TOOLS TO FACILITATE AUTONOMOUS QUADROTOR CINEMATOGRAPHY

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

Niels Joubert
June 2017

Abstract

There is considerable interest in using orientable cameras mounted on unmanned quadrotor aircraft for cinematography. These quadrotor cameras can fly to unique vantage points and execute dynamic camera moves in 3D space. Unfortunately, traditional methods of using quadrotor cameras to capture video, inherited from methods for manually piloting remote-controlled aircraft, does not consider the needs of cinematographers and requires much skill and dexterity. In this thesis, we present an alternative approach. Our key insight is to reify concepts from traditional and virtual filmmaking into tools for cinematographers. These tools enable the user to express their cinematic intent directly, while we automate flying the quadrotor camera.

First, we build a tool for designing and autonomously executing quadrotor-based camera shots. Our tool enables the user to compose shots visually using keyframes, and precisely specify shot timing using easing curves. The user can preview their resulting shot in a virtual environment before flying, and automatically capture shots in the real world with a single button click using commercially available quadrotors.

Since some visual compositions of shots are more favored than others, we next present a tool that automatically computes static shots based on well-established visual composition principles and canonical shots from cinematography literature. Furthermore, our tool calculates feasible, safe, and visually pleasing transitions between shots using a novel real-time trajectory planning algorithm. Using our tool, the user can capture shots that follow cinematic conventions without spending cognitive effort on setting up individual compositions, or specifying how to move between compositions.

Our quadrotor camera must know where the subjects are with sufficient accuracy to faithfully capture these canonical shots of people. To that end, we present a platform for accurately localizing multiple objects. By using RTK GPS and IMU sensors, our platform

provides centimeter-accurate tracking and decimeter-accurate quadrotor control in a large-scale outdoor environment.

In combination, this work enables novices and experts alike to capture high-quality video footage using quadrotors.

Acknowledgments

My graduate school career would not be possible without the love and support of my family and my friends. My mother Lydia and my father Pierre instilled in me a love of science and engineering from my birth during a thunderstorm. I would not be here without their tireless and selfless investment into myself and my two incredible brothers, Dieter and Pierre-Henri. To them, and to the broader Joubert, Oosthuizen, and Du Toit families, thank you!

What gumption and skill I brought to my thesis work was infinitely magnified by my many advisors, peers, and students at Stanford and UC Berkeley. It's been my great fortune to work with you! It's an unfortunate limitation of the written medium that I must order people's names. Any perceived value judgment from this ordering is false! As a staunch egalitarian, I firmly hold that everyone's contribution brought unique and indispensable value to my work. That being said, special dispensation—and the honor of being named first—is given to my advisor, Pat Hanrahan, who gave me the wonderful gift of exploring so widely that I changed thesis topics three times.

My deepest thanks to my many mentors, advisors, colleagues, and co-authors: Mike Roberts and Jane E who deserve much credit for the success of this work, Prof. Stu Card, Prof. Maneesh Agrawala, Prof. Juan Alonso, Prof. Alex Aiken, Prof. Andrew Kalman, Prof. James O'Brien, Prof. Alex Filippenko, Dr. Zach DeVito, Dr. Eric Schkufza, Dr. Daniel Ritchie, Dr. Trent Lukaczyk, Dr. Dan Goldman, Dr. James Hegarty, Dr. Matthew Cong, Dr. Floraine Berthouzoz, Sebastian Burke, The Stanford UAV Club, Neil McGowan, Henry Chamberlain, Lona Antoniades, Annelie Starke.

Mentoring eight students over three summers was one of my most rewarding experiences at Stanford. Thank you to my students Anh Troung, Harrison Wray, Victoria Flores, Elias Wu, Noa Glaser, Katherine Phan, Stephanie Tang, and Jorge Lara-Garduno, for allowing me to learn the intricate and difficult skill of being an advisor to your most capable and wonderfully curious minds.

Academic work that includes a robotics component is notoriously challenging. It demands interacting with the messiness of reality outside the nicely-sandboxed virtual environments we prefer in computer science. It also demands extensive investment and expertise in hardware and software engineering to support core research problems. Our work would not have been possible without the active involvement of the following companies, foundations, and professionals: Fergus Noble, Colin Beighley, Timothy Harris and the Swift Navigation team, Dr. Andrew Tridgell, Randy MacKay, and the ArduPilot team, Dr. Brandon Basso, Dr. Dave Merrill and the 3D Robotics team, Derek Chung and the Canary Drones team, Andy Putch and the Freeskies team, and the Adobe Research group.

Thank you as well to the many professional and amateur cinematographers we interviewed, and who participated in our user studies: George Krieger, Joseph Picard, Brian Streem, Barry Blanchard, Jeff Foster, Russell Brown, Romeo Durscher, Blake Marvin.

In California, I found the proverbial pot of gold at the end of the rainbow in the multitude of incredible people I've befriended and experiences I've partaken in. I'm deeply grateful for all my friends who accompanied me on this journey of highest highs and lowest lows, Gleb Denisov, Ashley Brown, Dr. Eric Schkufza, Derek Chung, Marcello Bastea-Forte, Yizhuo Wang, Patrick Tierney, Trisha McNamara, Carrie Smith, Sher Chu, Jennifer Hsiaw, Madeline Fok, Liz Wheatley, and EECS House, you are all incredible. Many of the ideas in this thesis were born in the unlikeliest setting for a transformative gathering, the Black Rock Desert. Thank you to everyone in the extended families of The Dusty Connection and Shady Waffle, and the colorful citizens of Black Rock City, home of the Burning Man.

I'd like to single out those that provided me with professional mental health support. What little sanity I can still lay claim to is undoubtedly your fine work – Stanford's CAPS Center, Dr. Mark Abrahamson and the Mindfulness Based Stress Reduction program, and my therapist. Approximate one-third of Ph.D. students are at risk of developing psychiatric disorders, such as depression, and one in two experience psychological distress [51]. At one top-3 engineering school, 47 percent of Ph.D. students qualify as depressed, and 10 percent of graduate students have contemplated suicide [10]. It's concerning that, beyond my own wellness and development challenges, it appears we're provide inadequate support for the development of not just highly intelligent but also mentally healthy individuals, even at our top institutions. I'm heartened by the gradual incorporation of mindfulness, psychotherapy, and sports psychology into graduate programs, and hope this trend expands rapidly! If you're currently in the trenches and suffering, I implore you to seek professional counsel.

Lastly, I dedicate this thesis to someone who will sadly never have the chance to read these words - my grandfather, Oupa Nellis Oosthuizen.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Film as an art form is driven by technological innovation. The earliest films were made with a static camera from a fixed point of view, like a play. Over time it was discovered that splicing shots from different cameras together could fuse them perceptually and contextually, achieving new narrative and artistic effects. These techniques were on the one hand dependent on artistic insight—how an effect would contribute to a film's narrative, mood, and purpose—but also dependent on technological innovation which made new effects possible, or changed the skill and cost required by the filmmaking process. This thesis is about how a rapidly emerging technology, quadrotor cameras, can continue this process and enable significant further advances in cinematography.

Quadrotor cameras are digital cameras mounted on self-stabilized unmanned multirotor aircraft. In the thesis, we develop and implement approaches to three major problems blocking the development of this technology: (1) how to control quadrotor cameras for cinematography (2) how to accurately track objects in the real world and (3) how to encode cinematic knowledge into a quadrotor camera control system. The key result of this work is that by reifying concepts from virtual and traditional filmmaking into tools, we can enable users to express quadrotor camera shots at a high level, automate the role of the pilot, and enable a broader range of users to capture high-quality video footage. In short, we allow the user to drive the film, and the film to drive the quadrotor.

The traditional approach to interacting with quadrotor cameras is ill-suited for cinematography. Providing a set of joysticks and buttons—although suitable for maneuvering a quadrotor in real time—requires much skill and dexterity. The user must know how to fly a quadrotor and manipulate the attached camera. They must also know how to compose

film in a visually pleasing way. They must constantly translate from their desired visual composition to the quadrotor camera's controls. Since the quadrotor can only move in certain ways, they must also adapt their cinematic vision to its physical limitations. The combination of these challenges means that only a small set of highly-skilled artists can take advantage of the full capabilities of quadrotor cameras.

Say the user wants to film a tower, keeping it in the center of the frame as the camera circles clockwise around it. The user will have to execute a carefully coordinated series of joystick movements to fly the quadrotor to the left while rotating it counterclockwise. The user's movements must be precisely coordinated. If the user rotates too fast, the user won't be able to fly fast enough to keep the tower in view. If the user over or undercompensates, the tower will move away from the center of the frame and ruin the visual composition. If this isn't challenging enough, the user must also avoid crashing the quadrotor into objects in the environment, potentially destroying equipment or causing injury. Even expert quadrotor cinematographers would struggle to capture this visually simple shot.

In the first part of this thesis, we demonstrate an alternative approach to interacting with quadrotor cameras, rooted in well-established camera planning principles from 3D animation. We show how to use keyframes—a picture that encodes the visual composition of a shot at a representative point during that shot—along with a small set of additional concepts, to inform how to build a tool for designing and automatically executing quadrotor-based cinematography. Our tool enables users to (1) compose shots visually before flying, (2) preview the resulting shots in a virtual environment, (3) precisely control the timing of shots using easing curves and (4) capture the resulting shots in the real world with a single button click using commercially available quadrotors. To build our tool, we show how we can derive the real-time control inputs of a physical quadrotor camera model from a series of camera keyframes and easing curves. The result is that cinematographers can capture shots with a quadrotor camera without having to pilot the quadrotor aircraft.

For a quadrotor to capture accurate shots, it must know where it is in relation to its subjects and environment. In the second part of this thesis, we investigate applying a newly-available centimeter-accurate positioning approach to quadrotor localization. We quantitatively demonstrate centimeter-accurate tracking and decimeter-accurate control in large-scale real-world outdoor conditions. We use this approach, called RTK GPS, to build a testbed for localizing arbitrary objects, including people.

Unsurprisingly, some visual compositions of shots are more favored than others. In the last part of this thesis, we present a tool that guides quadrotor cameras using visual composition principles. Specifically, we focus on autonomously capturing video of one or two people. Rather than having the user compose a shot manually, our tool provides a small set of canonical shots to the user, such as the apex, external or internal shot. We show how to capture these shots of people autonomously, using a quadrotor camera and our accurate tracking testbed. We find that, unlike standard GPS-based systems, our tracking and control testbed is sufficiently accurate for capturing well-composed shots of people. Furthermore, we show how to move the quadrotor between these canonical shots in a way that's both visually pleasing and keeps the quadrotor from crashing into subjects. The result is that users can use elements from the language of film to control quadrotor cameras capturing video of people, without having to spend cognitive effort on setting up individual compositions or specifying how to move between these compositions.

Over the past few years, we've seen the rise of flying cameras. In combination, this work takes a step towards building what could be considered a self-flying camera. Just as a self-driving car has the potential to take its user to a destination without requiring they know how to drive, a self-flying camera gives the user the ability to capture footage without requiring they to know how to fly.

Chapter 2

Background

Quadrotors are vertical takeoff and landing aircraft, where lift is generated using four rotating airfoils. This particular aircraft design is experiencing massive popularity growth in the form of micro unmanned aerial vehicles. These MAVs, weighing anywhere from 50 g to 50 lb, can stably hover in place, execute aggressive flight paths, fly autonomously, and carry various sensors aloft. They've been the subject of interdisciplinary research and engineering efforts, enabled by recent advances in batteries, electric motors, sensors, and microprocessors. These efforts have been successful enough for quadrotor MAVs to appear as consumer and industrial devices, everything from toys to professional tools.

One area where quadrotor aircraft is being adopted, is cinematography. Quadrotors outfitted with orientable cameras are being used to film everything from Hollywood blockbusters to campy home videos. As a result, quadrotors are changing to suit the needs of cinematographers. For example, various quadrotors now support a dual-controller configuration with a high definition video downlink. One operator flies the aircraft. A second operator watches the real-time video feed and reorients the camera to compose a shot. This way, cinematographers can experiment and make decisions about the visual footage being captured, in real-time. More broadly, we believe that cinematic knowledge is ready to influence quadrotor camera interaction design directly.

In this chapter, we survey work done on quadrotor MAVs and work done on tools for virtual and real-world cinematography. We will examine how roboticists control and interact with quadrotors. We will examine how cinematographers and computer graphicists control and interact with cameras. We will set the stage to combine these fields into a set of tools for autonomous cinematography—the topic of this thesis.

2.1 The Origins of Quadrotor Aircraft

To understand the current state of quadrotors, how they are well-matched for cinematography, and how they fall short, it's useful to understand the historical development of quadrotors MAVs. Today's quadrotors can be traced back to early multirotor prototypes introduced in the 1930s. In fact, the first rotorcraft to achieve hover featured multiple lifting rotors. These early prototypes demonstrated the main aerodynamic properties of a lifting rotor: (1) the aircraft can be moved vertically by varying the amount of lift produced by the main rotor, (2) the aircraft can be steered by changing the orientation of the rotor, thus changing the direction of lift (3) the rotor produce a reactionary torque on the airframe which must somehow be balanced, and (4) a rotor moving sideways through an airflow experiences "dissymmetry of lift", where the forward blade produces more lift than the retracting blade. This dissymmetry will roll the aircraft, and must be counteracted.

With the development of manned helicopters, the aerodynamics of rotors was extensively studied throughout the 1900s, and detailed models of rotor aerodynamics are available in the literature [73, 50]. Multiple multirotor aircraft designs were investigated, including quadrotor configurations such as the Curtis X19-A from 1963 and the Bell X-22A from 1966. These aircraft are all built on the basic quadrotor configuration: four lifting rotors, arranged as two pairs spinning in opposite directions. This arrangement means the reactionary torque and dissymmetry of lift produced by one rotor is naturally balanced by its opposite pair. The craft can be accelerated in a direction by changing the total speed of all rotors. The craft can change the direction in which it is accelerated by rotating around its center of mass, which is accomplished by having one rotor speed up and the opposite rotor slow down. Since its rotors move in a plane fixed relative to the quadrotor body, this method of moving through an environment creates the effect of a quadrotor "leaning" into the direction of acceleration. For more details, we refer the reader to the accessible and comprehensive overview of quadrotor dynamics presented by Mahony et al. [58]

Unfortunately, early quadrotors suffered from two major flaws. First, they were mechanically complex with multiple combustion engines and gearboxes. This made them expensive to manufacture and difficult to maintain. Secondly, pilots found them extremely challenging to control, and pilot augmentation systems were not yet advance enough to stabilize these aircraft effectively. None of the efforts to build a manned quadrotor aircraft made it into production, and the development of vertical takeoff and landing aircraft focused on designs

with a single lifting rotor a small tail rotor. This design allows a single main rotor to spin at a constant speed - easy to drive using internal combustion engines. A complex mechanical system that allows cyclic and collective pitch adjustment of the rotor blade provides control to a pilot. For the time being, quadrotor development remained a theoretical possibility. Certainly, there were no commercially available micro quadrotors throughout most of the 20th century.

Arrival of Commercial MAV Quadrotors The development of quadrotor MAVs took place simultaneously in the hobby and research worlds. Remote control toy companies were introducing cheap and small components needed for miniature quadrotors: LiPo battery systems, high speed microprocessors, electronic speed controllers, and brushless motors. The hobby "maker-movement", building on the explosion of cheap microcontrollers and sensors brought about by the proliferation of smartphones, was making microcontroller development and robotics accessible and affordable. Simultaneously, the research world was investigating new modeling, control, and planning methods for aerial robots.

In 1989, the Japanese electronics manufacturer Keyence introduced a set of miniature mechanical gyroscopes, used to electrically measure the rate of rotation around an axis. Using two of these sensors as the basis for an analog control system, Keyence productized the first miniature quadrotor. Called the "Gyro Saucer", this remote controlled miniature quadrotor could hover for about a minute using Nickel Cadmium rechargeable batteries and brushed electric motors. This design was severely limited by low capacity batteries and imprecise gyroscopes and never took off commercially.

In 1999, Mike Dammar of Area Fifty-One Technologies introduced the first practical, commercially available quadrotor[1]. Dubbed the Roswell Flyer, it was sold as a build-it-yourself kit marketed to remote control toy enthusiasts. It relied on ceramic piezo-electric gyroscopes to measure its rotation rate, and a digital control system to auto-level the quadrotor. It furthermore used lithium-based rechargeable batteries and brushless motors to achieve a flight time of approximately 15 minutes [86]. Next, Area Fifty-One Technologies partnered with DraganFly, and produced the DraganFlyer series of quadrotors. In 2002, Dragan released the DraganFlyIII, which included an optional camera and video transmission system. For the first time, hobbyists could use micro quadrotors to capture aerial imagery. Several early quadrotor research projects were built on the DraganFlyer [37, 19].

[1]Personal Communication with Mike Dammar, February 2017

Since the introduction of the DraganFlyer series, quadrotor MAVs have massively proliferated. In 2006, DJI was founded, selling flight controllers for model helicopters and quadrotors. In 2007, the ArduPilot project was founded, building an open-source flight controller using the popular Arduino microcontroller platform. In 2009, ETH Zurich founded the PX4 project, providing a research-grade open-source autopilot. In the same year, 3D Robotics was founded, providing open-source and open-hardware quadrotors using the ArduPilot and PX4 projects. The fully-open nature of their quadrotors created an attractive platform for enthusiasts and researchers. In fact, the work presented in this thesis is built upon this platform.

At the same time, small, rugged, high-quality cameras were proliferating, sold by companies such as GoPro and Mobius. By mounting these cameras on quadrotor MAVs, photographers discovered a new type of low-altitude aerial photography, creating unique angles from new vantage points. To overcome the orientation changes of a quadrotor as it follows a trajectory, cameras are mounted on actively controlled gimbals. These gimbals actively reorient the camera by exploiting the same brushless motor control and MEMS sensor technologies that enable quadrotors, creating smooth and stable video footage regardless of the aggressive behavior of the quadrotor.

These technological advances paved the way for the proliferation of commercial quadrotor cameras. By 2015, DJI was selling upwards of half a million quadrotors with attached cameras a year, generating $1 billion in revenue, and valued at roughly $10 billion [72]. By 2016, the FAA required registration of all quadrotors MAVs. 300,000 MAVs were registered in the first 30 days of the program [1]. The camera-equipped quadrotor was becoming a mainstream tool for photographers and cinematographers.

When we look at the user interface provided by these quadrotor cameras, we notice an interesting phenomenon. These quadrotors still rely on the standard remote-control interface, appropriated from hobbyist radio control airplanes. This comes as no surprise, given the development history of quadrotors. This interface presents two 2-axis joysticks to the operator. These joysticks send real-time commands to the quadrotor and camera. The exact behavior of these joysticks is usually configurable. Commonly, these sticks command at least three different modes. In “rate mode”, joysticks command the rate of rotation around each axis of the quadrotor, and the onboard control system adjusts each propeller’s thrust accordingly. This mode requires only an onboard rotation rate sensor such as a gyroscope, and requires extreme user dexterity. In “orientation mode”, joysticks command

the absolute orientation of the craft relative to the horizon, and the onboard control system rotates the craft into the desired orientation. This requires an absolute orientation reference, normally supplied by an onboard accelerometer. In "position mode", joysticks command a change in position, and the onboard control system moves the quadrotor accordingly. This mode requires an absolute position reference, normally supplied by an onboard GPS.

This thesis claims that we can invent new tools to control quadrotor cameras by bringing together camera control concepts from the computer graphics community with autonomous control strategies and higher quality sensors from the robotics community.

2.2 Relevant Work on Quadrotor Planning and Control

Designing Trajectories for Quadrotors A trajectory is a function that describes a path through space over time. A trajectory is only useful if a physical quadrotor can execute this path, that is, there exists a sequence of motor speeds that, when executed, will move the quadrotor along this trajectory. For this to be the case, the trajectory must satisfy the physical equations of motion of the quadrotor. Trajectory planning algorithms solve the problem of finding a trajectory for a quadrotor given some high-level inputs. For a detailed discussion on the theory of trajectory planning, we refer the reader to the excellent course reader by Tedrake [81], and books by LaValle [49] and Karaman and Frazolli [42].

Mellinger and Kumar [63] and Richter et al. [74] introduced trajectory planning methods for quadrotors that finds a trajectory given a small set of waypoints. These methods make the observation that there exists a *reduced state space* in which all C^4 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. These approaches provide a theoretically sound foundation for higher level synthesis algorithms. Our methods for synthesizing quadrotor camera trajectories, presented in Section 3.3, extends this same observation to quadrotor cameras and adapts it to a camera planning space.

Current Quadrotor Trajectory Planning Tools Using these algorithms, the robotics and aeronautics communities have developed trajectory planning tools for autonomous quadrotors. The DJI Ground Station [26] and the APM Mission Planner [8] trajectory planning tools allow users to design quadrotor camera trajectories by placing waypoints

on a 2D map. The QGroundControl system [62] 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.

Designing Trajectories around Obstacles Obstacles appear as non-convex constraints in the otherwise convex reduced state space. This challenge is addressed in several different ways. Richter et al. used a sampling-based approach to first find a rough approximate path through an environment, then used an optimization approach to find a piecewise continuous polynomial path that satisfy continuity constraints [74]. Deits et al. preprocessed the free space into overlapping convex areas, then used a mixed-integer optimization approach to jointly find both the appropriate subset of free convex spaces and a smooth traversing trajectory [23]. Gebhardt et al. modeled obstacles as non-convex, spherical regions and relied on a sequential quadratic program to find a trajectory through the resulting non-convex space [32].

The previous algorithms were limited to offline usage, since runtimes are on the order of several seconds to minutes. Most recently, Allen and Pavone introduced state-of-the-art method that is fast enough to run in real time [7]. They relied on a combination of preprocessing, fast sample-based search, and trajectory smoothing.

Our method for finding trajectories around people, presented in Section 5.6, similarly plans smooth trajectories directly in a non-convex space by relying on a non-convex optimizer. Our approach takes advantage of our specific use case—filming two people—by planning trajectories in a unique reduced space, allowing us to find paths that also consider visual aesthetics along the trajectory, and produce a solution in under a second.

Accurate Position Sensing Autonomous control of quadrotors rely on locating the quadrotor in its environment. Commercial quadrotors rely on GPS to provide a position estimate, accurate to a few meters. The research community often localizes quadrotors to millimeter precision using visual tracking systems built for motion capture [64, 57]. Unfortunately, these motion capture systems rely on cameras in the environment, limiting their use case to relatively small areas outfitted in advance with an expensive array of sensors. Although several groups are investigating onboard visual and LIDAR-based localization [61, 11], so far the authors are unaware of any non-GPS-based absolute position

system for quadrotors that works reliably in arbitrarily large outdoor environments. Our work investigates using a new generation of centimeter-accurate low-cost RTK GPS sensors for quadrotor localization outdoors.

Tracking Objects using Quadrotors Computer vision algorithms and feedback control policies have been developed to track moving target objects using quadrotors equipped with cameras [83, 66, 53, 21]. However, existing approaches *react* to moving target objects by optimizing the *position* of the quadrotor. In contrast, we globally optimize the *trajectory* of the quadrotor.

An alternative approach is placing sensors on subjects, relieving the quadrotor from maintaining a visual line of sight to all subjects. Inspired by work using centimeter-accurate RTK GPS [82] to study human gait, we use RTK GPS combined with Inertial Measurement Unit (IMU) sensors to track subjects and demonstrate its efficacy for automating cinematography.

Physically Safer Quadrotors Several companies are developing safer quadrotor hardware, such as the Parrot AR, Hover Camera and Flyability Gimball. These quadrotors reduce the potential harm of a collision by enclosing the propellers inside a safety mesh or shell. Recently, the first consumer quadrotors with active obstacle avoidance became available. The DJI Phantom 4 and Yuneec H both attempts to detect an obstacle and take evasive action. However, these systems do not attempt to produce visually pleasing cinematography while avoiding obstacles. We see these approaches as important safety fallbacks for handling unexpected or unavoidable collisions, complementary to our subject-aware trajectory planner.

Quadrotors Equipped with Robotic Arms Quadrotors are being equipped with more than just cameras. There is a growing literature describing physical models for quadrotors equipped with robotic arms [56, 45, 87, 78]. 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.

Quadrotors Applied to Other Computer Graphics Problems Quadrotors have very recently been applied to computer graphics problems beyond cinematography. Srikanth et al. introduced a feedback control policy for maneuvering a quadrotor with a non-orientable light attached to it [79]. 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.

2.3 Relevant Work on Cinematography and Camera Control

Principles of Cinematography Filmmakers have extensively studied visual composition of cinematic shots [59, 43, 14]. These artistic professionals observed that some compositions are more visually pleasing than others. Starting from this observation, filmmakers have deduced a language of film, including canonical shots well-suited to a wide range of scenarios. These efforts culminated in the seminal work of Arijon, "The Grammar of Film Language", which systematically describes the various factors involved in composing a cinematic shot depending on scenario [9]. This body of work has proved to be an invaluable guide for the technical disciplines in their attempts to design tools for cinematographers.

Designing Trajectories for Virtual Cameras using 3D Animation Techniques Designing trajectories for virtual cameras is a classic problem in 3D animation, comprehensively discussed in the excellent survey paper by Christie et al. [20]. A wide variety of methods exist that take a series of input *keyframes* and *easing curves* and produce a camera trajectory. These methods enable a powerful interface for camera path planning, with users sparsely specifying high-level visual keyframes and receiving a continuous camera trajectory in return. We build on these principles in designing our tool for guiding quadrotor cameras using 3D animation principles, presented in Chapter 3. Especially relevant to this thesis is the work by Oskam et al. [70] and Hsu et al. [38]. Both these methods 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 [70], or by solving a continuous optimization problem [38].

These existing methods for synthesizing virtual camera trajectories guarantee C^1 or C^2 continuity. However, we demonstrate in Section 3.3.4 that camera trajectories must be C^4 continuous in order to obey the physical equations of motion for quadrotors. With this requirement in mind, all our tools synthesizes C^4 camera trajectories. We achieve this level of smoothness using both discrete and continuous optimization techniques.

Autonomous Cinematography in Virtual Environments using Composition Principles Computer graphicists have also invented methods to automatically find visually pleasing compositions and trajectories. The computer gaming industry have developed a deep knowledge of fully-automatic real time camera control [35]. Furthermore, interests in virtual worlds have spurned automatic real-time camera control algorithms [20]. A common approach is to find camera poses based on the aforementioned principles of cinematography [36, 22, 52].

Especially relevant to our work is the seminal paper by He et al. [36]. This paper present a set of heuristics to pose virtual cameras based on visual composition principles, and use these heuristics to design "The Virtual Cinematographer". In their system, visually pleasing camera poses are encoded as a set of *camera modules*. Each camera module encodes a shot type from cinematography literature, and generates a camera pose that follows composition heuristics. In Chapter 5, we extend their method to quadrotors. In doing so, we satisfy the physical constraints imposed by quadrotors, track objects in the real world, and consider safety of subjects, all while maintaining the same high-level visual composition principles.

Finding Transitioning Trajectories between Camera Poses Our tool for guiding quadrotor cameras using visual composition principles, presented in Chapter 5, has to find trajectories that automatically transition our quadrotor camera between canonical camera poses. Fortunately, interpolating between multiple camera poses in a visually pleasing manner is a well-studied subproblem of autonomous cinematography. Recently, Lino and Christie [54] demonstrated a fast analytic method for interpolating between viewpoints of subjects in a way that produces visually pleasing results. Their main insight was to define a visual interpolation space relative to each subject, and analytically compute a resulting camera path. They show how to solve for a camera position given two screen-space positions and a distance to the closest subject, also known as the Blinn spacecraft problem [13]. Lino

and Christie's approach has been used to generate smooth trajectories using an iterative approximation approach [29], and as a target for the Prose Storyboard Language [31]. Their approach has also been extended to force-based camera models with soft constraints [30]. We build on Lino and Christie's visual interpolation space to design a new method that also produces visually pleasing results, while we specifically respect the safety of *both* subjects and the requirements of quadrotor hardware.

2.4 Examples of Current Robotic Cinematography Systems

New Quadrotor Flight Modes for Cinematographers The quadrotor community is actively inventing new flight modes that raise the level of abstraction presented to operators. Rather than having joysticks simply change the position of the quadrotor, the 3D Robotics Solo [4] and DJI Go [25] systems change the behavior of the operator's joysticks in such a way as to aid cinematography. These systems allow users to control the orientation of the camera and speed of the quadrotor as it flies between pre-defined waypoints [4], or in a circular orbit around a point of interest [25]. Whereas these systems can be used to *modify* shots as they are being executed, our systems 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 systems can be used to design shots where there is no particular target object. Fundamentally, these flight modes still rely on an operator to fly the vehicle and make continuous changes to capture a shot, while our tools alleviate the user from having to pilot a quadrotor.

Autonomous "Follow-me" Quadrotors There is active interest in designing quadrotor camera control systems that can fully autonomously capture video of subjects. Naseer et al. demonstrated a quadrotor that follows a person using RGB-D depth tracking [66]. The 3DR Solo, DJI Phantom, Yuneec Typhoon, AirDog, and Ghost Drone all feature a "follow-me" mode that tracks subjects. These follow-me approaches have much in common with pilot aids such as position-hold mode. They only attempt to keep the subject visible, and still rely on the operator to compose the shot or smoothly move between shots. In contrast, our approach uses high-level cinematography principles to *automatically* find visually pleasing poses and present interfaces to plan cinematography before flight.

Tools for Cameras Attached to Robotic Arms Tools for designing camera trajectories that support cameras mounted on robotic arms have been developed. Most relevant to this thesis is the work by Bot & Dolly. They incorporated trajectory planning for robot arms into the popular Maya 3D animation package, leveraging the cinematography-oriented features of a 3D animation tool and the physical abilities of industrial robot arms[2]. Although we draw inspiration from their work, these tools are not directly applicable to quadrotor cameras. We enable cinematography-oriented features in a tool for quadrotor cameras, and thus more effectively assist quadrotor cinematographers.

Autonomous Cinematography using Wheeled and Stationary Robotic Cameras More broadly, guiding robotic cameras using visual composition principles has been investigated in the robotics literature. However, this work mostly focuses on wheeled or stationary robots [17, 6, 44, 28]. In contrast, our system explicitly considers the dynamics of quadrotors when planning shots. Similar to our technique, some of this work crops the resulting footage to improve visual composition when the robot cannot place itself in the desired pose [18].

[2]Bot & Dolly is now defunct, and the specifications for the IRIS camera control system are no longer publicly available. The details of their implementation comes from personal correspondence with Joe Picard, the Director of Photography of Bot & Dolly

Chapter 3

Guiding Quadrotors with 3D Animation Primitives

This chapter introduces an interactive tool for designing quadrotor camera shots, designed around well-established 3D animation primitives. We call our tool "Horus". Horus assists users in designing shots before capturing with a quadrotor, and assumes full control of the quadrotor during capture. In doing so, our tool enables novices and experts to capture high-quality aerial footage without requiring any piloting skill.

The contents of this chapter was adapted from a 2015 SIGGRAPH Asia conference paper co-authored by Niels Joubert, Mike Roberts, Anh Troung, Floraine Berthouzoz and Pat Hanrahan. All uses of "we", "our", and "us" in this chapter refer to all listed authors.

3.1 Approach

Typically, users control quadrotors with hand-held joysticks, which requires significant physical 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

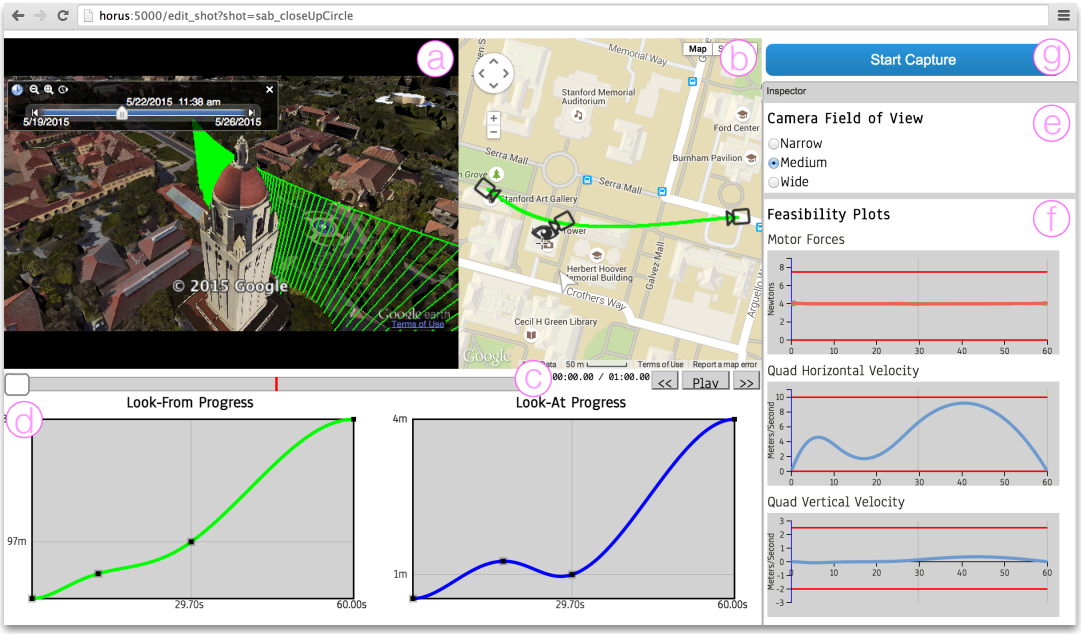

Figure 3.1: Our interactive tool for designing quadrotor camera shots, Horus. In Horus, users specify camera pose keyframes in a virtual environment using a 3D scene view (a) and a 2D map view (b). Horus synthesizes a camera trajectory that obeys the physical equations of motion for quadrotors, and interpolates 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). Horus provides the user with visual feedback about the physical feasibility of the resulting trajectory, notifying the user if their intended trajectory violates the physical limits of their quadrotor hardware (f). Once the user is satisfied with their shot, they presses the Start Capture button (g).

to deviate significantly from the intended trajectory, or even crash.

To inform the design of Horus, we conducted formative interviews with professional quadrotor photographers and cinematographers, 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 3.1) instantiates these principles by (1) allowing users to specify shots visually in a realistic 3D Google Earth environment; (2) providing a virtual preview of the entire shot; (3) providing users with precise control over the timing of the shot; and (4) notifying users if their intended

Figure 3.2: Horus commands a quadrotor camera to execute the user's 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

shot violates the physical limits of their quadrotor hardware. Finally, Horus commands a quadrotor camera to execute the user's trajectory fully autonomously, capturing real video footage that is faithful to the virtual preview (see Figure 3.2) 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 Horus, 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 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 Horus to provide the user with visually accurate shot previews, and visual feedback about the physical feasibility of camera trajectories.

We use our tool Horus to generate a variety of quadrotor camera shots. We evaluate Horus 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 Horus. Novices stated that Horus would empower them to shoot high-quality aerial footage, a skill otherwise inaccessible to them, and experts stated that Horus would improve and extend their existing workflow.

3.2 Designing A Shot Planning Interface

3.2.1 Design Principles

In order to design more effective tools for quadrotor camera control, we began by analyzing manuals on cinematography [59, 9, 43], 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, 48]. 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 [59]. 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.

3.2.2 User Interface

We reify the design principles described in Section 3.2.1 into an interactive tool for planning and capturing quadrotor camera shots (Figure 3.1). In Horus, the user specifies camera pose keyframes at specific times in a virtual environment. Our tool synthesizes a camera

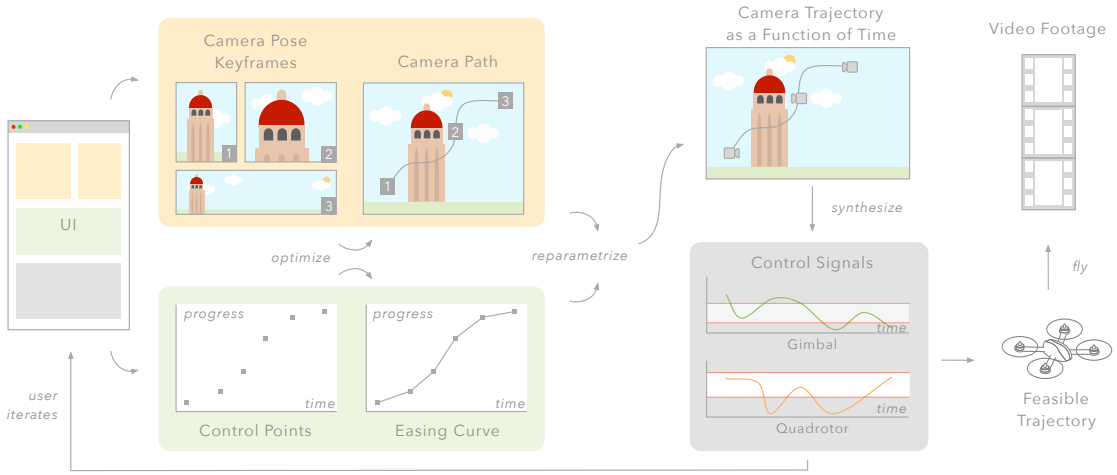

Figure 3.3: Overview of the major technical components of our system. We begin with two user-specified inputs: (1) camera pose keyframes in a virtual environment (e.g., Google Earth); and (2) a sequence of easing curve control points which specify how the camera progress over time between keyframes. 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-parametrize 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.

trajectory that obeys the physical equations of motion for quadrotors, and interpolates between the user-specified keyframes.

In Horus, 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 3.1a). The user can set keyframes in this view by moving the virtual camera using a trackball interface. This interface enables the user

to design shots visually. Our tool also provides a 2D map view of the scene using Google Maps (Figure 3.1b). The user can set keyframes in this view by dragging look-from and look-at markers around the 2D map.

The user can add, edit, and delete keyframes using the 3D scene view and the 2D map view. These views are linked: edits in one view instantly update the other view. Whenever a keyframe is added, edited, or deleted, Horus 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 their shot, and navigate through time using a scrubber interface (Figure 3.1c). To set a keyframe at a specific time, the user scrubs to that moment in time and edits the camera pose, as described above. When the user clicks the *Play* button or moves the scrubber, Horus 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 their shot, and supports rapid iteration.

The user can edit distinct easing curves for look-at and look-from position trajectories (Figure 3.1d). The user can add, edit, and delete control points on these easing curves. Editing these easing curves enables the user to precisely control the timing control of their 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 Horus that exceed the physical limits of their quadrotor hardware. For example, the user might specify two keyframes so close together in time, but so far apart in space, that their quadrotor cannot fly fast enough to capture the shot. Our tool provides the user with visual feedback about the physical feasibility of their trajectory, notifying the user if their intended trajectory violates the physical limits of their hardware.

Every time the user edits their shot, Horus 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 3.1f). In each plot, Horus shows the physical limits of the quantity with two horizontal red lines. If any dynamic or kinematic quantity exceeds these physical limits, Horus highlights the corresponding feasibility plot. Our tool also highlights any infeasible regions directly in

the 3D scene view (Figure 3.1a), the 2D map view (Figure 3.1b), and on the easing curves (Figure 3.1d). In each of these views, Horus 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 their shot to the physical limits of their hardware.

Capturing Real Video Footage At any time during the design process, the user can save their shot. Once the user is pleased with their shot, they can take a laptop running Horus, and their quadrotor, to the approximate real-world starting location of their shot. The user can initiate an automatic capture session by clicking the *Start Capture* button (Figure 3.1g). Once the user clicks this button, Horus commands a quadrotor camera to execute the user-specified shot fully autonomously, capturing real video footage.

3.3 Generating Feasible Trajectories for Quadrotor Cameras

3.3.1 Technical Overview

We provide an overview of the major technical components of our system in Figure 3.3. 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 3.3.2). 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 3.3.3). 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 3.3.4). This algorithm enables Horus 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 3.2.2, our strategy for handling these physically infeasible trajectories is interactive.

Once the user is satisfied with their camera trajectory, we command a quadrotor camera to execute the trajectory fully autonomously, capturing real video footage (Section 3.3.5). At a high level, we use the camera trajectory computed in Section 3.3.3 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 3.3.3, without going to the extra trouble of computing control signals in Section 3.3.4. However, computing control signals enables Horus to provide visual feedback about the physical feasibility of trajectories, which is an important safety feature. Moreover, computing control signals enables Horus to *certify* the accuracy of visual shot previews, since the visual preview will be accurate only if the trajectory is physically feasible.

3.3.2 A Quadrotor Camera Model

In this section, we introduce our physical quadrotor camera model, in which a rigid body quadrotor is attached to a camera mounted on a gimbal. We model the gimbal as a ball-and-socket joint that is rigidly attached to the quadrotor's center of mass. We provide an overview of our model in Figure 3.4.

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.

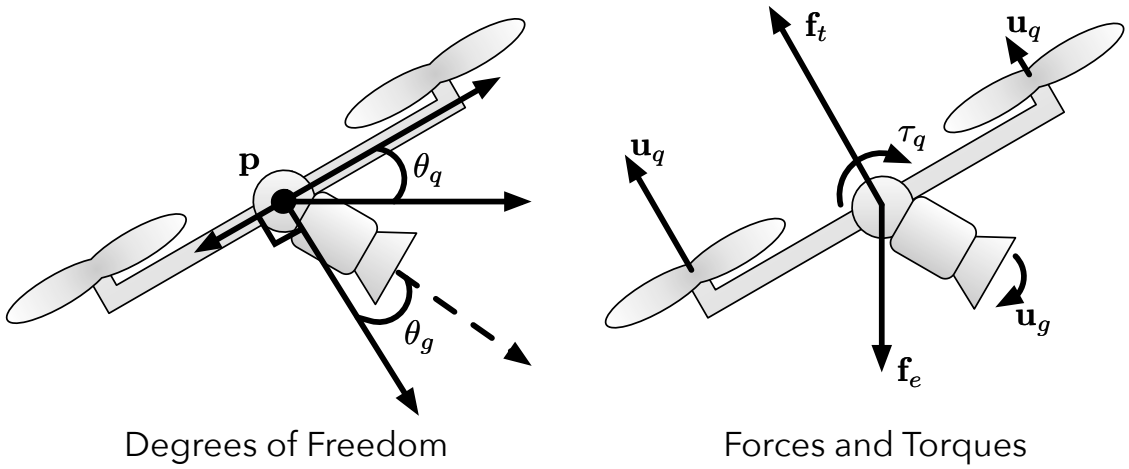

Figure 3.4: Overview of our quadrotor camera model, shown in 2D for simplicity. Degrees of freedom: We model the physical state of a quadrotor camera with the following degrees of freedom: the position of the quadrotor in the world frame, $\mathbf{p}$; the orientation of the quadrotor in the world frame, 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. Forces and torques: We maneuver the quadrotor by applying thrust control at the propellers, $\mathbf{u}_q$. 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.

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,

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

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 state constraints and actuator limit constraints of our system. This equation fully determines the evolution of our quadrotor camera model over time. Tedrake [81] 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 [63] to include a fully actuated 3 degree-of-freedom gimbal. We include a concise definition and a more detailed derivation for these matrices in Appendix A.

3.3.3 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 3.3.4.

We follow the standard practice in computer graphics [71] 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 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 [63].

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,

$$\begin{aligned} \text{subject to} \quad & p_i(0) = v_i & & p_i(1) = v_{i+1} \\ & \frac{d^j}{d\bar{u}_i^j} p_i(0) = w_i^j \mathbf{d}_{i,j} & & \frac{d^j}{d\bar{u}_i^j} p_i(1) = w_i^j \mathbf{d}_{i+1,j} \end{aligned} \tag{3.2}$$

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.

We can express the optimization problem in equation (3.2) as a constrained quadratic minimization problem as follows,

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{x}^T\mathbf{Q}\mathbf{x} \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b} \tag{3.3}$$

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 (3.2); $\mathbf{A}$ is the matrix and $\mathbf{b}$ is the vector that can be obtained by expressing the equality constraints from equation (3.2) in matrix form. The problem in equation (3.3) can be solved by solving the following linear system,

$$\begin{bmatrix} 2\mathbf{Q} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix} \tag{3.4}$$

where λ is the Lagrange multiplier variable obtained by transforming equation (3.3) into unconstrained form [15].

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* [88] 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 [12].

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 [34].

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.

3.3.4 Synthesizing State Space Trajectories and Control Trajectories

In this section, we consider the problem of synthesizing a *state space trajectory* and corresponding *control trajectory* that will command our quadrotor and gimbal to follow a given *virtual camera trajectory* in the world frame. At a high level, our approach is to compute a trajectory through our quadrotor camera's state space, that places the gimbal at the same world frame pose as the virtual camera we are trying to follow at all times. We then substitute this *state space trajectory* into equation (3.1) 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 (3.1). Or stated more precisely, that the state space

trajectory we compute in this section, when substituted into equation (3.1), 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 [63]. 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 [24]. 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.

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 (3.1). Or stated more precisely, that the state space trajectory we compute in Section 3.3.4, when substituted into equation (3.1), always yields a left hand side that is in the column space of the matrix $\mathbf{B}$. In our algorithm, the quadrotor's local $\mathbf{y}$ axis, in combination with the virtual camera's local $\mathbf{x}$ axis, uniquely determines the orientation of the quadrotor (lines 4–6).

Computing a Control Trajectory In this subsection, we compute a control trajectory $\mathbf{u}_{0:T}$, as a function of our state space trajectory $\mathbf{q}_{0:T}$. We begin by computing the 1st and 2nd derivatives of our state space trajectory, $\dot{\mathbf{q}}_{0:T}$ and $\ddot{\mathbf{q}}_{0:T}$ respectively, using finite differences. We compute our control trajectory by repeatedly substituting $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ into equation (3.1), 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 (3.1), 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 (3.1) is always in the column space of $\mathbf{B}$, and $\mathbf{B}$ is always full column rank. We include a proof that $\mathbf{B}$ is always full column rank in the Appendix A

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 (3.1) 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 [63].

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 (3.1). 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 (3.1). 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 3.2.2.

3.3.5 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 3.5. We build our real-time control system on top of the open source ArduPilot autopilot software [8]. 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 [47]. The ArduPilot feedback controller takes as input

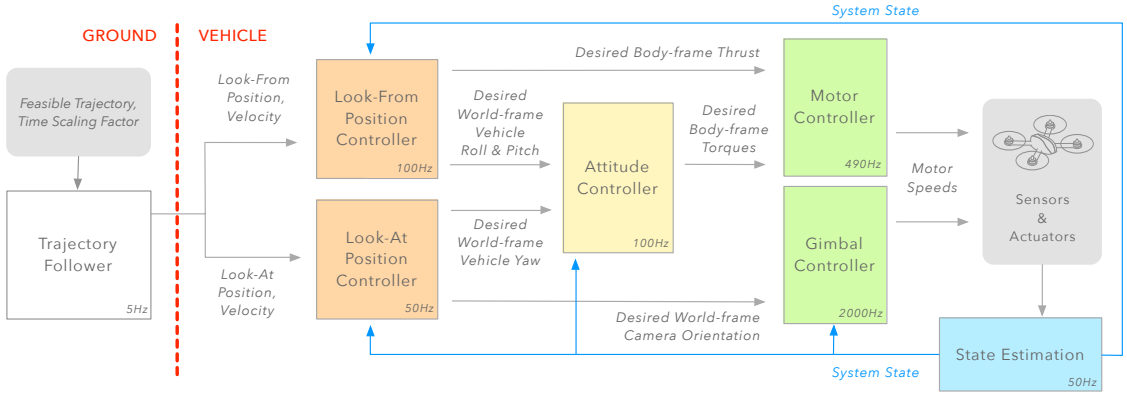

Figure 3.5: Block diagram of our real-time control system for executing camera trajectories. On a ground station (left), our trajectory follower (white) samples the camera trajectory, transmitting the sampled position and velocity of look-at and look-from points to the quadrotor. Our trajectory follower 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 [47].

the position and velocity of look-at and look-from points along a camera trajectory. Our real-time control system runs on a ground station. Our system executes the user's intended camera trajectory by sampling the position and velocity of look-at and look-from points along the trajectory, and transmitting these quantities to the quadrotor.

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

Hardware Platform Our hardware platform consists of an 3D Robotics IRIS+ quadrotor [2] running the open source ArduPilot autopilot software [8] on a Pixhawk autopilot computer [62]. We equip our quadrotor with a 2-axis gimbal and a GoPro Hero 4 Black

camera. At the time of writing, this hardware setup is priced at $1300, and is representative of an entry-level quadrotor for aerial cinematography.

System Identification We determined the system parameters used in our quadrotor camera model, which are specific to our hardware, partially through direct measurement and partially through published engineering specifications. We used a dynamometer to measure the maximum force and torque our rotors could generate, and estimated the moment of inertia from the quadrotor's mass and shape. We used the maximum lean angles, maximum velocities, and maximum accelerations published by the ArduPilot community [8].

3.4 Evaluation and Discussion

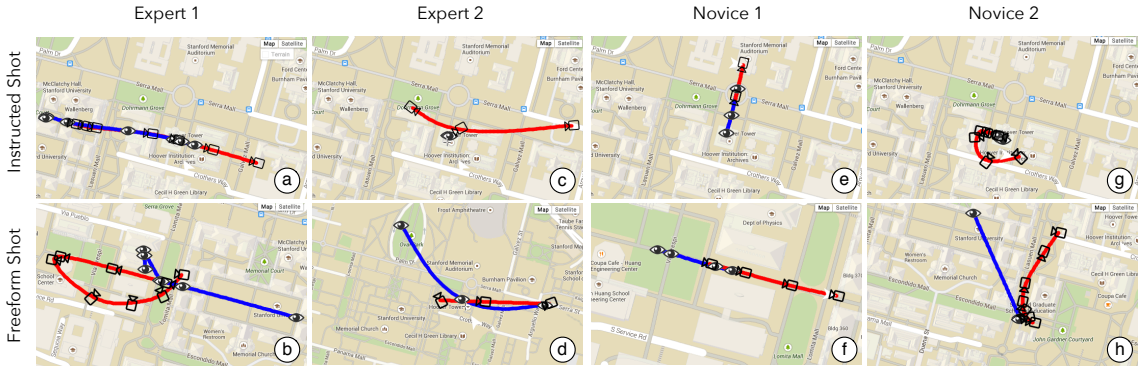

Figure 3.6: Camera shots created by the two experts (left) and two novices (right) in our user study. The look-from and look-at trajectories for each shot are shown in red and blue respectively. The shots created by our participants contain a wide variety of camera motions.

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

User Study We performed a user study aiming to understand whether our tool Horus 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 Horus 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 Horus. 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 3.6, 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 Horus. We were able to successfully capture all eight shots true to the virtual previews from Horus.

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 3.7). 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,

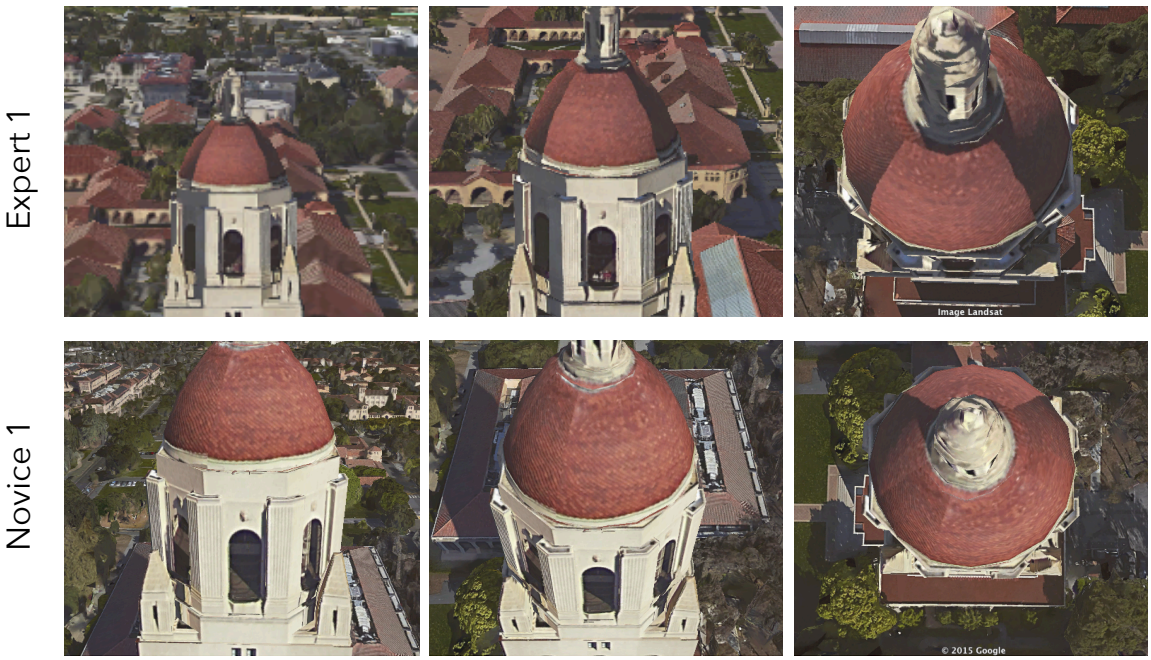

Figure 3.7: Novices and experts successfully designed shots with challenging camera motions using Horus. The expert shot (top) is especially challenging to execute manually, since it requires smoothly changing the camera orientation to look down at Hoover Tower exactly as the quadrotor flies over it. The novice shot (bottom) contains a similar camera motion.

since small initial errors in the direction of flight have to be corrected, leading to visual artifacts in the resulting video.

- Expert shot (a) required smoothly adjusting the rate of camera re-orientation, to end at a specific orientation at a specific time. We found this camera motion in novice shot (a). We show these two shots in Figure 3.7. 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 Horus, regardless of their level of expertise with quadrotors.

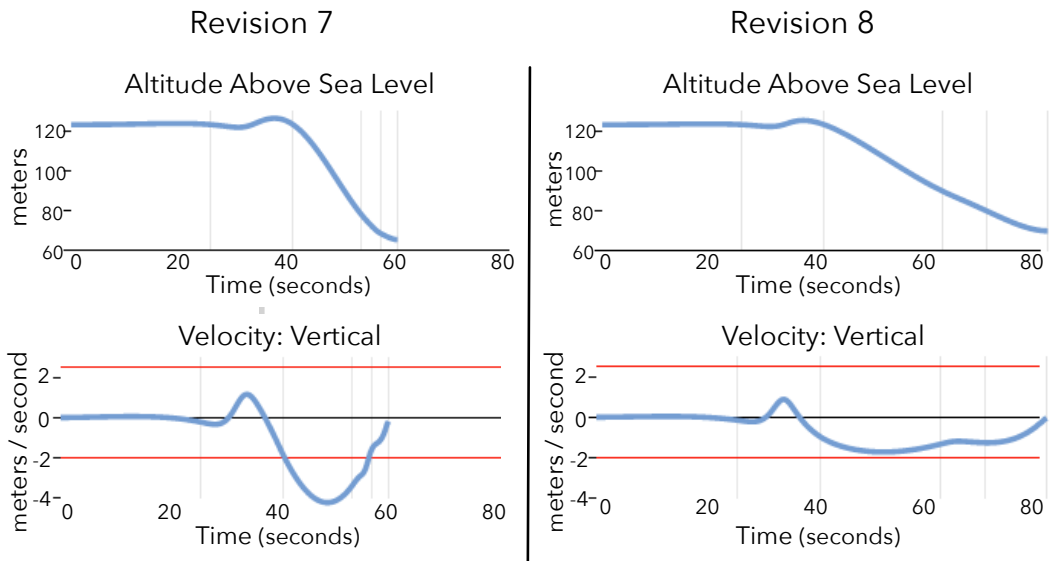

Figure 3.8: Participants were able to modify infeasible shots into feasible shots using the visual feedback we provide in Horus. After his 7th revision, Expert 1 found that his shot was infeasible (left). He edited both the altitude and timing of 3 keyframes to create a feasible shot as his 8th revision (right). Horizontal red lines indicate physical limits of our quadrotor hardware.

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 3.1 shows that our visual preview accurately estimates the appearance of recorded video footage. This finding validates our approach of enabling users to design shots visually, and highlights the importance of ensuring physical feasibility during the design process.

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

Visual Feasibility Feedback was Useful, Although Some Participants Would Have Preferred an Automatic Solution All of our participants responded to the visual feasibility feedback in Horus. Users successfully modified their shots until they were within the physical limits of our hardware, as shown on the feasibility plots in Horus. Figure 3.8 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.

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 3.9). 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.''*).

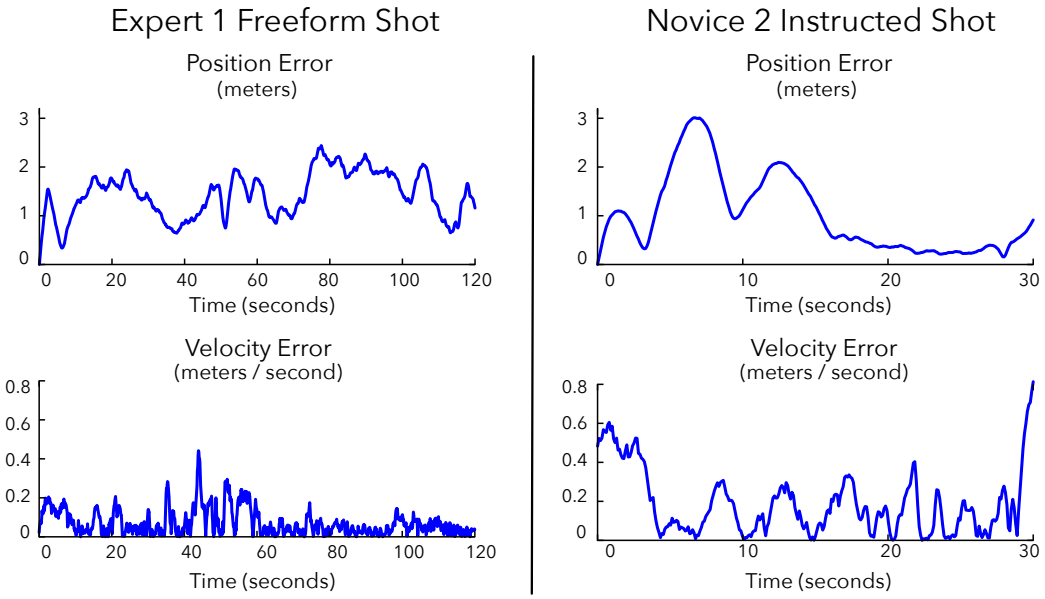

Figure 3.9: Position and velocity error of our quadrotor for the longest (left) and shortest (right) shots. The position error is less than 3.01 m at all times, and the velocity error is less than 0.80 m/s at all times. Note that the horizontal scaling varies varies on the left and right subplots.

Chapter 4

Evaluating RTK GPS for Quadrotor Localization

In the previous chapter, we were limited to filming large structures largely due to the relative inaccuracy of automatic quadrotor positioning. We would like to autonomously capture smaller objects at closer range, such as people—which happen to be the subject of the next chapter. We choose to approach this problem by investigating a high accuracy localization sensor. Specifically, we investigate the precision and accuracy of Real-Time Kinematic GPS. Real-Time Kinematic or RTK GPS is a signal processing technique that can estimate positions of GPS receivers to within 2 cm accuracy, which is much better than the roughly 2 m accuracy of conventional GPS.

In our work, we rely on RTK GPS to enable repeatable autonomous cinematography, but accurate localization has further-reaching implications: improving the quality of mapping and surveying, enabling autonomous navigation, and aiding collision avoidance. In this chapter, we examine the positioning performance of these newly-available single-frequency RTK GPS receivers to provide high-accuracy positioning for a quadrotor MAV. Given the arrival of cheap and compact single-frequency RTK GPS receivers, we feel an analysis of their performance in the context of micro aerial vehicles is a timely contribution to the broader robotics field.[1]

[1]We further analyze the impact of using RTK GPS as part of a localization suite for filming subjects in the upcoming chapter, section 5.8

Figure 4.1: Our modified 3D Robotics Iris Quadrotor. A second platform on top of the quadrotor body holds a RTK GPS antenna and a conventional GPS module. This platform provides electromagnetic shielding between the sensitive GPS antennas and the rest of the quadrotor, as well as a clear sky-view during flight. An Intel Galileo companion computer is mounted on the underside of the body. The companion computer serves as an independent datalogger.

4.1 Approach

Real-time kinematic (RTK) GPS exploits information about the GPS carrier wave to improve the accuracy of position estimates. RTK GPS compares the phase of the received carrier wave between two GPS receivers to derive an accurate vector between a stationary "base station" GPS and moving "rover" GPS. Note the requirement for two GPS receivers, in comparison to conventional GPS positioning that requires only a single receiver. This approach can provide centimeter-accurate positioning with millisecond latencies. However, RTK GPS systems tend to be expensive and bulky. Furthermore, RTK GPS can only calculate centimeter-accurate positions once an integer ambiguity is resolved, and ambiguity resolution algorithms have runtimes on the order of minutes for single-frequency

Figure 4.2: Our base station cart, from where we issue commands and monitor the system. It contains a Piksi RTK GPS connected to a NovaTel pinwheel antenna, a laptop computer, a telemetry radio for data communication, and a remote control radio for joystick control inputs.

receivers [80, 84]. That being said, single-frequency receivers are desirable since they can be built using off-the-shelf GPS components that have benefited from the economies of scale from integration in mobile phones and vehicle navigation systems. For a full treatment of the RTK GPS technique, we refer the reader to the excellent book by Kaplan and Hegarty [41].

In this work we examine Swift Navigations Piksi RTK GPS receiver [80]. We choose to focus on the Piksi GPS since it is the first, and at the time of this writing, only commercially-available, affordable RTK GPS system designed for micro aerial vehicles. Piksi is a single-frequency GPS receiver, operating in the consumer L1 GPS band (1.5GHz). Piksi is attractive as a GPS receiver for commercial and research micro-UAVs due to its small size, low cost, low weight, and open source hardware and software. Since Piksi's release, additional single-frequency RTK GPS receivers are appearing, such as the u-blox Neo-M8P [84].

Our approach to quantify Piksi's performance is to compare Piksi to a conventional single point position GPS under real world conditions on a quadrotor micro aerial vehicle. We quantify the precision and accuracy of the Piksi RTK GPS in three real-world scenarios: (1) recording the position of a given point (2) hovering the quadrotor at a given point, and (3) autonomously land at the quadrotor's takeoff point.

4.2 Methodology

We perform a series of tests to characterize the accuracy and precision of quadrotor localization. These tests were designed to reflect real-world outdoor conditions for a quadrotor MAV in a suburban location. Tests were performed in Northern California, USA. These tests were designed such that unambiguous ground truth data are easy to acquire without elaborate measuring tools. To that end, the tests focused on measurements of stationary or zero-setpoint conditions, where the vehicle was commanded to remain in a fixed location, or return to a previous location.

We first investigate GPS performance without flying the quadrotor. We quantify precision as the GPS's ability to measure a fixed position over time. We quantify accuracy as the GPS's ability to measure returning to a given position after moving away, commonly referred to as a "loop closure test". These tests use known, fixed positions as ground truth, against which we compare both Piksi and conventional GPS. These tests are designed to show the baseline accuracy of Piksi in comparison to conventional GPS, summarized in the first row of Table 4.1.

We then investigate the performance of the quadrotor's flight behavior when Piksi is used as a localization method. We perform the equivalent non-flying tests in this flying condition: We quantify the precision of our quadrotor's position hold capability over time. We quantify the accuracy with which the quadrotor returns and lands at a known, fixed position. These tests also rely on known, fixed positions as the setpoints for the quadrotor control system. For position hold, we use the Piksi itself as the ground truth measurement of accuracy - a reasonable ground truth given the accuracy we find in our non-flying condition. These tests are designed to show the accuracy of quadrotor flight when RTK GPS is deployed, summarized in the second row of Table 4.1.

Lastly, we investigate the end-to-end acquisition performance of Piksi in terms of the time from initial boot until acquiring an RTK lock, known as the Time To First Fix test.

Condition	Precision Test	Accuracy Test	Acquisition Test
Non-Flying	Stationary Position	Loop Closure	TTFF
Flying	Stationary Hover	Landing	

Table 4.1: Summary of tests performed to quantify RTK GPS accuracy. Non-flying tests demonstrate the baseline performance of RTK GPS in comparison to conventional GPS. Flying tests demonstrate the performance of quadrotor flight when guided by RTK GPS. Acquisition tests demonstrate the Time To First Fix performance of single-frequency RTK GPS.

Across all the previous tests, we measure the time from startup until first RTK lock. This test is designed to show a design trade-off when using single-frequency RTK GPS systems, summarized in the last column of Table 4.1.

We quantify positional accuracy and precision as the Circular Error Probability measure, or "CEP". CEP is defined as the radius of the circle in which some percentage of samples fall. This measurement is commonly used in the literature as a measure of GPS performance [41]. For our tests, we measure CEP95, the radius in which 95% of samples fall. We measure CEP95 in the north-east coordinate system. Since GPS is known to provide significantly less accuracy in altitude, we separately quantify altitude performance in meters.

4.2.1 Hardware and Software Platform

Our testing hardware consists of three physical components: A quadrotor equipped with an RTK and a conventional GPS, a base station which includes a second RTK GPS, and a landing pad.

We selected the 3D Robotics Iris quadrotor [16], shown in Figure 4.1. We feel this quadrotor is a good representation of today's commercially available quadrotor MAVs, and its fully-open-source nature allows for easy modification. The Iris quadrotor measures 0.55 m from motor-to-motor, has an all-up weight of 1.282 kg, with an additional payload capacity of 0.425 kg.

The Iris comes equipped with a 3D Robotics single point precision GPS unit containing a conventional u-blox Lea-6 conventional GPS module and a Taoglas GPS antenna [5]. The quadrotor uses this conventional GPS to measure its 2D latitude-longitude position, and an onboard barometer to measure its altitude.

For telemetry data, this quadrotor features a 57 600 baud 900 MHz data radio [3]. A

second, independent RC radio system provides joystick control for manual flight by a pilot. We added an Intel Galileo Gen 2 single-board computer [40] running custom data logging software to serve as an independent verification of Piksi's performance separate from the onboard flight controller's logging system. We also integrated the Piksi GPS as the primary flight control GPS for our test platform.

The integration of Piksi with the quadrotor followed instructions provided by Swift Navigation [67]. The top cover of the Iris was removed, and an additional platform was added, as seen in figure 4.1. This platform held both the Piksi and u-blox receivers. To reduce electromagnetic interference, this platform was shielded with copper tape, and the telemetry functionality of the remote control receiver was disabled. We further reduced electromagnetic interference by braiding and copper-wrapping the serial cable from the Piksi GPS to the Pixhawk flight controller. Piksi was connected to the GPS2 port of the onboard Pixhawk flight controller. The Piksi onboard the vehicle was configured to produce position, velocity and time solutions at 5 Hz. The Piksi GPS was connected to a Linx SH external antenna [55], mounted on a 10cm x 10cm aluminum ground plane. This antenna comes equipped with a 31db LNA. The 3D Robotics GPS unit was mounted directly behind the Linx antenna on the same platform.

The Iris quadrotor is equipped with a Pixhawk autopilot hardware [60] running the ArduCopter v3.3 RC9 [8] autopilot software. The ArduCopter estimation and control system consists of the following relevant parts: (1) Incoming sensor data from onboard inertial measurement units, barometers, and both GPSes. (2) Efficient internal storage, into which sensor data is discretized. (3) An Extended Kalman Filter of EKF, which blends sensor data into an estimated state of the quadrotor. (4) A sequence of PID controllers arranged as a successive loop closure based controller. The lowest level PID controller is responsible for stabilizing the rate of rotation and net thrust, while the top level controller is responsible for holding a position or following a given path. We used the default parameter tuning for the control and estimation system provided for the 3D Robotics Iris quadrotor.

Our base station consisted of a hand-built cart, equipped with a base station Piksi RTK GPS antenna, telemetry and control radios, and a laptop computer, as shown in Figure 4.2. The base station laptop computer ran ArduPilot's ground station software, Mission Planner [69]. This software package was used to send commands, forward GPS observations from the base station GPS, and monitor the quadrotor during flight.

As mentioned above, RTK GPS requires a base station GPS providing observations

of the GPS constellation to the RTK GPS mounted on the quadrotor. Traditionally, the moving RTK GPS is referred to as the "rover" GPS, versus the stationary "base" GPS. The base station Piksi was connected to a NovaTel Pinwheel 701-GG antenna [68]. Observation data from the base station Piksi was sent to the quadrotor using the 3D Robotics 900 MHz telemetry radio. This base station Piksi GPS provides GPS observations at 5 Hz.

Lastly, a custom landing pad was used to record takeoff and landing locations by hand. We constructed this landing pad from a 4' by 4' foam-core board. A point was marked on the board as the takeoff location, where we placed the center of gravity of the quadrotor before takeoff. During landing tests, we manually marked the board with the position of the center of gravity of the quadrotor after landing.

4.2.2 Data Collection and Analysis

We separately report on several components of this localization system. (1) The raw position measurement from the Piksi GPS, reported as double precision floating point latitude and longitude degrees. (2) The position measurement of the Piksi GPS as discretized into fixed-point degrees to an accuracy of 7 decimal places. (3) The u-blox GPS as discretized into fixed-point degrees to an accuracy of 7 decimal places, along with the barometer altitude discretized into centimeters. (4) The fused position calculated by the EKF, blending the Piksi RTK GPS and the IMU. During conventional operation, the u-blox GPS is only used for north-east positioning, and the barometer is used for altitude measurement. We follow this approach when we report conventional GPS performance, with altitude coming from the barometer.

We record several data streams, shown in Table 4.2. Data was recorded on the onboard Pixhawk flight controller. ArduCopter provides a "dataflash" library that records named and timestamped data messages for each subsystem of the controller. Additionally, the raw serial data from Piksi was also recorded on an Intel Galileo. The data from the Intel Galileo is used to verify the correctness of the dataflash logs. This independent measurement is useful, since the flight controller software relies on fixed-point integer arithmetic for position calculations which can introduce discretization errors. All the recorded data was analyzed using custom Python code. Statistical analysis was performed using numpy and scipy [85], and graphs were created using matplotlib [39].

Datastreams collected during each test
Time To First RTK Fix
RTK GPS 3D Positions (t)
Discretized RTK GPS 3D Positions, 7 decimal places (t)
EKF-fused RTK GPS 3D Positions (t)
Conventional Barometer-aided GPS 3D Positions (t)

Table 4.2: Data streams collected during RTK GPS testing. First, we measure the time from power-on until first RTK fix. We collect both raw and discretized RTK GPS positions, since the flight controller performs a position discretization. We then collect the 3D positions produced the quadrotor's EKF. Lastly, we collect the 3D positions produced by the u-blox GPS in latitude-Longitude, and the onboard barometer for altitude. Position streams are recorded at 5 Hz.

4.2.3 Testing Procedure

The general procedure followed the pattern as follows:

- Pick a testing location with a clear sky view.
- Place the base station cart and lock in position.
- Place the landing pad at least 6 ft from base station.
- Set quadrotor in location as demanded by the specific test in question.
- Power on quadrotor
- Connect base station software to quadrotor.
- Start transmitting GPS observations from the base station Piksi to the rover Piksi. Start timer to record time-to-first-fix.
- Wait for RTK lock.
- Once GPS has achieved RTK lock, record the time-to-first-fix (TTFF)
- Perform specific test
- Power off quadrotor

	Raw Piksi	**u-blox & Baro**	**ArduCopter Piksi**	**ArduCopter EKF**
North-East CEP95 (m)	0.0176	1.675	0.0186	0.051
Altitude Std. Dev. (m)	0.02037	0.1084	0.02065	0.1094

Table 4.3: Results demonstrating the precision of the components of a quadrotor's localization system when stationary. Raw Piksi exhibits two orders of magnitude higher precision that a conventional u-blox GPS in the north-east reference frame, and an order of magnitude higher precision than the barometer in altitude. ArduCopter's fixed-point math dilutes the precision by less than 2 mm, while ArduCopter's EKF dilutes the precision by about 3 cm. Position measurements reported over 5 trials of 5 minutes each, 1500 position samples per trial, with the quadrotor stationary on the ground during each trial.

4.3 Results

We now present quantitative and visual results for all our tests. We first present the precision and accuracy of the Piksi RTK GPS when manipulated by hand. We then present the performance of our quadrotor using the Piksi GPS during flight.

4.3.1 Precision of Stationary Position Measurement

This test measures Piksi's precision while stationary, with the quadrotor placed at a fixed position on the ground. We quantify position precision over a 5-minute window across 5 trials. We also report position precision over a 20-minute window across 2 trials. North-East precision is reported as the CEP95 radius. Altitude precision is reported as the standard deviation over the duration of the test.

Test Procedure We perform this test as follows: Set the quadrotor in a fixed position with a clear view of the sky, and power on. Wait until the Piksi GPS acheives RTK lock, then arm the quadrotor into flight mode without enabling the motor controllers. Let the throttle idle and record data for 5 minutes, disarm, and turn off the quadrotor. Repeat for 5 trials. We then follow the same procedure and record data for 20 minutes over two trials.

Test Result We present the results of each 5-minute test in Figure 4.3, and summarize the results in Table 4.3. We find that the Piksi RTK GPS provides two orders of magnitude

	Raw Piksi	**u-blox & Baro**	**ArduCopter Piksi**	**ArduCopter EKF**
North-East CEP95 (m)	0.016	1.672	0.0175	0.023
Altitude Std. Dev. (m)	0.01652	0.3109	0.01675	0.31255

Table 4.4: Longer-term precision results for a quadrotor's localization subsystems when measuring a stationary position. Over two 20-minute trials, raw Piksi continues to exhibit two orders of magnitude more north-east precision and one order of magnitude more altitude precision than a conventional u-blox barometer-aided GPS. Data is reported over two 20-minute trials, 6000 position samples per trial, with the quadrotor stationary on the ground during each trial.

higher precision in north-east position than the conventional u-blox GPS, with 95% of position samples falling within a 1.6cm radius. Furthermore, altitude as reported by the Piksi RTK GPS is an order of magnitude more precise than the barometer, and drifts significantly less over time. The 20 minute test corroborates these results, and demonstrates

We present the results of each 20-minute test in Figure 4.4 and summarize the results in Table 4.4. These longer-term results further corroborated by the 5-minute test results. The Piksi RTK GPS continues to provide two orders of magnitude better horizontal position precision. For vertical position precision, the barometer drift over 20 minutes is even more significant, deteriorating the conventional localization system's performance. In comparison, the Piksi GPS continues to provide centimeter precision.

4.3.2 Accuracy of Loop Closure Measurement

The loop closure test demonstrates a sensor's ability to detect revisiting the same physical position after moving away. We use this test to quantify the accuracy of the Piksi RTK GPS relative to a known point, which establishes the baseline accuracy we can expect from an autonomous landing. Furthermore, this test quantifies RTK GPS accuracy undergoing movement, but without the antenna orientation changes or electromagnetic interference from a quadrotor flight that might adversely impact performance.

Test Procedure We perform this test as follows: Set the quadrotor in fixed position with a clear view of the sky, mark this fixed position, and power on. Wait until the Piksi GPS achieves RTK lock, then arm quadrotor into flight mode without enabling motor controllers.

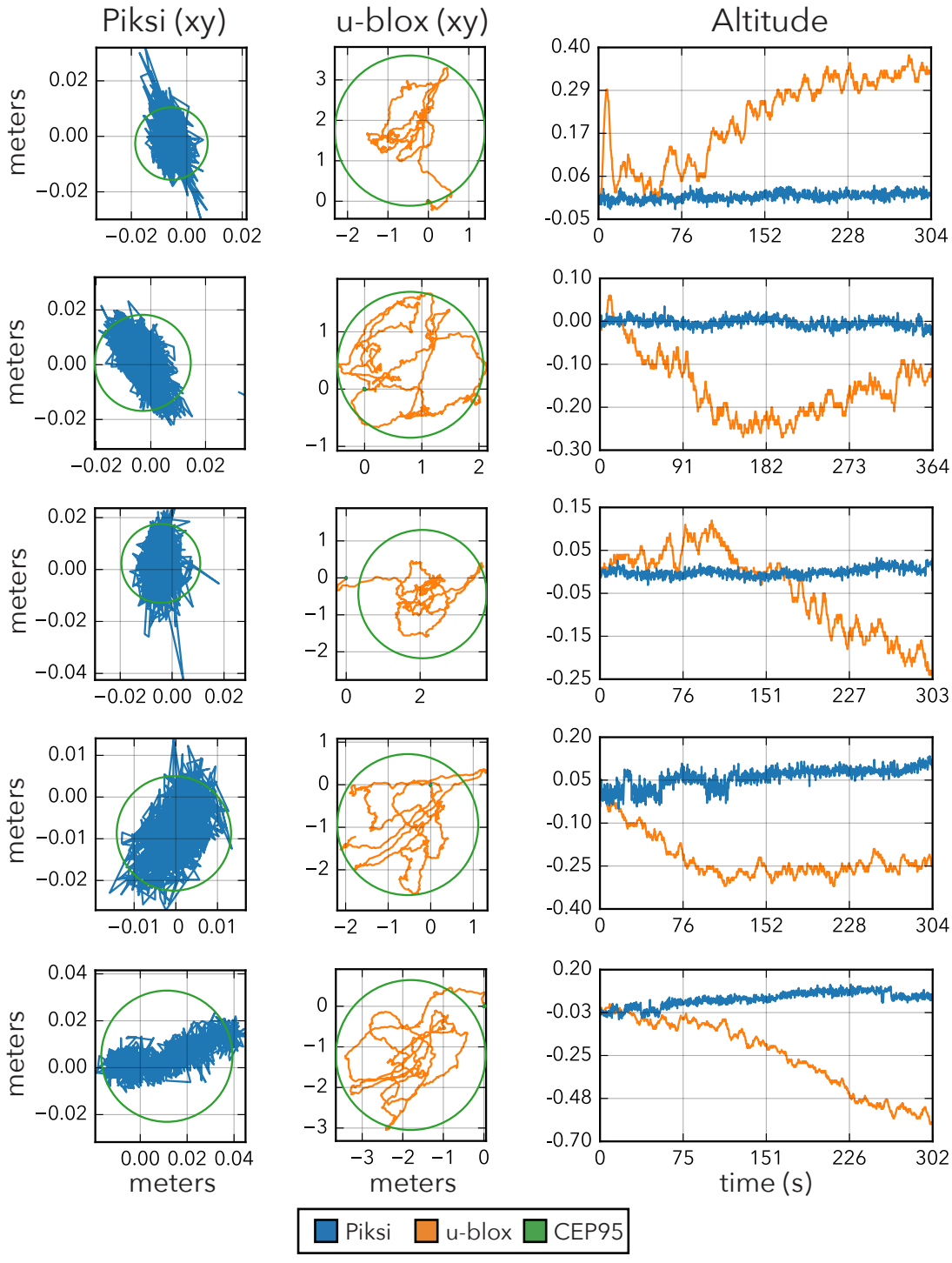

Figure 4.3: Each row visualizes the reported position of the quadrotor over 5 minutes for a single test trial. The quadrotor itself was stationary on the ground. The conventional u-blox GPS (orange) exhibits a wandering path over multiple meters. In comparison, Piksi (blue) stays within 2 cm for 95% of samples. We also plot the circle within which 95% of samples fall (green). Altitude precision of Piksi does not exhibit the drift of the barometer-aided u-blox GPS.

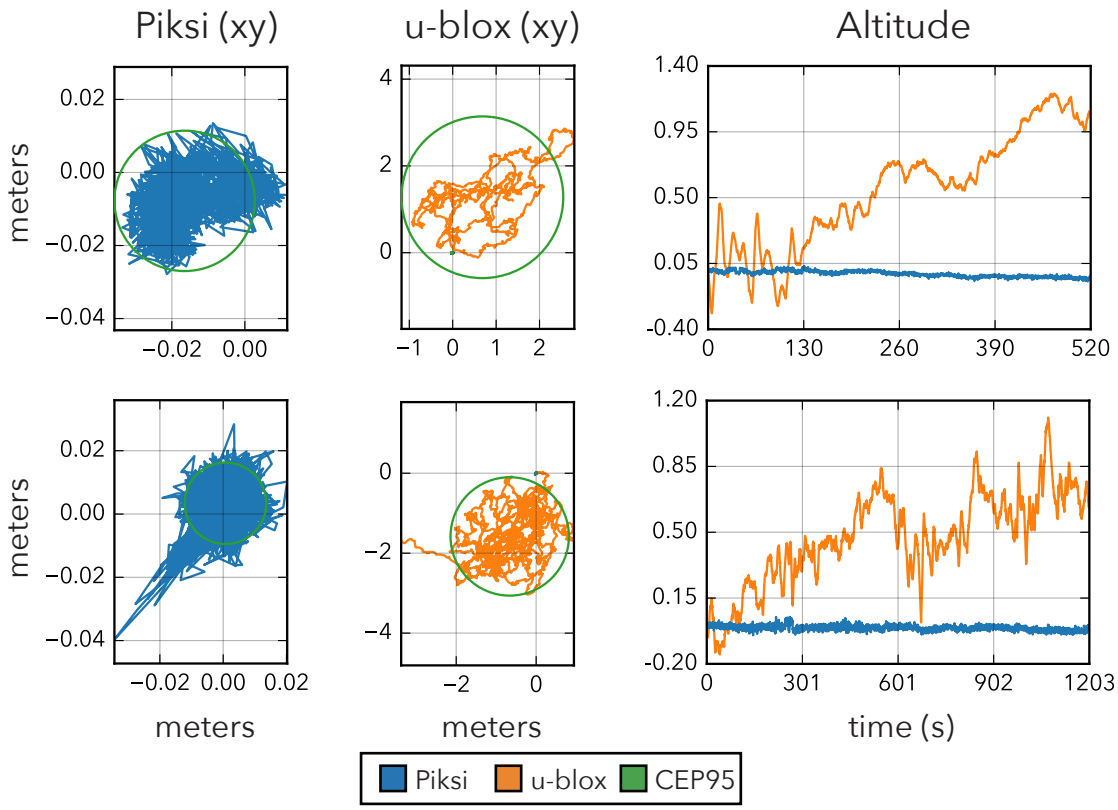

Figure 4.4: Each row visualizes the reported position of the quadrotor over 10-20 minutes for a single test trial. The quadrotor itself was stationary on the ground. All tests exhibit the same high-level qualities as the tests represented in Figure 4.3. The difference in altitude precision between Piksi and the barometer-aided u-blox GPS is further apparent here, given the additional time the barometer has to drift away from ground truth.

Sensor	Average (m)	Std Dev (m)
RTK GPS	0.011 m	0.010 m
Conventional GPS	1.058 m	0.740 m

Table 4.5: Results for localization sensor accuracy during a loop-closure test. The RTK GPS outperforms the conventional GPS by two orders of magnitude in accuracy during a loop-closure test, demonstrating the sensor's ability to detect revisiting the same physical position after moving away. Both GPSes were moved through an arbitrary pattern in a $10\,m^2$ space for 1 minute. Accuracy was measured over 5 trials.

Carefully carry the quadrotor in a arbitrary pattern around a $10\,m^2$ space, taking care to keep the quadrotor level and with an unobstructed view of the sky. After a minute, return the quadrotor to the marked starting position. Then disarm and turn off quadrotor. Repeat for 5 trials.

Test Result We present the results of the loop closure test in in Table 4.5. We find that the RTK GPS returns to within about 1 cm, while the conventional GPS returns to within about 1 m. These results demonstrate that, at least over short intervals, RTK GPS provides both precise and accurate positioning. Thus, we can expect autonomous landing to achieve up to centimeter precision. Furthermore, these results are accurate enough that we can use the Piksi RTK GPS as a ground truth measurement for the quadrotor's position during flight.

4.3.3 Precision of Fixed Point Hovering

This test measures how precise a quadrotor can hold position under real world outdoor conditions, when localization data is provided by a RTK GPS system. We investigate both holding position for 5 minutes. In the previous section we established that the Piksi RTK GPS provides centimeter-accurate positioning under these conditions, thus we use the Piksi RTK GPS as ground truth data in measuring how precisely our quadrotor hovers at a fixed position. The ArduCopter autopilot software provides a position hold ability in the form of its "Loiter" flight mode. When Loiter mode is enabled, the autopilot software uses the EKF state estimate and the control system to compensate for all external position and velocity disturbances, attempting to hover the quadrotor at a fixed position with zero velocity.

Accuracy Measurement	Raw Piksi	u-blox & Baro
North-East CEP95 (m)	0.3544	1.0474
Altitude Std. Dev. (m)	0.8047	0.7808

Table 4.6: Results for an end-to-end system test of quadrotor hover precision when using RTK GPS. The quadrotor manages to hold position to within 35 cm of the commanded horizontal position when using the Piksi GPS as a localization sensor. In this test, the Piksi itself is used as ground truth. In comparison, the u-blox GPS only manages to track the quadrotor to approximately 1 m accuracy. Altitude position hold managed to hold within 80 cm of the commanded vertical position. Position data reported over five 5-minute trials during which the quadrotor attempted to hover at a fixed position.

Test Procedure We perform this test as follows: Set the quadrotor in a fixed position with a clear view of the sky, and power on. Wait until the Piksi GPS achieves RTK lock, arm the quadrotor, takeoff up to 5 m above ground, and enable loiter mode. Allow the quadrotor to hold this position for 5 minutes. Manually land, disarm, and turn off the quadrotor. Repeat 5 for trials.

Test Result We plot the trajectory of each flight relative to the first recorded latitude-longitude and altitude point in Figure 4.5. We summarize the results in Table 4.6. The quadrotor achieves an average position hold precision of 35cm, as measured by the Piksi RTK GPS. Unlike our stationary tests from section 4.3.1, we see the quadrotor drifting around tracing out a path clearly distinguished from the positioning noise. This effect is most pronounced in the second and fourth trial. During these two trials we observed more pronounced wind gusts, corresponding to the quadrotor's position deviating from the commanded position before the control system restored the quadrotor to its initial position.

The 3D Robotics Iris can achieve flight times of 12-15 minutes on a battery charge. We also measured the precision of RTK GPS-guided position hold over an entire flight, using the same test procedure as above. We performed 2 trials with each flight achieving 12 minutes of hover. We present the trajectory of these longer trials in Figure 4.6, and summarize in Table 4.7. We observe a similar precision as in the shorter 5-minute flights, suggesting that 30 cm is a reasonable steady state hover precision.

Accuracy Measurement	**Raw Piksi**	**u-blox & Baro**
North-East CEP95 (m)	0.3015	0.804
Altitude Std. Dev. (m)	1.156	1.108

Table 4.7: Results for a longer end-to-end system test of quadrotor hover precision when using RTK GPS. During 20-minute-long hover tests, the quadrotor continues to hold position within approximately 30 cm, suggesting that our 5-minute tests demonstrated a reasonable steady state hover precision. Position data reported over two flights, each taking an full battery charge (approximately 12 minutes).

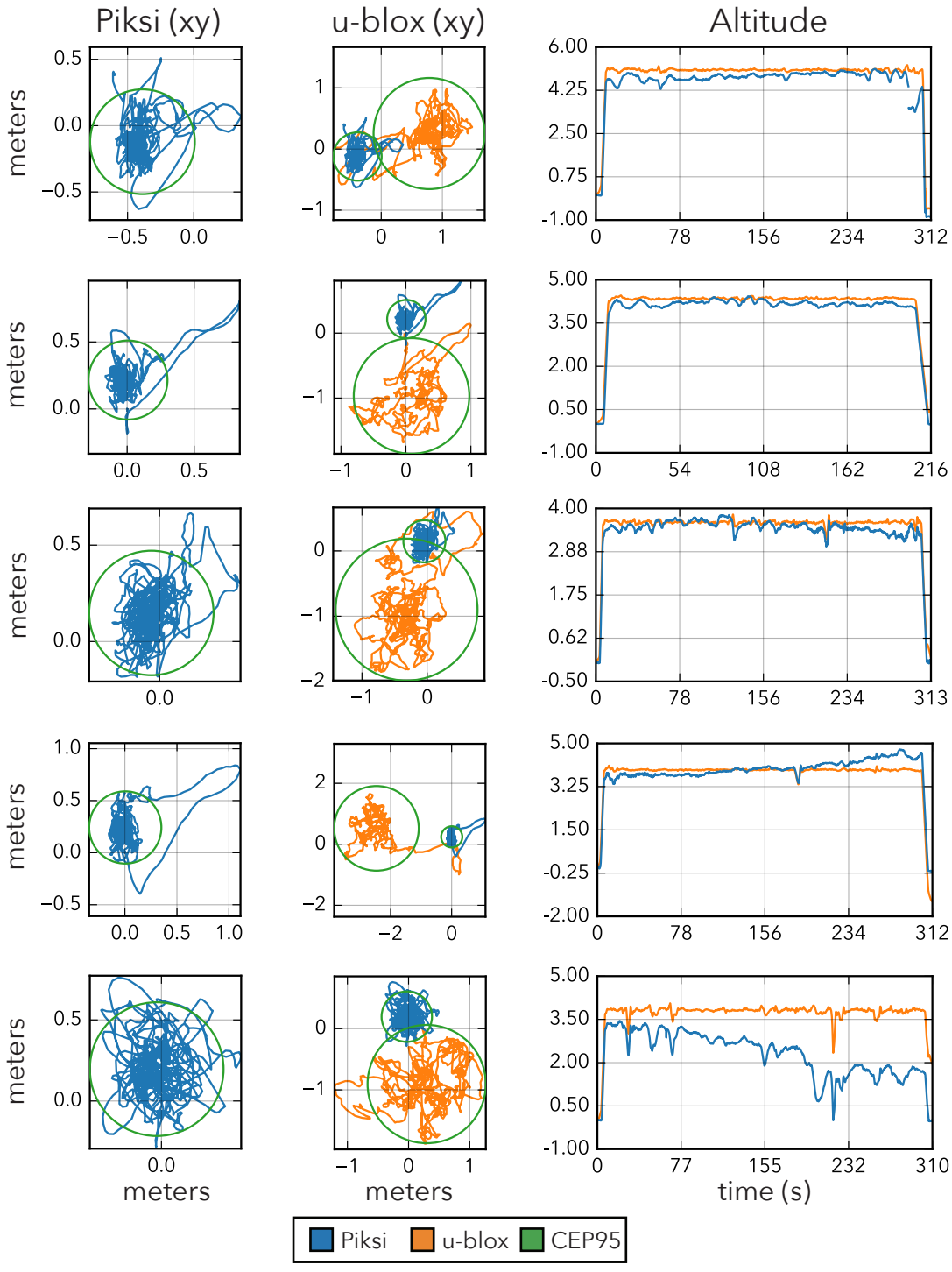

Figure 4.5: Each row visualizes a 5 minute flight during which the quadrotor attempted to hover at a fixed position. Using the Piksi RTK GPS (blue), the control system manages to keep the quadrotor within 35 cm of the setpoint 95% of the time. We also see wind gusts periodically moving the quadrotor away from the setpoint, such as in row 4. The position recorded by the conventional u-blox GPS is presented only for comparison.

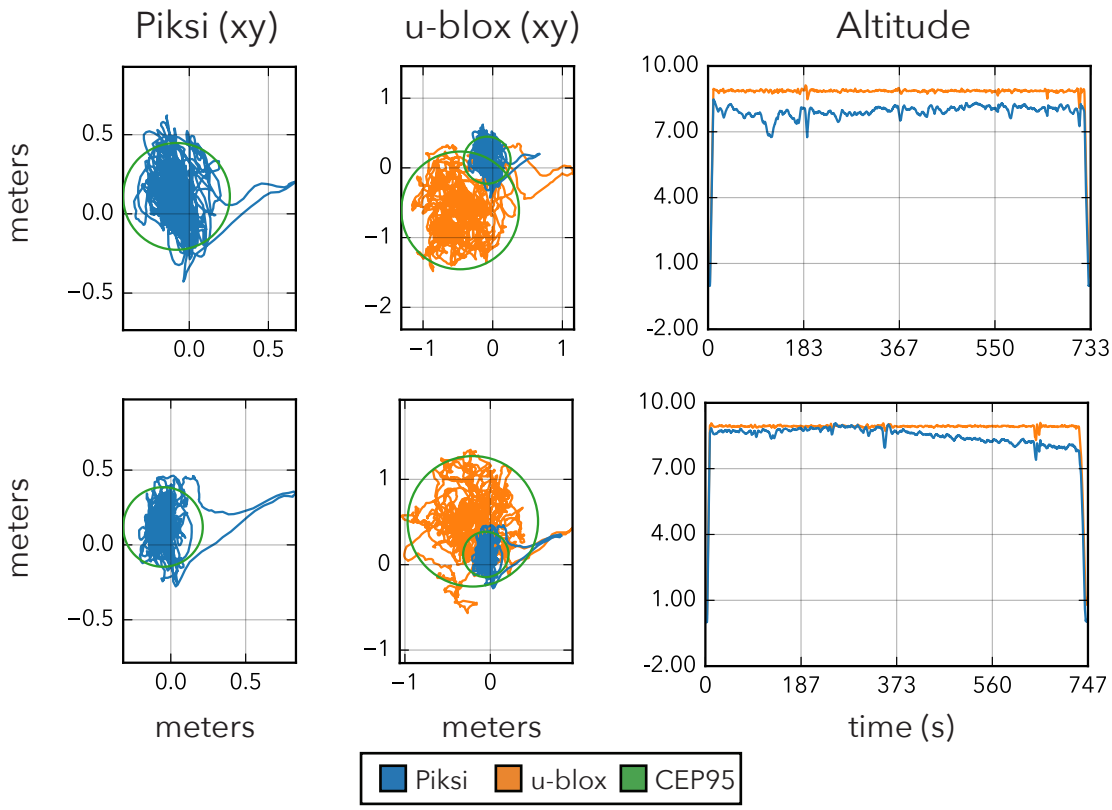

Figure 4.6: Each row visualizes the reported position of the quadrotor as it attempted to hover at a fixed position for the full flight time of a single battery charge. Comparing these graphs to the equivalent 5-minute over in Figure 4.5, we see similar behavior, suggesting that our 5-minute tests represent a reasonable steady-state behavior of our quadrotor under RTK GPS control.

Number of trials (n)	10
Average distance from home	0.214 m
Maximum distance from home	0.379 m

Table 4.8: Results demonstrating the accuracy of autonomously landing a quadrotor at the takeoff position, using the Piksi RTK GPS as a localization sensor. We manually flew the quadrotor through a series of aggressive maneuvers, then commanded an autonomous landing.

4.3.4 Accuracy of Autonomous Return and Landing

We now investigate the accuracy of autonomous landing when using the Piksi RTK GPS. In this test, we manually fly the quadrotor through a series of aggressive movements for a few minutes. We then command an autonomous landing, with the quadrotor automatically flying back to the takeoff position. Thus, this test also gives some insight into the accuracy of the Piksi GPS when undergoing real world flight characteristics.

Test Procedure We perform this test as follows: Set quadrotor in fixed position on a fixed, clearly-marked position on the landing pad, and power on. Wait until the Piksi GPS achieves RTK lock, then arm quadrotor into "Loiter" flight mode and take off. Using a set of joysticks, fly the quadrotor by hand, taking care to reach the maximum pitch and roll angles the control system supports in this flight mode. Switch the control system into "Return-To-Land" mode. Allow the quadrotor to fly back to the takeoff position and land autonomously. Wait 2 seconds, then disarm and turn off the quadrotor. Repeat this test a total of 10 trials.

Test Result We present the results of this test in Table 4.8. Using RTK-GPS, the quadrotor lands an average within 0.21 m of the takeoff position. For context, the quadrotor's longest dimension is 0.55 m, thus we land in about a third of the quadrotor's dimensions.

4.3.5 Performance: Time to First RTK Fix

GPS receivers must find a set of parameters that depend on the GPS constellation in view before a position can be calculated. These parameters include the specific Doppler frequency offset of each satellite and the orbital data that defines the exact position of each satellite, amongst others. Estimating these parameters takes time, and must be re-estimated as

Number of trials (n)	26
Median TTFF (min:sec)	5:31
Average TTFF (min:sec)	8:17
TTFF Std. Deviation (min:sec)	5:56

Table 4.9: Results measuring the time from Piksi GPS boot until acquisition of first RTK fix. Single-frequency RTK GPS systems suffer from relatively long fix times, a limitation that can be addressed by moving to multi-band receivers with a corresponding increase in cost and complexity.

satellites disappear from or come into view. Furthermore, an RTK GPS receiver must find the difference in integer carrier wave cycles between the base station and rover receivers using a set of techniques known as "integer ambiguity resolution". L1-only RTK GPS systems are particularly handicapped in resolving this integer ambiguity, since multi-frequency RTK GPS receivers can take advantage of redundant information between different frequencies to speed up resolution.

Test Result In this test, we report on the duration from initial GPS power-on, with no initial data for any of the required GPS parameters, until the first RTK GPS position fix is produced. We report this Time To First RTK Fix (TTFF) for all 26 trials presented in previous sections. This includes the time it takes to acquire the GPS satellites and resolve the integer ambiguity, with the majority of the acquisition time spent on the integer ambiguity resolution algorithm. We plot the cumulative distribution function of the TTFF measure over all trials in Figure 4.7. We find that the fastest 50% of our trials produce a first RTK fix within 5 min and 31 s. We summarize these results in Table 4.9.

4.4 Discussion

Overall Accuracy is High First and foremost, we find that the Piksi GPS is an order to two orders of magnitude more precise and accurate than a conventional barometer-aided u-blox receiver across all our tests. Secondly, we find that a quadrotor using RTK-GPS achieves decimeter position accuracy in both hover and autonomous landing—a control accuracy that's below the raw positioning ability of conventional GPS.

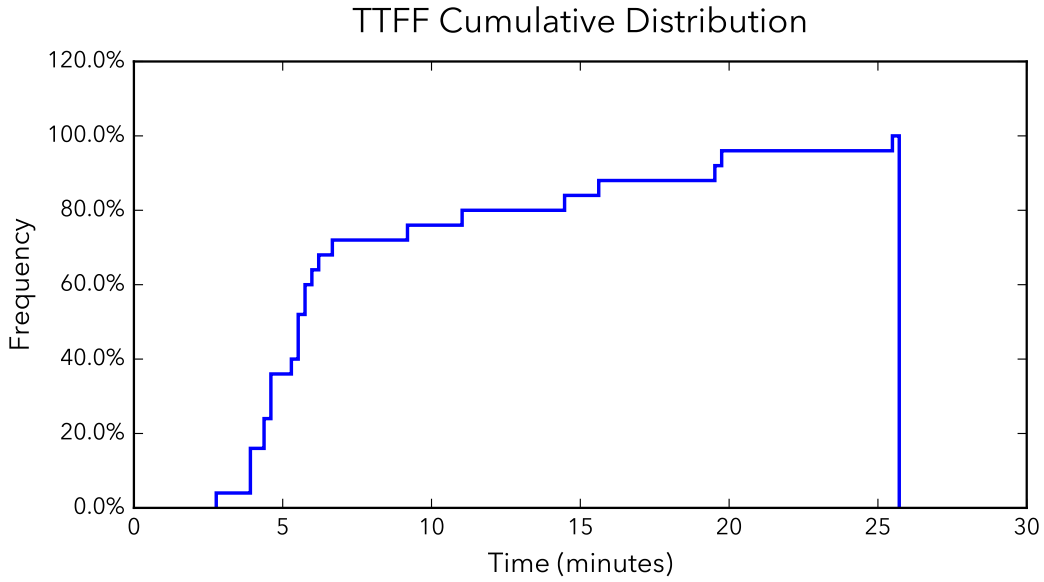

Figure 4.7: Cumulative distribution function of the Time To First RTK Fix measurements. 50% of trials fall within 5 minutes, 31 seconds, and 95% of trials fall within 19 minutes, 58 seconds.

Altitude Accuracy RTK GPS has drastic implications for localization in altitude. Current quadrotors eschew conventional GPS for altitude measurements, since altitude accuracy is even lower than several meter 2D position accuracy, due to geometry considerations [41]. Instead, current quadrotors rely on barometers to provide altitude hold. Although barometers can sense centimeter-precise relative changes over timescales of a few seconds, barometers are affected by changes in the surrounding atmosphere. These pressure changes cause unpredictable drift, affecting both the absolute and long term accuracy of an altitude estimates. The ArduCopter flight controller attempt to mitigate drift by blending accelerometer data with barometer data. Although this approach helps, we still see drift of a meter or more over 5 minutes. In comparison, RTK GPS provides a stable and low-noise altitude estimate.

Incorporating RTK GPS into a MAV is not without its challenges. The Piksi GPS is a significantly more complex system, relying on tracking both the digital code of GPS satellites, and the phase of the raw carrier wave. The result of these requirements are that RTK GPS requires a higher signal strength and correspondingly higher quality GPS constellation. Furthermore, RTK GPS is sensitive to multi-path propagation where the GPS signal reflects off surfaces, changing phase in the process. Additionally, the MAV

must have a reliable telemetry radio connection to a base station GPS. We anticipated these challenges, designed our system accordingly, and chose to only fly when the GPS constellation provided at least 9 visible satellites. We now discuss how these additional sensitivities manifest practically.

Reliability of Lock The Piksi GPS can lose RTK lock during flight, dropping back to the same accuracy as a conventional GPS. During our testing, we observed Piksi dropping out of fixed RTK mode in two of our tests once at 1:41 after takeoff in a 5 minute hover test, and once at 10:31 after takeoff in a full battery rundown hover test.

Sensitivity to electromagnetic interference We found that electromagnetic noise was an important consideration for achieving reliable and accurate positioning from both GPS receivers, particularly the Piksi GPS. In fact, our initial integration failed to achieve RTK lock. We mitigated electromagnetic interference by removing all non-essential RF equipment, copper-shielding all cabling, and placing the GPS antennas on a copper-shielded platform above the rest of the quadrotor's electronics.

Long Time to First RTK Fix Arguably the largest limitation of single-frequency RTK GPS receivers is their relatively long time to achieve a RTK lock. The Piksi GPS achieves RTK lock 6 minutes or less over the fastest 50% of our tests. 95% of our tests locked in 19 minutes or less. Other single-frequency RTK GPS receivers report similar lock times [84]. When an RTK lock is lost, this process has to start from scratch. For many applications, such as surveying a fixed point or flying in environments with a clear sky view, this limitation is quite reasonable. That being said, in cases where the Piksi GPS is anticipated to lose RTK lock during operation, such as moving through an area with limited sky view, single-frequency RTK GPS might not be viable. One possible approach is mitigating these outages by incorporating GPS satellite visibility into trajectory planning approaches. Another is providing additional information from other sensors, such as the quadrotor's Kalman filter, to the integer ambiguity resolution algorithm. Alternatively, a multi-band RTK GPS can be considered. These units can achieve RTK lock in seconds rather than minutes, but tend to be significantly more expensive.

Additional Tests The tests in this section demonstrate the performance of RTK GPS in two foundational scenarios—holding a fixed position and autonomously landing. Furthermore, these tests were designed to have an unambiguous ground truth without additional hardware requirements. That being said, RTK GPS on quadrotors can benefit from further analysis.

Quadrotors change position by changing orientation, which changes the orientation of the GPS antenna. An analysis of the correspondence between signal strength and antenna orientation can inform limits on control.

We only investigate the accuracy of position hold. A next step would be analyzing the accuracy of trajectory tracking, especially as the speed of the quadrotor increases. Since the results of this test depends strongly on the latency of GPS measurements, it would benefit from an additional measurement of ground truth. One approach may be to visually track a trajectory marked on the ground, or to use outside-in motion capture systems to track the quadrotor during flight.

4.5 Conclusion

RTK GPS is currently the only onboard absolute positioning system that can provide centimeter accuracy. In this section we have demonstrated the viability of using cheap, single-frequency RTK GPS sensors on quadrotor aircraft. These sensors are currently growing in popularity, as demonstrated by the release of several new RTK GPS sensors during the time of this writing. In fact, we expect to see the adoption of RTK GPS as part of the sensor suite for most autonomous robots. In the next chapter, we will make a case for exactly this adoption. We will exploit the accuracy of RTK GPS to track people and control a quadrotor. This will allow us to autonomously capture compelling close-up cinematography of subjects, and we will demonstrate the impact of RTK GPS on autonomous cinematography in section 5.8.

Chapter 5

Guiding Quadrotors with Composition Principles

This chapter presents "The Drone Cinematographer", a tool to use quadrotors for filming people doing everyday activities, such as playing sports, performing dance, or having a conversation. The key insight powering this tool is encoding established composition principles and canonical shots into a camera controller, enabling users to use high-level elements from the language of film to guide quadrotors. Furthermore, we show how to move the quadrotor between these canonical shots in a way thats both visually pleasing, and keeps the quadrotor from crashing into subjects. This tool builds on the RTK GPS localization testbed from the previous chapter to accurately track both our quadrotor and our subjects, enabling accurate and autonomous camera placement in large-scale outdoor environments.

The contents of this chapter was adapted from a 2016 arXiv paper co-authored by Niels Joubert, Jane E, Dan B Goldman, Floraine Berthouzoz, Mike Roberts, James A. Landay and Pat Hanrahan. All uses of "we", "our", and "us" in this chapter refer to all listed authors.

5.1 Approach

Our idea is to create a semi-autonomous quadrotor camera system that positions itself relative to the people in a scene according to rules of cinematography. This allows the quadrotor to capture well-composed footage of people without needing another person to

Figure 5.1: In this chapter we present "The Drone Cinematographer", an end-to-end system for autonomously capturing well-composed footage of two subjects with a quadrotor in the outdoors. Here we show a set of static shots captured by our system, covering a variety of perspectives and distances. We demonstrate people using our system to film a range of activities – pictured here: taking a selfie, playing catch, receiving a diploma, and performing a dance routine.

manually fly the quadrotor. Flying a quadrotor is challenging and requires skill, and needing a pilot requires an additional person. Our work builds on the capabilities of recent commercial systems that have a *follow-me* mode. We seek to enable a more sophisticated *cinema-mode*, where the quadrotor seeks to capture visually pleasing shots of the activity being undertaken.

We focus on filming scenes with one or two people, where both people are fairly stationary. Under these simplifying assumptions, we demonstrate the efficacy of our idea for a specific set of scenarios, such as two people taking a "selfie", playing catch, or performing a hip-hop dance routine.

We draw upon cinematographic practice, and past work in computer graphics that adds visual composition principles to virtual camera controllers. Cinematographers have devised a small set of canonical shot types (e.g., apex, internal, and external), and computer graphics researchers have developed algorithms for placing the camera to generate these shot types. The result is that if we know the locations of the subjects, we can compose visually pleasing footage by correctly placing the camera.

However, there are several challenges to using visual composition principles and canonical shots with quadrotors, all related to the fact that the quadrotor is a physical device

moving in the real world: A quadrotor must obey the laws of physics, which constrain how it can fly. Moreover, the quadrotor must know where the subjects are to be able to film them. Most importantly, the quadrotor must not fly into people and cause them harm.

We overcome these challenges through accurate tracking and a unique trajectory planning algorithm. We track the position of both the quadrotor and two people using the Real Time Kinematic (RTK) GPS sensor investigated in Chapter 4. RTK GPS allows us to estimate positions to within 2 cm accuracy, which is much better than the roughly 2 m accuracy of conventional GPS. Given the sizes and distances involved, this increased accuracy is essential to place the subjects correctly within a frame.

Our trajectory planning algorithm builds upon previous work in designing and optimizing quadrotor camera trajectories. Like previous work, we require that the trajectory flown by the quadrotor obey the laws of physics. This requires that the path be C^4 continuous, and that the movement along the path not exceed a maximum velocity. The main new technical contribution in this chapter is a method to move between camera shots safely; that is, we guarantee that the position of the quadrotor relative to a subject is greater than a minimal distance. We enforce this no-fly "safety sphere" while maintaining a pleasing composition of the image.

We present the first end-to-end system that leverages composition principles and canonical shots to guide autonomous quadrotor cameras filming people in the real world.

5.2 Design Goals and Challenges

We design our system to achieve the following visual composition goals:

Employ Canonical Shots The literature of cinematography offers numerous high-level composition principles that guide the framing of a set of commonly used shots [9, 36, 77]. These shots specify where subjects lie within the frame, and implicitly define the relative placement of the camera with respect to the subjects. Following this approach, we implement a set of canonical static shots. These shots place the camera at a fixed pose, allowing subjects some freedom to move in the frame. We also take care to respect the compositional principle of the *rule of thirds*: the focal point of a shot is placed at the intersection of horizontal and vertical lines splitting the screen into thirds.

Maintain Compositional Continuity Moving shots and transitions should be pre-planned such that start and end frames are compositionally balanced, and intermediate framings should vary smoothly, with subjects moving in roughly straight lines from one camera movement to the next. We wish to avoid indecisive and jerky motions: Movement that is too fast can be hard on viewers' eyes, and can detract from the content of the frames. In addition, we seek to preserve the *line of action* between the two subjects: Throughout a shot sequence, the camera stays on the same side of this line to encourage visual continuity. The rationale for this principle is that if the camera switches sides, the subjects will switch left and right sides in the frame, which is disorienting for the viewer.

To achieve our stated design goals, our system must overcome the following technical challenges:

Construct and Maintain a Virtual Representation of the Scene Our virtual representation of the scene must be accurate enough to plan shots with the intended visual composition. This implies that the system must accurately track the pose of both subjects and camera. Our system supports capturing shots containing one or both people, and therefore our tracking system cannot assume both subjects are always visible to the primary camera.

Plan Safe Camera Locations Based on the virtual scene, our system will plan locations for the camera in the physical world. In consideration of both safety and personal space, we introduce a safety constraint. This constraint states that we only choose camera positions outside of exclusion zones where the camera must not be placed. We call these zones, centered on each subject, *safety spheres*, represented as a minimum distance constraint in 3D space.

Plan Visually Pleasing Transitions In contrast to a virtual camera, a physical camera cannot be immediately placed at a new location: It has to transition in space between poses. We implement transitions between shot locations, planned such that they attempt to maintain pleasing composition throughout. During these transitions, the quadrotor camera must also maintain the safety minimum distance constraint to both subjects.

Place the Camera According to Plan Finally, the virtual plan of static shot locations and transitions must be executed by a physical quadrotor control system in real time.

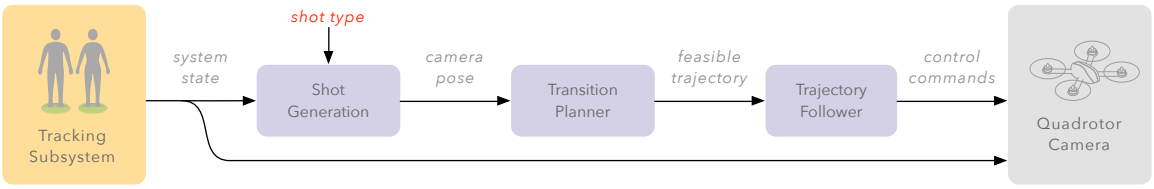

Figure 5.2: Major technical components of our real-time autonomous cinematography system, the Drone Cinematographer. Our tracking subsystem estimates poses of subjects and the quadrotor camera in the real world. Whenever a new shot type is provided by a user (shown in red), our system generates a camera pose that satisfies visual composition principles and physical placement constraints. To move the camera from its current pose to this new camera pose, a feasible, safe, and visually pleasing transition is planned. Finally, a sequence of quadrotor and gimbal commands control the quadrotor camera autonomously.

5.3 Technical Overview

We provide an overview of the major technical components of the Drone Cinematographer system in Figure 5.2. At the core of our system is a shot generator that produces well-composed static shots of two subjects, described in Section 5.5. We build this shot generator based on a set of canonical shot types and visual composition principles. This shot generator enables users to specify a desired camera pose at a high level, using terminology from the cinematography literature. Given a canonical shot type, our shot generator produces a static camera pose consisting of a look-from and look-at point, taking care to ensure the resulting pose is safe with respect to both subjects. Our system then places and holds a camera at this pose, recording video.

We prototype a simple user interface driven by a visualization of the virtual scene representation and a simulation of our robotic camera. Our interface displays a 3D rendering of the current virtual representation of the scene from the perspective of the quadrotor camera. A user can select any shot type, and virtually see the resulting shot. The user can also issue shot types to the real quadrotor camera. Our system will then place the virtual camera at a new static camera pose corresponding to the selected shot type.

To place the camera at a new static camera pose, our system creates a transition from the current camera pose to this new pose. This transition needs to take into account the

visual contents of video recorded during a transition, respect the safety of subjects, and adhere to the capabilities of quadrotors. With this in mind, we design an algorithm for synthesizing quadrotor camera trajectories between two static camera poses (Section 5.6). At a high level, our approach is to optimize a *blend* of easy-to-generate basis trajectories by solving a constrained non-convex optimization problem. Using this algorithm, we produce a look-at and look-from trajectory for our quadrotor camera.

Our shot generator produces a camera pose relative to subjects, and thus needs to know the pose of each subject. Our system tracks subjects by having them wear a helmet containing the high-accuracy RTK GPS from Chapter 4, and inertial measurement unit (IMU) sensors. We use the same tracking system to accurately localize our quadrotor camera.

Lastly, our system captures shots by issuing control commands to a quadrotor camera. These control commands takes the form of look-from and look-at setpoints driving a feedback controller running on a real-world quadrotor. During a static shot, our system holds a quadrotor camera at a fixed look-from and look-at setpoint until the user commands a new shot. During a transition to a new shot, our system sends a stream of look-from and look-at samples along the transition trajectory, moving the quadrotor camera.

Throughout our system, we consider various approaches to keep our subjects safe in the presence of a quadrotor aircraft. When our system places static shots, it keeps the quadrotor a safe distance from our subjects. While a camera is at a static camera pose, subjects are free to move around and the camera will not change its position. When our system plans a transition, it ensures the resulting trajectory stays a safe distance from subjects, but assumes the subjects will not leave their safety spheres during a transition. Overall, our system does not prevent a subject from intentionally colliding with the quadrotor. We expect advances in dynamic obstacle avoidance to address this problem. For the purposes of this work, we feel it is reasonable to assume a benevolent subject that is willing to remain fairly stationary within the bounds of the safety sphere.

5.4 Modeling Subjects and Cameras

In this section, we introduce the subject and quadrotor camera models used in our system, shown in Figure 5.3.

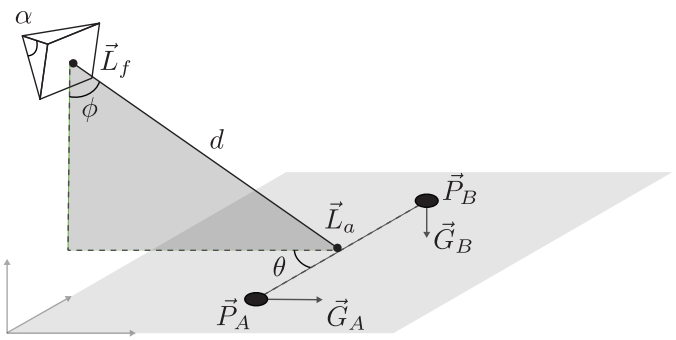

Figure 5.3: Overview of our camera and subject model. Each subject has a position and a gaze vector, $\vec{P}$ and $\vec{G}$. We model our camera as a look-from point $\vec{L}_f$, a look-at point $\vec{L}_a$, and a field of view α. We also introduce angles θ and ϕ to describe angles between the direction that the camera is pointed and the line of action, and d to indicate the distance between the look-at and look-from points.

5.4.1 Subject Model

Each subject is modeled as a position $\vec{P}_i$ and gaze vector $\vec{G}_i$ that represents the subject's head position and orientation. Our tracking system, presented in Section 4, estimates these positions and orientations. We calibrate our tracking system so that the position of the subject corresponds to the center of their head at eye level. Each subject is further described by their height, and by a minimum distance value d_{min} defining the subject's safety sphere. Our system will not place the quadrotor camera closer than d_{min} to a subject.

5.4.2 Quadrotor Camera Model

We use the joint quadrotor and camera model introduced in Chapter 3, which models a camera on a gimbal attached to a quadrotor aircraft. This model has two important implications for us. First, this model enables our system to specify the behavior of a quadrotor camera using *look-from* and *look-at* world space points (L_f and L_a in Figure 5.3). We fix the *up* vector to be vertical.

To model moving quadrotors and subjects, we allow the look-at and look-from points to follow a trajectory. To respect the dynamics and physical limits of the quadrotor, these look-at and look-from trajectories must be C^4 continuous, and the velocity along the look-from trajectory must be less than the maximum speed at which the quadrotor can fly.

Finally, our physical camera model has a fixed field of view $\alpha_{\max}$.

5.5 Generating Static Shots

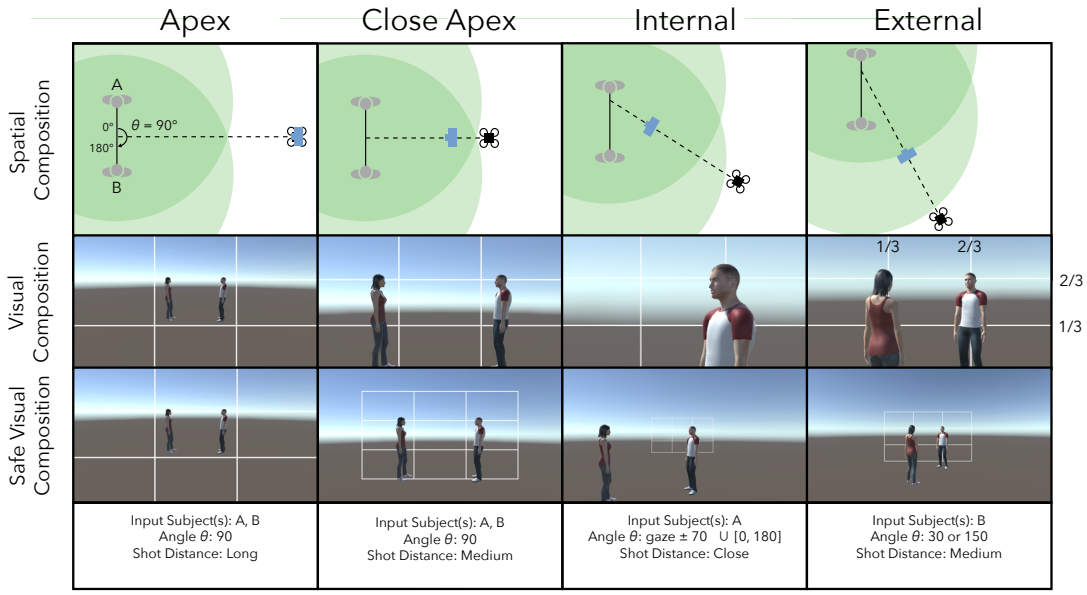

Figure 5.4: This table shows the four main types of shots we implemented in our system. The top row shows the spatial layout of each shot from a bird's eye view. The blue camera is the goal virtual camera position, and the black quadrotor shows the same shot from a safe distance. The second row shows the intended visual composition, applying the rule of thirds. In the third row we show the same shot after applying our minimum distance constraint. We move the camera to a safe distance by decreasing the field of view, while maintaining the size of the subjects by cropping the frame. The gaze direction of the primary subject is measured relative to the line of action. Finally, we list the parameters that define each shot. In this illustration, the pitch angle $\phi = 0$.

Our system is designed to capture canonical shots which adhere to principles gleaned from the cinematography literature. Here we describe the set of shots we selected, and how they are implemented in our system.

5.5.1 Defining Shots

The inputs to the shot selection system are the positions and gaze directions of the two subjects, and the desired type of shot. The outputs of the shot selection system are the look-from and look-at points, and the field of view of the camera.

A shot type is defined by the input subject(s), shot distance, and orientation angles. Here we describe these parameters and how they impact the outputs for a given shot.

If the shot has a single *input subject* (e.g. the internal and external shots described below), we place our camera so that this primary input subject is at a screen space position that follows the compositional principle of the *rule of thirds*. More specifically, the eyes of the subject are placed at the intersection of horizontal and vertical lines splitting the screen into thirds. Subjects facing to screen right lie along the left vertical one-third line, and subjects facing to screen left lie along the right vertical one-third line. If a shot has multiple input subjects (e.g. an apex shot), we frame both subjects by placing the camera so the average position of the eyes of the two subjects lies on the center of the upper horizontal two-thirds line.

We next set the distance of the camera to the primary subject, or, in the case of two subjects, to the average position. We specify *shot distances* qualitatively using well-defined cinematographic conventions: close, medium, or long [9]. These distances are defined by the portion of a subject's body that should appear in frame—specifically, we represent these as the approximate number of heads below the horizon line (close is 2.5, medium is 4, and long is 7.5). We geometrically calculate an absolute distance d based on shot distance and the subject(s)' average height.

The orientation angle θ defines the yaw of the camera relative to the line of action. Cameras are placed on the same side of the line of action. The system initially chooses the side that sees more of the subject(s)' faces, which can be determined from the gaze directions, and keeps the camera on this side. We also have an angle ϕ that defines the pitch of the camera. This is usually set to 0 degrees, generating shots with a straight-on view of subjects.

The look-from and look-at point for a shot is calculated from the screen space position of the subjects, the distance of the camera, and the orientation angle. An approximate look-at point is placed on the line of action, and an approximate look-from point is placed relative to the approximate look-at point using θ, ϕ, and d. This places the camera at the correct orientation and distance from the subject, but does not yet guarantee the subject appears in the correct screen space position. We shift the approximate look-from and look-at point to move the subjects to the correct screen space position in the frame.

The distance of the camera to the subject depends on the field of view of the camera and the desired size of the subject in the frame. Unfortunately, this may cause the camera to be placed inside the safety spheres surrounding the subjects. We fix this hazard by moving the camera further away until it is outside the safety spheres. Moving the camera further

away causes the subject size to shrink, and the composition to change. To compensate for this change, we calculate a crop, $\alpha \leq \alpha_{\max}$, that maintains the visual composition of the shot. It is possible for the crop to be so extreme that the resolution loss makes the resulting footage practically unusable even though subjects are correctly framed. Fortunately, our system can use the same approach for quadrotors with optical zoom lenses to change the view of view without incurring resolution loss.

5.5.2 Types of Canonical Shots

We chose to implement four main shots in our system: apex, close apex, internal, and external. These shots are adapted from the camera modules in He et al. [36]. Figure 5.4 shows these four shot types, the relative spatial placement of the camera and subjects, as well as the resulting visual compositions.

- **Apex** A long shot of both subjects, vertically centering characters in the frame. The subjects' average eye level is placed centered horizontally at the two-thirds line (Figure 5.4 (a)).
- **Close apex** A medium shot of both subjects, framed similarly to the Apex shot (Figure 5.4 (b)).
- **Internal** A close shot of a single input subject, oriented relative to gaze to guarantee a semi-frontal view of the primary subject. The subject is placed on one of the vertical thirds lines such that the majority of empty screen space is in front of their (Figure 5.4 (c)).
- **External** A medium shot of a single input subject, looking over the shoulder of the other subject. If the primary subject is on the left side, they are placed at the one-thirds line with the other subject in the right third of the frame, and vice versa (Figure 5.4 (d)).
- **Apex From Above, External From Above** We also implemented alternate versions of the Apex and External shots, but placed above subjects to mimic canonical top-down shots [77]. These are implemented with the same parameters as the shot types described above, but with the pitch angle, ϕ, set to place the camera looking down from above the subjects.

5.6 Transitioning Between Shots

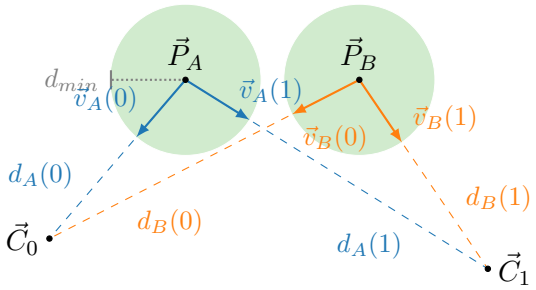

Figure 5.5: Here we show the terms we use to construct basis vector paths. We assume we are given two subject positions $\vec{P}_A$, $\vec{P}_B$ and an initial and final camera position $\vec{C}_0$, $\vec{C}_1$. In blue, we show the terms that generate a basis path for Subject A. We extract an initial and final vantage vector $\vec{v}_A(0)$, $\vec{v}_A(1)$, and an initial and final distance $d_A(0)$, $d_A(1)$. We linearly interpolate from $d_A(0)$ to $d_A(1)$, and spherical linearly interpolate from $\vec{v}_A(0)$ to $\vec{v}_A(1)$. Scaling the interpolated vantage vector by the interpolated distance as we interpolate produces a basis path. In orange, we show the same quantities relative to Subject B.

In this section, we consider the problem of moving a quadrotor camera from one static shot to another. Specifically, we want to find a quadrotor camera trajectory that maintains a visually-pleasing composition, respects the dynamics and physical limits of our hardware, and ensures safety of our subjects. Our main insight is to avoid solving a general trajectory optimization problem in the full state space of the quadrotor. Instead, we *blend* between two easy-to-generate and visually-pleasing *basis* trajectories. This approach is more computationally efficient than solving a general trajectory optimization problem, and produces safe and visually pleasing trajectories.

We summarize our method for transitioning between static shots as follows. First, we generate a pair of basis paths by adapting a composition-aware interpolation technique introduced by Lino and Christie [54]. These basis paths produce visually pleasing results, but might get too close to the subjects. We produce a final path that respects our minimum distance constraints, by optimizing a *blending function* that blends between our basis paths. We then apply an easing curve to our final path, producing a final look-from trajectory. We generate a look-at path by linearly interpolating look-at points in world space. We apply the same easing curve (as was applied to our look-from path) to our look-at path to produce a final look-at trajectory.

We assume we are given as input a start and end camera position $\vec{C}_0$ and $\vec{C}_1$, as well as

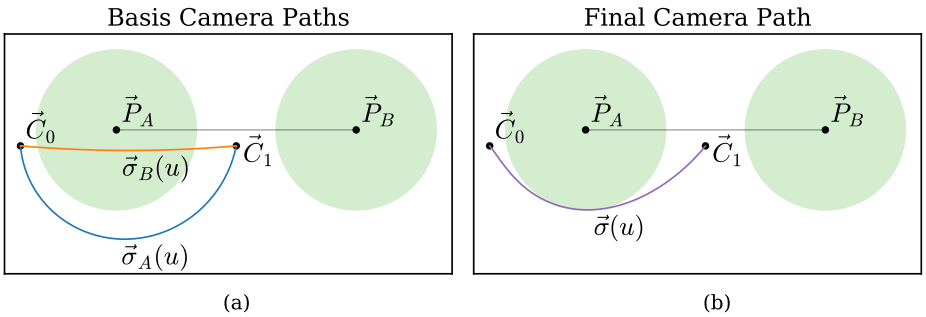

Figure 5.6: Blending between quadrotor camera trajectories. (a) A top-down view of our basis camera paths, generated using spherical interpolation around Subject A (blue path) and Subject B (orange path), respectively. The green circles represent the safety sphere of each subject. Note that the orange basis path violates the minimum distance constraint (green circle) around Subject A. (c) We find a final camera path by blending these two basis paths, enforcing the constraint that the final path is outside *both* unsafe regions.

start and end look-at points. We assume the start and end positions of the look-at point do not change during the transition. We also assume the start and end camera positions are safe – that is, the distance from the start and end camera position to each subject is greater than d_{min}.

5.6.1 Generating Basis Paths

We assume that the paths we consider in this section are parameterized by a scalar *path parameter* u. We define two basis paths, one for each subject. We define the camera-to-subject distance at each point along each basis path as $d_i(u)$, where i is an index that refers to each subject. We set $d_i(u)$ to be the linear interpolation between the camera-to-subject distances of our initial and final camera positions $\vec{C}_0$ and $\vec{C}_1$. Likewise, we define the vantage-vector at each point along the basis path as $\vec{v}_i(u)$. We set $\vec{v}_i(u)$ to be the spherical linear interpolation of the normalized camera-to-subject vantage vectors corresponding to our initial and final camera positions. We define our two basis paths $\vec{\sigma}_i(u)$ as follows. This construction is shown in Figure 5.5.

$$\vec{\sigma}_i(u) = \vec{P}_i + d_i(u) \cdot \vec{v}_i(u) \quad \text{for } u \in [0,1], i = \{A, B\} \tag{5.1}$$

Our basis paths have the following useful properties:

- $\vec{\sigma}_i(u)$ is C^∞ continuous with respect to u. This is because spherical linear interpolation between two vectors and linear interpolation between two points are both C^∞ continuous interpolation schemes. This property is useful, since trajectories must be at least C^4 continuous with respect to time in order to satisfy the quadrotor dynamics.
- It is guaranteed that $\vec{\sigma}_A(u)$ will never get too close to subject A, and $\vec{\sigma}_B(u)$ will never get too close to subject B. This is because our start and end camera positions satisfy the minimum distance constraint, and we linearly interpolate distance. This property is useful, because it suggests that we can generate a path that never gets too close to *either* subject by blending between our basis paths.

5.6.2 Optimal Blending of Basis Paths

In the previous section we generated two paths, one relative to each subject. Previous work averages these two paths together to produce a final path. Unfortunately, the resulting path can violate our minimum distance constraint (see Figure 6).

We introduce a blend function $w(u)$ that blends the two basis paths into a final path $\vec{\sigma}(u)$ as follows,

$$\vec{\sigma}(u) = w(u) \cdot \vec{\sigma}_A(u) \; + \; (1 - w(u)) \cdot \vec{\sigma}_B(u) \tag{5.2}$$

We now use constrained optimization to find a good blend function. We seek a blend between the two basis paths (1) that is as close as possible to the two input paths, and (2) obeys the minimum distance constraint. During this optimization procedure, we also enforce C^4 continuity (and hence, C^4 continuity of our final path), and we optimize the overall smoothness of our blend.

Enforcing C^4 Continuity In order to enforce C^4 continuity, we discretize our blend function $w(u)$ into a sequence of n sample points w_k where $k = 1 \ldots n$ (thus $u = \frac{k}{n}$).

Following the approach outlined by Roberts and Hanrahan [76], let $\mathbf{w}_k$ be the value and the first four derivatives of $w(u)$ at each sample point k. Let v_k be the 5^{th} derivative of $w(u)$ at sample point k. Let du be the delta between successive sample points. We enforce C^4 continuity of our blend as follows,

$$\mathbf{w}_{k+1} = \mathbf{w}_k + (\mathbf{M}\mathbf{w}_k + \mathbf{N}v_k)du$$

$$\text{where} \quad \mathbf{M} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{N} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{5.3}$$

$$\text{subject to} \quad v^{\min} \leq v_k \leq v^{\max}$$

Similarly to Roberts and Hanrahan [76], we introduce $v^{\min}$ and $v^{\max}$ to control how much $\frac{d^4w}{du^4}$ is allowed to vary between sample points while still considered continuous. In our implementation, we heuristically set these values inversely proportional to the number of samples n of our blend.

Optimization Problem Stating our optimization problem formally, let $\mathbf{W}$ be the concatenated vector of decision variables $\mathbf{w}_k$ and v_k across all sample points $k = 1, \ldots, n$. Let λ be a parameter that trades off between smoothness and our preference for giving equal consideration to each basis trajectory. We find the optimal set of blend function values and derivatives as follows,

$$\begin{aligned} \underset{\mathbf{W}}{\text{minimize}} \quad & \sum_{k=0}^{n} \left((w_k - \frac{1}{2})^2 du \; + \; \lambda (\frac{d^4 w}{du^4})^2 du \right) \\ \text{subject to} \quad & \mathbf{w}_{k+1} = \mathbf{w}_k + (\mathbf{M}\mathbf{w}_k + \mathbf{N}\mathbf{v}_k)du \\ & 0 \leq w_k \leq 1 \\ & v^{\min} \leq v_k \leq v^{\max} \\ & \|\vec{\sigma}_k - \vec{P}_0\| \geq d_{min} \\ & \|\vec{\sigma}_k - \vec{P}_1\| \geq d_{min} \\ \text{where} \quad & \vec{\sigma}_k = w_k \cdot \vec{\sigma}_A(\frac{k}{n}) + (1 - w_k) \cdot \vec{\sigma}_B(\frac{k}{n}) \end{aligned} \tag{5.4}$$

The problem in (5.4) is nonconvex, and is therefore sensitive to initialization. We initialize our solver with a default initial blend that averages the input trajectories exactly, with weights set to $\frac{1}{2}$.

Performance In our implementation, we solve the problem in (5.4) using the commercially available non-convex solver SNOPT [33]. We rely on SNOPT to numerically calculate the Jacobian matrices of our optimization problem. In all our experiments, we discretize w at a moderate resolution of $n = 50$ samples. We experimentally find that we can solve this optimization problem in under 500ms on a 2.8 GHz Intel Core i7 processor for all our shots.

5.6.3 Generating the Final Trajectory

Our final path is parameterized in terms of the path parameter u, and not time. We apply an easing curve to our path to generate a smooth final trajectory, using the method shown in Chapter 3.

So far we have only computed the look-from trajectory. We still need to generate the look-at and field of view trajectory. To do this, we linearly interpolate between the two start and end look-at points and field of view values, and then apply the same easing curve.

Once we have a final camera trajectory, we check it against our quadrotor model for violations of physical limits using our open-source Flashlight library [75]. If any exist, we linearly time-stretch the easing curve until no constraints are violated. Practically speaking, we conservatively set the total time of our transitions to avoid violating physical constraints. We have found that our default easing curve rarely produces trajectories that exceed these limits.

5.7 Tracking and Control Platform

Our system has to maintain a virtual representation of the scene to plan shots, and place a quadrotor camera accurately according to this plan. Here, we present a hardware platform that achieves these goals (Figure 5.7). We place active trackers containing the RTK GPS from Chapter 4 and an IMU in the shape of the Pixhawk flight controller hardware on

(a) (b) (c)

Figure 5.7: Overview of the major physical components of our hardware platform. A subject (a) wears a position and orientation tracker on a helmet. GPS corrections are provided from a base station (b). The quadrotor (c) is equipped with an orientable camera and similar tracking hardware.

each subject and the quadrotor. We use the IMU to estimate the orientation of the tracker. Specifically, both the IMU and GPS output is fed into a Kalman filter to estimate the current position and orientation state. This state is communicated to a central computer using a low latency network. The central computer executes our shot planning algorithm, producing a quadrotor trajectory. The trajectory is turned into a sequence of control commands, sent to the quadrotor camera using the same wireless network.

We use off-the-shelf long range radios to communicate between the various components of our platform. Specifically, we use the Ubiquiti Bullet M5 radio. This radio provides sub 5 ms communication latency over a range of several hundred meters, enabling our system to maintain an up-to-date virtual representation of the scene.

The quadrotor we use in this chapter is a modified 3DR Solo, carrying a gimbal-mounted GoPro Hero 4 Black camera, and the RTK GPS. This quadrotor uses the APM autopilot software [8], which includes an onboard Kalman filter for position estimation. We extend and tune the Kalman filter to accept position estimates from the RTK GPS. To fly a quadrotor camera according to a look-from and look-at trajectory, we use the same trajectory follower as in Chapter 3 to drive the onboard control system. High-accuracy position estimates from the RTK GPS aid the control system in placing the quadrotor accurately. In our results, we present experimental evidence to support the efficacy of this platform for autonomously capturing cinematography.

Figure 5.8: Here we show a sequence of static shots captured by our system during a single shoot of two subjects playing catch. Top left to bottom right: apex from above, external of subject in red, external from above of subject in red, internal of subject in gray, external of subject in gray, apex. Notice the line of action is maintained throughout these shots: The person in red is always on the left, and the person in gray remains on the right.

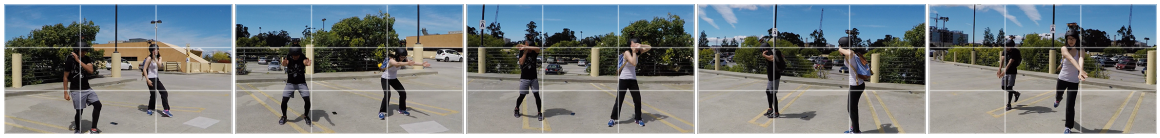

Figure 5.9: Here we show a sequence of frames from a transition captured by our system while filming a choreographed dance routine. This transition goes from an external shot of the right character to an external shot of the left character.

5.8 Results

In order to test our system, we captured footage using our system of a range of scenarios including taking a selfie, playing catch, and performing a choreographed dance routine. We show several shots produced by our system in Figure 5.1 and Figure 5.8.

Well-composed Static Shots We present examples of several static shots generated using our system in Figure 5.8. These shots are captured from a single flight, and feature two subjects playing catch. Each of these shots respect the rule of thirds and our safety constraints, and are cropped to match the intended compositions. Furthermore, our system maintains the line of action over successive shots. As a consequence, in each of these shots, the relative left-right positioning of subjects in frame is consistent.

Safe and Visually Pleasing Transitions Our transition planner is able to produce transitions that are both safe and visually pleasing. Figure 5.9 shows a set of still frames from a transition captured using our system. Both subjects change smoothly in size, and move reasonably in screen space through the transition.

Capturing Scripted Scenarios We also used our system to capture a fully scripted scenario. We staged a simulated graduation ceremony, and captured multiple takes of the entire performance, repositioning the camera between takes. Before a take, we pose our actors for a specific visual framing and autonomously place the camera. An editor used this footage to create a short narrative, cutting between different angles as the action smoothly unfolds. This use case demonstrates how our system can be used as part of the traditional cinematography process.

Imposing Safety Constraints Our decision to prioritize safety causes some failure cases where we do not manage to frame a subject accurately. These failure cases occur when we attempt to capture a close shot from far away. These situations are particularly challenging, since small errors in orientation can significantly impact the visual composition. Internal shots are particularly sensitive to this effect. If the primary subject is looking at another subject, the internal has to be placed behind the other subject to capture the face of the primary subject. Figure 5.10 shows an example of this occurrence. The internal shot is cropped significantly, and the resulting footage does not manage to respect the rule of thirds.

Screen-Space Impact of Accurate Tracking and Control We investigate the screen space impact of using our RTK GPS-based tracking system over conventional GPS for

Figure 5.10: A failure case, where the suggested crop does not match the target framing. $\alpha = 14.9$ while $\alpha_{\max} = 50$.

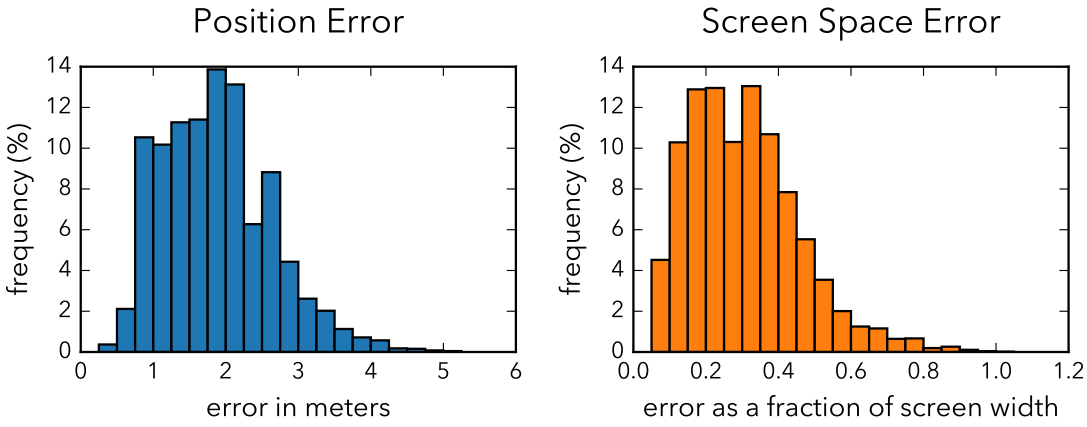

Figure 5.11: World space and screen space error incurred when using conventional GPS to track subjects and plan shots. We track subjects through an 8 minute session using both RTK and conventional GPS. We use RTK GPS as ground truth, and conventional GPS to plan shots. Conventional GPS produced world space error of several meters, potentially violating our safety constraint. We automatically planned a virtual camera shot every 4 seconds, and report the resulting screen space error. Using conventional GPS incurs unacceptable screen space error, potentially placing the subject halfway across the frame or more. Furthermore, world space error is significant enough that the quadrotor can easily violate the safety sphere around our subjects.

shot-planning. We fed our system with position data from both a conventional and RTK GPS. We use RTK GPS as our ground truth measurement, and automatically plan each shot in our system using position information from both sources. We repeatedly planned a virtual camera shot every 4 seconds, over data recorded during all our real-world user trials. For this experiment, we assume we can place the virtual quadrotor at exactly the desired position, and render a virtual preview of the scene as it would appear given the ground-truth data. Thus, we can render an image that realistically places people at the correct location they would take in the frame if their position was measured with the conventional GPS. For every shot, we measure a world-space and a screen-space error: (1) We measure the error in screen space between the planned position of each subject and the actual position of each subject. (2) We measure the world-space error between the desired quadrotor placement when fed conventional versus RTK GPS data. We report the results of this test in Figure 5.11. On average, subjects appear approximately a third of screen-width away from their desired position. This is a significant error, given that a subject would, on

average, fall on the *wrong* rule-of-thirds line when using conventional GPS. Furthermore, the desired quadrotor position, even if we were to be able to place it perfectly, would fall on average approximately two meters away from its correct position. This amount of error makes it likely for the quadrotor to violate our safety spheres. Given the significant screen space and world space error incurred when using conventional GPS to capture our set of shots, we feel validated our design choice to use a higher accuracy approach.

5.9 Discussion

In our work, we consider the placement and size of subjects in screen space. However, we plan paths in world space, only indirectly controlling the screen space behavior of subjects. An exciting follow-up to our transition planner is an algorithm for directly controlling screen space behavior of subjects, solving for the equivalent camera path. In fact, since the publication of this work, a follow-up paper by Naegeli et al. [65] which investigates this problem has appeared!

Our tracking system in its current form is fairly bulky and intrusive. Our trackers are a prototype, and the assembly can easily be miniaturized. RTK GPS is also under active development, with modules becoming more robust and affordable, and base stations being offered as a cloud service. We expect that, in the near future, we can build this technology into wearable.

We chose a design of our tracking system that operates independently of the camera. An exciting path forward is to additionally control the quadrotor by visually tracking the primary subject whenever they are in frame.

In this instantiation of the Drone Cinematographer, we made a set of simplifying assumptions for the scenarios we considered. We limited ourselves to up to two subjects, and assumed the subject stays within a fixed safety sphere during filming. The next step for a more general Drone Cinematographer is to lift both these restrictions.

Currently our system does not attempt to aggressively follow or respond to people's movement. We are interested in extending our system to capture moving versions of our static shots while maintaining the safety of our subjects. Given the tight framing of our shots, we imagine that doing so is a nontrivial problem, potentially addressed using concepts from model predictive control [81].

In this chapter we considered the composition of shots. There are also many other factors that play into producing aesthetic footage, such as lighting and color. Broadly speaking, we think that incorporating aesthetic considerations into quadrotor camera control can significantly alter the way people produce video.

Chapter 6

Conclusion

In this thesis, we described new tools for creating cinematography with quadrotors. Whereas current approaches present pilot-first interfaces rooted in aviation and mission planning, our approach presents *cinematography-first* interfaces grounded in 3D animation and visual composition principles. By reifying concepts from virtual and traditional filmmaking into tools, we enabled users to specify their cinematic intent directly, while we automated flying the quadrotor camera.

In Chapter 3, we introduced Horus, a tool for designing quadrotor camera shots. Horus is built on a set of design principles informed by 3D animation and exploratory studies of professional quadrotor cinematographers. Specifically, Horus is built around four concepts: (1) designing shots visually using keyframes; (2) preview shows virtually; (3) precisely specify shot timing using easing curves; and (4) visually receive feedback on the feasibility of a shot. Using Horus, both novice and expert users designed compelling shots that would be challenging to create otherwise. We then autonomously captured these shots on a real quadrotor camera platform.

Users of Horus were limited to preplanning shots of large static scenes, where the meter-plus positioning accuracy of standard GPS and the lack of object tracking did not prevent us from faithfully capturing shots. In Chapter 4 we addressed these limitations by examining RTK GPS—a technique that provides centimeter-accurate absolute positioning. Specifically, we quantified the performance of the first consumer-grade single-frequency RTK GPS as an accurate sensor for quadrotor localization. We first demonstrated centimeter-precise and accurate 3D position measurement using the raw sensor. We then integrated the RTK

GPS receiver into a quadrotor and found that RTK GPS provides decimeter accurate position control. These results led us to conclude that cheap, consumer-grade RTK GPS sensors are a viable approach for accurate object tracking and quadrotor localization in large outdoor environments.

Armed with this sensor, we tackled the problem of filming people using quadrotors. In Chapter 5, we presented the Drone Cinematographer. This tool is built on well-known composition principles, enabling users to capture footage of people autonomously with a quadrotor. We encoded a set of canonical shots from cinematography literature, and adapted them to respect safety considerations imposed by quadrotors. We presented a novel transition planner that produces visually pleasing transitions, while also satisfying our safety constraints and quadrotor dynamics. Our implementation is built on a tracking system that uses RTK GPS to localize the positions of both subjects and the quadrotor camera with centimeter accuracy. Using the Drone Cinematographer, we successfully captured multiple scenarios with reasonable accuracy using a real quadrotor camera.

6.1 Impact

Several exciting new developments have taken place since we started publishing the work in this thesis. Researchers at ETH Zurich have presented two systems that also provide new interfaces and algorithms for quadrotor cameras [65, 32]. Researchers at Stanford recently extended Horus to generate feasible variants of infeasible trajectories automatically [76]. Coaguila et al. [21] demonstrated a method to capture well-composed video of a single subject using a quadrotor camera by visually tracking their face.

The consumer industry is also starting to investigate tools that provide cinematography-first interfaces. In fact, the largest consumer quadrotor manufacturer, DJI, now provide autonomous "follow-me" and "active track" features that visually track and attempt to keep a subject in view [27]. These methods, although still in infancy, are enabling operators to focus on cinematography rather that manually piloting a quadrotor.

6.2 Final Remarks

The tools presented in this thesis are the first steps towards an exciting new field - that of autonomous flying robots enabling compelling content creation. We fundamentally view today's quadrotors as computational cameras, enabling algorithmic and interaction advances to support creativity. For example, quadrotor cameras might soon be able to autonomously film a person skiing down a mountain, intelligently avoiding obstacles and framing the subject as they move through complex environments. We see a future where robotic systems intelligently balance human interfaces and aesthetic and technical knowledge, enabling you to capture your stories while you remain immersed in your experience.

Appendix A

Quadrotor Camera Model Details

A.1 Deriving the Quadrotor Camera Manipulator Matrices

In this section, 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* [46], 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* [46], 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 $25\times$ 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,

$$m\ddot{\mathbf{p}} = \mathbf{f}_e + \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_{\mathbf{f}}\mathbf{u}_q \tag{A.1}$$

where m is the mass of the quadrotor camera; $\mathbf{f}_e$ is the external force; $\mathbf{M_f}$ 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,

$$\mathbf{M_f} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{A.2}$$

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,

$$\mathbf{I}_q\dot{\omega}_q + \omega_q \times \mathbf{I}_q\omega_q = \mathbf{M}_\tau\mathbf{u}_q \tag{A.3}$$

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,

$$\mathbf{M}_\tau = \begin{bmatrix} ds_\alpha & ds_\beta & -ds_\beta & -ds_\alpha \\ \gamma & -\gamma & \gamma & -\gamma \\ -dc_\alpha & dc_\beta & dc_\beta & -dc_\alpha \end{bmatrix} \tag{A.4}$$

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 constant 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* [81] 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* [81] control input applied at the gimbal, $\mathbf{u}_g$, as follows,

$$\dot{\omega}_g = \mathbf{u}_g \tag{A.5}$$

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,

$$\begin{aligned}\mathbf{R} &= \mathbf{R}_{\mathbf{y}}^{\psi}\mathbf{R}_{\mathbf{z}}^{\theta}\mathbf{R}_{\mathbf{x}}^{\phi} \\ &= \begin{bmatrix} c_\psi & 0 & s_\psi \\ 0 & 1 & 0 \\ -s_\psi & 0 & c_\psi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\theta & -s_\theta \\ 0 & s_\theta & c_\theta \end{bmatrix} \begin{bmatrix} c_\phi & -s_\phi & 0 \\ s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}\end{aligned} \tag{A.6}$$

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,

$$\mathbf{A}\dot{\mathbf{e}} = \omega \tag{A.7}$$

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

$$\dot{\mathbf{A}}\dot{\mathbf{e}} + \mathbf{A}\ddot{\mathbf{e}} = \dot{\omega} \tag{A.8}$$

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,

$$\begin{aligned}\mathbf{A} &= \begin{bmatrix} c_\psi & 0 & s_\psi c_\theta \\ 0 & 1 & -s_\theta \\ -s_\psi & 0 & c_\psi c_\theta \end{bmatrix} \\ \dot{\mathbf{A}} &= \begin{bmatrix} -s_\psi \dot{\psi} & 0 & -s_\psi s_\theta \dot{\theta} + c_\psi c_\theta \dot{\psi} \\ 0 & 0 & -c_\theta \dot{\theta} \\ -c_\psi \dot{\psi} & 0 & -s_\psi c_\theta \dot{\psi} + s_\theta c_\psi \dot{\theta} \end{bmatrix}\end{aligned} \tag{A.9}$$

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

Relating the Quadrotor's Orientation to the Control Inputs We can rotate equations (7) and (8) into the body frame of the quadrotor, and substitute them into equation (3), to get the following expression for the quadrotor's rotational dynamics,

$$\begin{aligned}\mathbf{I}_q(\mathbf{R}_{\mathcal{Q},\mathcal{W}}\dot{\mathbf{A}}_q\dot{\mathbf{e}}_q + \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\ddot{\mathbf{e}}_q) + \\ \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\dot{\mathbf{e}}_q \times \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\dot{\mathbf{e}}_q = \mathbf{M}_\tau\mathbf{u}_q\end{aligned} \tag{A.10}$$

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 (8) into equation (5) to get the following expression for the gimbal's rotational dynamics,

$$\dot{\mathbf{A}}_g\dot{\mathbf{e}}_g + \mathbf{A}_g\ddot{\mathbf{e}}_g = \mathbf{u}_g \tag{A.11}$$

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,

$$\mathbf{q} = \begin{bmatrix} \mathbf{p} \\ \mathbf{e}_q \\ \mathbf{e}_g \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} \mathbf{u}_q \\ \mathbf{u}_g \end{bmatrix} \tag{A.12}$$

Based on this layout, we can express equations (1), (10), and (11) in manipulator form. In doing so, we get the following expressions for our quadrotor camera manipulator matrices,

$$\begin{aligned}
\mathbf{H}(\mathbf{q}) &= \begin{bmatrix} m\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{A}_g \end{bmatrix} \\
\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) &= \\
&\begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\dot{\mathbf{A}}_q - \left(\mathbf{I}_q\mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q\dot{\mathbf{e}}_q\right)_\times \mathbf{R}_{\mathcal{Q},\mathcal{W}}\mathbf{A}_q & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \dot{\mathbf{A}}_g \end{bmatrix} \\
\mathbf{G}(\mathbf{q}) &= \begin{bmatrix} -\mathbf{f}_e \\ \mathbf{0}_{3\times1} \\ \mathbf{0}_{3\times1} \end{bmatrix} \\
\mathbf{B}(\mathbf{q}) &= \begin{bmatrix} \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f} & \mathbf{0}_{3\times3} \\ \mathbf{M}_\tau & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times4} & \mathbf{I}_{3\times3} \end{bmatrix}
\end{aligned} \tag{A.13}$$

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.

A.2 Proof that B is Always Full Column Rank

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

$$\mathbf{M} = \begin{bmatrix} \mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f} \\ \mathbf{M}_\tau \end{bmatrix} \tag{A.14}$$

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 $\mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f}$ 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 $\mathbf{R}_{\mathcal{W},\mathcal{Q}}\mathbf{M}_\mathbf{f}$. Representing $\mathbf{r}$ as a linear combination of the rows in $\mathbf{M}_\tau$, we get the following system of equations,

$$\begin{aligned} r = &\ ids_\alpha &&+ j\gamma &&- kdc_\alpha \\ r = &\ ids_\beta &&- j\gamma &&+ kdc_\beta \\ r = &\ -ids_\beta &&+ j\gamma &&+ kdc_\beta \\ r = &\ -ids_\alpha &&- j\gamma &&- kdc_\alpha \end{aligned} \tag{A.15}$$

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.

Bibliography

[1] FAA registered nearly 300,000 unmanned aircraft owners. *FAA*, Jan 2016. URL: https://www.faa.gov/news/press_releases/news_story.cfm?newsId=19914.

[2] 3D Robotics. IRIS+, 2014. URL: http://3drobotics.com/iris/.

[3] 3D Robotics. 3DR Radio Set, Aug 2015. URL: https://3dr.com/support/articles/207681193/3dr_radio_set/.

[4] 3D Robotics. Solo, 2015. URL: http://3drobotics.com/solo/.

[5] 3D Robotics. u-blox GPS with compass kit, Aug 2015. URL: https://3dr.com/support/articles/207681053/3dr_ublox_gps_with_compass_kit/.

[6] Hyunsang Ahn, Dohyung Kim, Jaeyeon Lee, Suyoung Chi, Kyekyung Kim, Jinsul Kim, Minsoo Hahn, and Hyunseok Kim. A robot photographer with user interactivity. In *Intelligent Robots and Systems (IROS)*, 2006.

[7] Ross Allen and Marco Pavone. A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance. In *Guidance, Navigation, and Control*, 2016.

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

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

[10] Graduate Assembly. Graduate Student Happiness and Well-Being Report, 2014. Accessed: 2017-04-10. URL: http://ga.berkeley.edu/wp-content/uploads/2015/04/wellbeingreport_2014.pdf.

[11] Abraham Bachrach, Samuel Prentice, Ruijie He, and Nicholas Roy. Range–robust autonomous navigation in gps-denied environments. *Journal of Field Robotics*, 28(5), 2011.

[12] 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.

[13] Jim Blinn. Where am I? What am I looking at? (cinematography). *IEEE Computer Graphics and Applications*, 8(4), 1988.

[14] Bruce Block. *The Visual Story: Creating the Visual Structure of Film, TV and Digital Media*. CRC Press, 2013.

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

[16] Brian Heater. 3D Robotics launches Iris quadcopter, for pre-assembled drone action. *Engadget*, Aug 2013. URL: `https://www.engadget.com/2013/08/19/3d-robotics-iris/`.

[17] Zachary Byers, Michael Dixon, Kevin Goodier, Cindy M Grimm, and William D Smart. An autonomous robot photographer. In *Intelligent Robots and Systems (IROS)*, 2003.

[18] Jason Campbell and Padmanabhan Pillai. Leveraging limited autonomous mobility to frame attractive group photos. In *International Conference on Robotics and Automation (ICRA)*, 2005.

[19] Pedro Castillo, Alejandro Dzul, and Rogelio Lozano. Real-time stabilization and tracking of a four rotor mini-rotorcraft. In *European Control Conference (ECC)*. IEEE, 2003.

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

[21] Rey Coaguila, Gita Sukthankar, and Rahul Sukthankar. Selecting vantage points for an autonomous quadcopter videographer. In *Florida Artificial Intelligence Research Society Conference*, 2016.

[22] Nicolas Courty, Fabrice Lamarche, Stéphane Donikian, and Éric Marchand. A cinematography system for virtual storytelling. In *Virtual Storytelling*, 2003.

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

[24] James Diebel. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors, 2006.

[25] DJI. DJI Go, 2015. URL: `http://www.dji.com/product/goapp`.

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

[27] DJI. Mavic Pro User Manual v1.6, Apr 2017. Accessed: 2017-05-16. URL: `https://dl.djicdn.com/downloads/mavic/20170428/Mavic+Pro+User+Manual+V1.6.pdf`.

[28] Raghudeep Gadde and Kamalakar Karlapalem. Aesthetic guideline driven photography by robots. In *International Joint Conference on Artificial Intelligence*, 2011.

[29] Quentin Galvane, Marc Christie, Chrsitophe Lino, and Rémi Ronfard. Camera-on-rails: Automated computation of constrained camera paths. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, MIG '15. ACM, 2015.

[30] Quentin Galvane, Marc Christie, Rémi Ronfard, Chen-Kim Lim, and Marie-Paule Cani. Steering behaviors for autonomous cameras. In *Proceedings of Motion on Games*. ACM, 2013.

[31] Quentin Galvane, Rémi Ronfard, Marc Christie, and Nicolas Szilas. Narrative-Driven Camera Control for Cinematic Replay of Computer Games. In *Motion In Games*, November 2014.

[32] Christoph Gebhardt, Benjamin Hepp, Tobias Nägeli, Stefan Stevšić, and Otmar Hilliges. Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI. ACM, 2016.

[33] 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.

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

[35] Mark Haigh-Hutchinson. *Real time cameras: A guide for game designers and developers*. Morgan Kaufmann Publishers Inc., 2009.

[36] Li-wei He, Michael F. Cohen, and David H. Salesin. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *SIGGRAPH*, 1996.

[37] Gabe Hoffmann, Dev Gorur Rajnarayan, Steven L Waslander, David Dostal, Jung Soon Jang, and Claire J Tomlin. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). In *Digital Avionics Systems Conference (DASC)*, volume 2. IEEE, 2004.

[38] 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 (SciVis)*, 19(12), 2013.

[39] John D Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3), 2007.

[40] Intel. Intel Galileo Gen 2 Board, Sep 2014. URL: `https://ark.intel.com/products/83137/Intel-Galileo-Gen-2-Board`.

[41] Elliot Kaplan and Christopger Hegarty. *Understanding GPS: Principles and Applications*. Artec House, 2006.

[42] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7), 2011.

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

[44] Myung-Jin Kim, Tae-Hoon Song, Seung-Hun Jin, Soon-Mook Jung, Gi-Hoon Go, Key-Ho Kwon, and Jae-Wook Jeon. Automatically available photographer robot for controlling composition and taking pictures. In *Intelligent Robots and Systems (IROS)*, 2010.

[45] 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.

[46] 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.

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

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

[49] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[50] J. Gordon Leishman. *Principles of Helicopter Aerodynamics*. Cambridge University Press, 2000.

[51] Katia Levecque, Frederik Anseel, Alain De Beuckelaer, Johan Van der Heyden, and Lydia Gisle. Work organization and mental health problems in PhD students. *Research Policy*, 46(4), 2017.

[52] Tsai-Yen Li and Xiang-Yan Xiao. An interactive camera planning system for automatic cinematographer. In *Multimedia Modeling*, 2005.

[53] Hyon Lim and Sudipta N Sinha. Monocular localization of a moving person onboard a quadrotor MAV. In *International Conference on Robotics and Automation (ICRA)*, 2015.

[54] Christophe Lino and Marc Christie. Intuitive and efficient camera control with the toric space. *Transactions on Graphics (SIGGRAPH)*, 34(4), 2015.

[55] Linx. SH Series GPS Antenna, Feb 2014. URL: `https://linxtechnologies.com/wp/product/sh-series-gps-antenna/`.

[56] 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.

[57] Sergei Lupashin, Markus Hehn, Mark W Mueller, Angela P Schoellig, Michael Sherback, and Raffaello DAndrea. A platform for aerial robotics research and demonstration: The flying machine arena. *Mechatronics*, 24(1), 2014.

[58] R. Mahony, V. Kumar, and P. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics Automation Magazine*, 19(3), Sep 2012.

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

[60] Lorenz Meier. Pixhawk Autopilot. URL: `https://pixhawk.org/modules/pixhawk`.

[61] Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *IEEE international conference on Robotics and automation (ICRA)*. IEEE, 2011.

[62] 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.

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

[64] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The grasp multiple micro-uav testbed. *IEEE Robotics & Automation Magazine*, 17(3), 2010.

[65] T. Naegeli, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges. Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization. *IEEE Robotics and Automation Letters*, PP(99), Feb 2017.

[66] Tayyab Naseer, Jurgen Sturm, and Daniel Cremers. Followme: Person following and gesture recognition with a quadrocopter. In *Intelligent Robots and Systems (IROS)*, 2013.

[67] Swift Navigation. Integrating Piksi with the Pixhawk platform, Jun 2015. URL: `http://docs.swiftnav.com/wiki/Integrating_Piksi_with_the_Pixhawk_platform`.

[68] NovaTel. GPS-701-GG Single-Frequency Pinwheel GNSS Antenna, Feb 2014. URL: `http://www.navtechgps.com/gps_701_gg_single_frequency_pinwheel_gnss_antenna/`.

[69] Michael Osborne. Mission Planner Overview. Accessed: 2017-04-10. URL: `http://ardupilot.org/planner/docs/mission-planner-overview.html`.

[70] Thomas Oskam, Robert W. Sumner, Nils Thuerey, and Markus Gross. Visibility transition planning for dynamic camera control. In *Symposium on Computer Animation (SCA)*, 2009.

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

[72] Ben Popper. Drone maker DJI nabs $75 million in funding at a $10 billion valuation. *The Verge*, May 2015. URL: `http://www.theverge.com/2015/5/6/8554429/dji-75-million-funding-investment-accel-10-billion-valuation`.

[73] Raymond W Prouty. *Helicopter Performance, Stability, and Control*. Krieger Pub Company, 1995.

[74] 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.

[75] Mike Roberts. Flashlight: A Python Library for Analyzing and Solving Quadrotor Control Problems, 2016. URL: `http://mikeroberts3000.github.io/flashlight`.

[76] Mike Roberts and Pat Hanrahan. Generating dynamically feasible trajectories for quadrotor cameras. *Transactions on Graphics (SIGGRAPH)*, 35(4), 2016.

[77] Michael Rubin. *The Little Digital Video Book*. Peachpit Press, Pearson Education, 2009.

[78] 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.

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

[80] Swift Navigation. Piksi Datasheet, 2013. URL: `https://www.u-blox.com/sites/default/files/NEO-M8P_DataSheet_%28UBX-15016656%29.pdf`.

[81] Russ Tedrake. Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832), 2014. URL: `http://people.csail.mit.edu/russt/underactuated/`.

[82] Philippe Terrier and Yves Schutz. How useful is satellite positioning system (GPS) to track gait parameters? A review. *Journal of Neuroengineering and Rehabilitation*, 2(1), 2005.

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

[84] u-blox. Neo-M8P Datasheet, Jan 2017. URL: `http://docs.swiftnav.com/pdfs/piksi_datasheet_v2.3.1.pdf`.

[85] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2), 2011.

[86] Peter Wayner. How it works; gyroscopes that don't spin make it easy to hover. *New York Times*, August 2002.

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

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