael Goesele. MVE–An image-based reconstruction environment. *Computer Graphics Forum*, 53(Part A), 2015.

[49] Yasutaka Furukawa and Carlos Hernandez. *Multi-View Stereo: A Tutorial*. Now Publishers, 2015.

[50] Enric Galceran and Marc Carreras. A survey of coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12), 2013.

[51] Ellen Gamerman. Drones invade Hollywood. Wall Street Journal, March 26 2015.

[52] C. Gebhardt, B. Hepp, T. Nageli, S. Stevsic, and O. Hilliges. Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals. In *CHI 2016*.

[53] T. Geijtenbeek and N. Pronost. Interactive character animation using simulated physics: A state-of-the-art review. *Computer Graphics Forum*, 31(8), 2012.

[54] Bernard Ghanem, Yuanhao Cao, and Peter Wonka. Designing camera networks by convex quadratic programming. *Computer Graphics Forum (Proc. Eurographics 2015)*, 34(2), 2015.

[55] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12(4), 2002.

[56] Ardeshir Goshtasby, Fuhua Cheng, and Brian A. Barsky. B-spline curves and surfaces viewed as digital filters. *Computer Vision, Graphics, and Image Processing*, 52(2), 1990.

[57] Brian Guenter and Richard Parent. Motion control: Computing the arc length of parametric curves. *Computer Graphics Applications*, 10(3), 1990.

[58] Aldy Gunawan, Hoong Chuin Laua, and Pieter Vansteenwegenb. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2), 2016.

[59] Gurobi. Gurobi Optimizer. http://www.gurobi.com, 2017.

[60] Lionel Heng, Alkis Gotovos, Andreas Krause, and Marc Pollefeys. Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. In *International Conference on Robotics and Automation (ICRA) 2015*.

[61] Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Autonomous visual mapping and exploration with a micro aerial vehicle. *Journal of Field Robotics*, 31(4), 2014.

[62] Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Real-time photorealistic 3D mapping for micro aerial vehicles. In *Intelligent Robots and Systems (IROS) 2011*.

[63] Geoffrey A. Hollinger, Brendan Englot, Franz S. Hover, Urbashi Mitra, and Gaurav S. Sukhatme. Active planning for underwater inspection and the benefit of adaptivity. *International Journal of Robotics Research*, 32(1), 2013.

[64] Christof Hoppe, Andreas Wendel, Stefanie Zollmann, Katrin Pirker, Arnold Irschara, Horst Bischof, and Stefan Kluckner. Photogrammetric camera network design for micro aerial vehicles. In *Computer Vision Winter Workshop 2012*.

[65] Alexander Hornung, Boyi Zeng, and Leif Kobbelt. Image selection for improved multi-view stereo. In *CVPR 2008*.

[66] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3), 2013.

[67] Wei-Hsien Hsu, Yubo Zhang, and Kwan-Liu Ma. A multi-criteria approach to camera motion design for volume data animation. *Transactions on Visualization and Computer Graphics (Proc. SciVis 2013)*, 19(12), 2013.

[68] Indivisible Movement. A drone's-eye view of corruption in Russia. Medium, March 3 2017.

[69] Intel. Intel drones light up the sky, 2018.

[70] Rishabh Iyer and Jeff Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *NIPS 2013*.

[71] Rishabh Iyer, Stefanie Jegelka, and Jeff Bilmes. Fast semidifferential-based submodular function optimization. In *ICML 2013*.

[72] Niels Joubert, Mike Roberts, Anh Truong, Floraine Berthouzoz, and Pat Hanrahan. An interactive tool for designing quadrotor camera shots. *Transactions on Graphics (Proc. SIGGRAPH Asia 2015)*, 34(6), 2015. doi:10.1145/2816795.2818106.

[73] Steven D. Katz. *Film Directing Shot by Shot*. Butterworth Publishers, 1991.

[74] Michael Kazhdan and Hugues Hoppe. Screened Poisson surface reconstruction. *Transactions on Graphics*, 32(3), 2013.

[75] Suseong Kim, Seungwon Choi, and H. Jin Kim. Aerial manipulation using a quadrotor with a two DOF robotic arm. In *Intelligent Robots and Systems (IROS) 2013*.

[76] Kris M. Kitani, Brian D. Ziebart, J. Andrew Bagnell, and Martial Hebert. Activity forecasting. In *ECCV 2012*.

[77] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *Transactions on Graphics (Proc. SIGGRAPH 2017)*, 36(4), 2017.

[78] Konstantin Kondak, Kai Krieger, Alin Albu-Schaeffer, Marc Schwarzbach, Maximilian Laiacker, Ivan Maza, Angel Rodriguez-Castano, and Anibal Ollero. Closed-loop behavior of an autonomous helicopter equipped with a robotic arm for aerial manipulation tasks. *International Journal of Advanced Robotic Systems*, 10(145), 2013.

[79] Johannes Kopf, Michael Cohen, and Rick Szeliski. First-person hyperlapse videos. *Transactions on Graphics (Proc. SIGGRAPH 2014)*, 33(4), 2014.

[80] Michael Krainin, Brian Curless, and Dieter Fox. Autonomous generation of complete 3D object models using next best view manipulation planning. In *International Conference on Robotics and Automation (ICRA) 2011*.

[81] Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2014.

[82] Vijay Kumar and Nathan Michael. Opportunities and challenges with autonomous micro aerial vehicles. *International Journal of Robotics Research*, 31(11), 2012.

[83] John Lasseter. Principles of traditional animation applied to 3D computer animation. In *SIGGRAPH 1987*.

[84] Adam N. Letchford, Saeideh D. Nasirib, and Dirk Oliver Theis. Compact formulations of the Steiner traveling salesman problem and related problems. *European Journal of Operational Research*, 228(1), 2013.

[85] Thomas Lipp and Stephen Boyd. Minimum-time speed optimisation over a fixed path. *International Journal of Control*, 87(6), 2014.

[86] Vincenzo Lippiello and Fabio Ruggiero. Exploiting redundancy in Cartesian impedance control of UAVs equipped with a robotic arm. In *Intelligent Robots and Systems (IROS) 2012*.

[87] Giuseppe Loianno, Justin Thomas, and Vijay Kumar. Cooperative localization and mapping of MAVs using RGB-D sensors. In *International Conference on Robotics and Automation (ICRA) 2015*.

[88] Joseph Mascelli. *The Five C's of Cinematography*. Silman-James Press, 1965.

[89] Massimo Mauro, Hayko Riemenschneider, Luc Van Gool, Alberto Signoroni, and Riccardo Leonardi. A unified framework for content-aware view selection and planning through view importance. In *BMVC 2014*.

[90] Massimo Mauro, Hayko Riemenschneider, Alberto Signoroni, Riccardo Leonardi, and Luc Van Gool. An integer linear programming model for view selection on overlapping camera clusters. In *3DV 2014*.

[91] Aaron Mavrinac and Xiang Chen. Modeling coverage in camera networks: A survey. *IJCV*, 101(1), 2013.

[92] J. McCann, N. S. Pollard, and S. Srinivasa. Physics-based motion retiming. In *SCA 2006*.

[93] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1–2), 2012.

[94] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *International Conference on Robotics and Automation (ICRA) 2011*.

[95] Nathan Michael, Shaojie Shen, Kartik Mohta, Yash Mulgaonkar, Vijay Kumar, Keiji Nagatani, Yoshito Okada, Seiga Kiribayashi, Kazuki Otake, Kazuya Yoshida, Kazunori Ohno, Eijiro Takeuchi, and Satoshi Tadokoro. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5), 2012.

[96] Christian Mostegel, Markus Rumpler, Friedrich Fraundorfer, and Horst Bischof. UAV-based autonomous image acquisition with multi-view stereo quality assurance by confidence prediction. In *CVPR Workshop on Computer Vision in Vehicle Technology 2016*.

[97] Tobias Nägeli, Lukas Meier, Alexander Domahidi, Javier Alonso-Mora, and Otmar Hilliges. Real-time planning for automated multi-view drone cinematography. *Transactions on Graphics (Proc. SIGGRAPH 2017)*, 36(4), 2017.

[98] Thomas Oskam, Robert W. Sumner, Nils Thuerey, and Markus Gross. Visibility transition planning for dynamic camera control. In *SCA 2009*.

[99] Rick Parent. *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann Publishers, 2007.

[100] Hyun Soo Park, Eakta Jain, and Yaser Sheikh. 3D social saliency from head-mounted cameras. In *NIPS 2012*.

[101] Pix4D. Projeto redentor white paper, 2015.

[102] Pix4D. Pix4Dcapture. http://pix4d.com/product/pix4dcapture, 2017.

[103] Pix4D. Pix4Dmapper Pro. http://pix4d.com/product/pix4dmapper-pro, 2017.

[104] Weichao Qiu and Alan Yuille. UnrealCV: Connecting computer vision to Unreal Engine. arXiv, 2016.

[105] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *International Symposium of Robotics Research (ISRR) 2013*.

[106] Mike Roberts, Debadeepta Dey, Anh Truong, Sudipta Sinha, Shital Shah, Ashish Kapoor, Pat Hanrahan, and Neel Joshi. Submodular trajectory optimization for aerial 3D scanning. In *ICCV 2017*.

[107] Mike Roberts and Pat Hanrahan. Generating dynamically feasible trajectories for quadrotor cameras. *Transactions on Graphics (Proc. SIGGRAPH 2016)*, 35(4), 2016. doi:10.1145/2897824.2925980.

[108] F. Ruggiero, M.A. Trujillo, R. Cano, H. Ascorbe, A. Viguria, C. Perez, V. Lippiello, A. Ollero, and B. Siciliano. A multilayer control for multirotor UAVs equipped with a servo robot arm. In *International Conference on Robotics and Automation (ICRA) 2015*.

[109] Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *Transactions on Graphics (Proc. SIGGRAPH 2004)*, 23(3), 2004.

[110] Korbinian Schmid, Heiko Hirschmuller, Andreas Domel, Iris Grixa, Michael Suppa, and Gerd Hirzinger. View planning for multi-view stereo 3D reconstruction using an autonomous multicopter. *Journal of Intelligent & Robotic Systems*, 65(1), 2012.

[111] William R. Scott, Gerhard Roth, and Jean-Francois Rivest. View planning for automated three-dimensional object reconstruction and inspection. *Computing Surveys*, 35(1), 2003.

[112] Adrian Secord, Jingwan Lu, Adam Finkelstein, Manish Singh, and Andrew Nealen. Perceptual models of viewpoint preference. *Transactions on Graphics*, 30(5), 2011.

[113] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous indoor 3D exploration with a micro-aerial vehicle. In *International Conference on Robotics and Automation (ICRA) 2012*.

[114] Kang G. Shin and Niel D. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *Transactions on Automatic Control*, 30(6), 1985.

[115] Amarjeet Singh, Andreas Krause, Carlos Guestrin, and William J. Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, 34(1), 2009.

[116] Amarjeet Singh, Andreas Krause, and William J. Kaiser. Nonmyopic adaptive informative path planning for multiple robots. In *International Joint Conference on Artifical Intelligence (IJCAI) 2009*.

[117] Skydio. The Skydio R1. http://www.skydio.com, 2018.

[118] J.-J. E. Slotine and H. S. Yang. Improving the efficiency of time-optimal path-following algorithms. *Transactions on Robotics and Automation*, 5(1), 1989.

[119] Olga Sorkine-Hornung and Michael Rabinovich. Least-squares rigid motion using SVD, 2017.

[120] Manohar Srikanth, Kavita Bala, and Frédo Durand. Computational rim illumination with aerial robots. In *Computational Aesthetics (CAe) 2014*.

[121] J. Sturm, E. Bylow, F. Kahl, and D. Cremers. Dense tracking and mapping with a quadrocopter. In *Unmanned Aeriel Vehicles in Geomatics 2013*.

[122] SymPy. SymPy: Python library for symbolic mathematics. http://www.sympy.org, 2014.

[123] Konstantinos A. Tarabanis, Peter K. Allen, and Roger Y. Tsai. A survey of sensor planning in computer vision. *Transactions on Robotics and Automation*, 11(1), 1995.

[124] Russ Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for MIT 6.832). http://undreactuated.mit.edu, 2016.

[125] Celine Teuliere, Laurent Eck, and Eric Marchand. Chasing a moving target from a flying UAV. In *Intelligent Robots and Systems (IROS) 2011*.

[126] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[127] U.S. Geological Survey. The universal transverse mercator (UTM) grid fact sheet 077-01, 2001.

[128] Wannes Van Loock, Goele Pipeleers, and Jan Swevers. Time-optimal quadrotor flight. In *European Control Conference (ECC) 2013*.

[129] Pieter Vansteenwegena, Wouter Souffriaua, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1), 2011.

[130] Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl. Time-optimal path tracking for robots: A convex optimization approach. *Transactions on Automatic Control*, 54(10), 2009.

[131] Lukas von Stumberg, Vladyslav Usenko, Jakob Engel, Jorg Stuckler, and Daniel Cremers. Autonomous exploration with a low-cost quadrocopter using semi-dense monocular SLAM. arXiv, 2016.

[132] Michael Waechter, Mate Beljan, Simon Fuhrmann, Nils Moehrle, Johannes Kopf, and Michael Goesele. Virtual rephotography: Novel view prediction error for 3D reconstruction. *Transactions on Graphics*, 36(1), 2017.

[133] Michael Waechter, Nils Moehrle, and Michael Goesele. Let there be color! Large-scale texturing of 3D reconstructions. In *ECCV 2014*.

[134] Pengpeng Wang, Ramesh Krishnamurti, and Kamal Gupta. View planning problem with combined view and traveling cost. In *International Conference on Robotics and Automation (ICRA) 2007*.

[135] Andreas Wendel, Michael Maurer, Gottfried Graber, Thomas Pock, and Horst Bischof. Dense reconstruction on-the-fly. In *CVPR 2012*.

[136] A. Witkins and M. Kass. Spacetime constraints. In *SIGGRAPH 1988*.

[137] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3D photography. *Transactions on Graphics (Proc. SIGGRAPH 2000)*, 35(1), 2000.

[138] Changchang Wu. Towards linear-time incremental structure from motion. In *3DV 2013*.

[139] Changchang Wu. SiftGPU: A GPU implementation of scale invaraint feature transform (SIFT). http://cs.unc.edu/~ccwu/siftgpu, 2007.

[140] Changchang Wu. VisualSFM: A visual structure from motion system. http://ccwu.me/vsfm, 2011.

[141] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M. Seitz. Multicore bundle adjustment. In *CVPR 2011*.

[142] Shihao Wu, Wei Sun, Pinxin Long, Hui Huang, Daniel Cohen-Or, Minglun Gong, Oliver Deussen, and Baoquan Chen. Quality-driven Poisson-guided autoscanning. *Transactions on Graphics (Proc. SIGGRAPH Asia 2014)*, 33(6), 2014.

[143] Hyunsoo Yang and Dongjun Lee. Dynamics and control of quadrotor with robotic manipulator. In *International Conference on Robotics and Automation (ICRA) 2014*.

[144] Cem Yuksel, Scott Schaefer, and John Keyser. Parameterization and applications of Catmull-Rom curves. *Computer Aided Design*, 43(7), 2011.

[145] Haifeng Zhang and Yevgeniy Vorobeychik. Submodular optimization with routing constraints. In *Conference on Artificial Intelligence (AAAI) 2016*.

[146] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *Transactions on Graphics (Proc. SIGGRAPH 2004)*, 23(3), 2004.