High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo

by

Andrew J. Barry

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2016

Author .....
Department of Electrical Engineering and Computer Science
February 1, 2016

Certified by .....
Russ Tedrake
X Consortium Associate Professor of Electrical Engineering and Computer Science, Aeronautics and Astronautics, and Mechanical Engineering
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Theses

High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo

by

Andrew J. Barry

Submitted to the Department of Electrical Engineering and Computer Science
on February 1, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis presents the design and implementation of a small autonomous unmanned aerial vehicle capable of high-speed flight through complex natural environments. Using only onboard sensing and computation, we perform obstacle detection, planning, and feedback control in realtime. We introduce a novel stereo vision algorithm, pushbroom stereo, capable of detecting obstacles at 120 frames per second without overburdening our lightweight processors. Our use of model-based planning and control techniques allows us to track precise trajectories that avoid obstacles identified by the vision system. We demonstrate a complete working system avoiding trees at up to 14 m/s (31 MPH). To the best of our knowledge this is the fastest lightweight aerial vehicle to perform collision avoidance in such a complex environment.

Thesis Supervisor: Russ Tedrake
Title: X Consortium Associate Professor of Electrical Engineering and Computer Science, Aeronautics and Astronautics, and Mechanical Engineering

Acknowledgments

A huge number of people helped me with this thesis. First, I would like to thank my advisor, Russ Tedrake, for providing an incredible, supportive, and exciting environment to build and learn about robotics, controls, and vision. Thank you, Russ, for supporting me when I decided to try my hand at vision processing, and thank you for the advice and encouragement in lab, on the running trail, and in life.

My thesis committee, Nick Roy and Bill Freeman, provided excellent advice on my thesis's direction and experiments. Thank you for your comments, questions, and advice.

The memebers of the Robot Locomotion Group have made my time at MIT especially fulfilling. I owe special thanks to Ani Majumdar, Pete Florence, John Carter, Andy Marchese, Tim Jenks, and Benoit Landry for helping me model, build, fly, and film the aircraft. Thanks to Joe Moore for the advice on my embedded electronics and for the early morning conversations that convinced me my slides were ready for group meeting.

I have been fortunate to collaborate with some incredible and generous people during my thesis work. In particular, I owe thanks to Helen Oleynikova and Dominik Honegger for risking their FPGA stereo system onboard my aircraft and to Jacob Izraelevitz for being amazing at teaching, logistics, and avoiding angry dogs with an ornithopter. The labs involved in the MURI program were a great inspiration and always had interesting things to say. My thesis would have been much more difficult without everyone I worked with in CSAIL, including Ron Wiken, Mieke Moran, Bryt Bradley, Mark Pearrow, Adam Conner-Simons, Abby Abazorius, and Kathy Bates. I would also like to acknowledge Dave and everyone at Westview Farms Creamery for allowing me to fly on your fields and for having amazing food, ice cream, and cider donuts. And thanks to Juan and Cindy for just being there.

Many of the parts of theis thesis were built with skills I learned from Dave Barrett and Mark Chang at Olin College. To everyone in the Olin community, thank you for giving me an amazing undergraduate education, an awesome group of lifelong friends, and for paying my tuition through all of it. To Rhino-SCOPE, you all made sure I was laughing and smiling

even if my airplanes were falling out of the sky.

My time at MIT would have been much less fun, vibrant, and gratifying without East Campus and Fourth East. I am so grateful for the memories, friends, food, and learning that I shared with you all.

To Katya, thank you so for your support, encouragement, advice, and love. You were with my through the successes and failures, tirelessly encouraging and supporting me. The best part of every day is when you come home.

To my sister Jenny, thank you for the advice, be it during snack time or while running on those bitterly cold days. No matter what, I could count on having a better answer to any problem after discussing it with you.

Finally, my mother Sue and my father Dan have always encouraged me to follow my dreams, be it by hiding a beam on an oscilloscope, looking at paramecium through a microscope, building Robodad, or flying faster than ever before. I can not imagine my life without you both.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents a system that is capable of high-speed flight (10-14 m/s or 22-31 MPH) around natural obstacles. The system is fully autonomous and performs all computation and perception on-board. We believe that flying near natural obstacles is *hard for the right reasons.* In other words, algorithms capable of guiding an unmanned aerial vehicle (UAV) through a thicket of trees have broad applicability throughout the fields of robotics, control theory, and computer vision. By choosing this task, we are focusing on basic research questions, not flight and vision specifics.

Concretely, a control system that can guide our aircraft through a thicket will have good properties such as scaling with dimensionality, applicability to unknown environments, and a realtime response rate. The systems must couple perception and control at high rate to react quickly and effectively enough to avoid obstacles. These are the same qualities required to guide a walking robot through rough terrain or an underactuated manipulator carrying a heavy load.

To further our goals, we use a fixed-wing aircraft, for practical reasons (speed and payload capacity) but also for research reasons, as the dynamics of fixed-wings are more rich than rotorcraft such as quadcopters. Our vision is to create aggressive, provably safe flight through challenging environments. This thesis describes sensing and control schemes capable of such flight, but does not show verification for those systems. We provide references for potential

verification methods the controllers, but verification of the vision system remains an open question.

1.1 Contributions

We present a novel stereo vision system capable of high-speed obstacle detection integrated with a trajectory-library based flight control system that is capable of avoiding complex natural obstacles at speed up to 14 m/s (31 MPH). The system is completely self-contained on a small 34 inch wingspan, 664 gram aircraft, using only onboard sensing and onboard computation while in flight. To the best of our knowledge, this is the first system weighing under 1 kg capable of flying through these complex environments at such speed without relying on external sensors or computation.

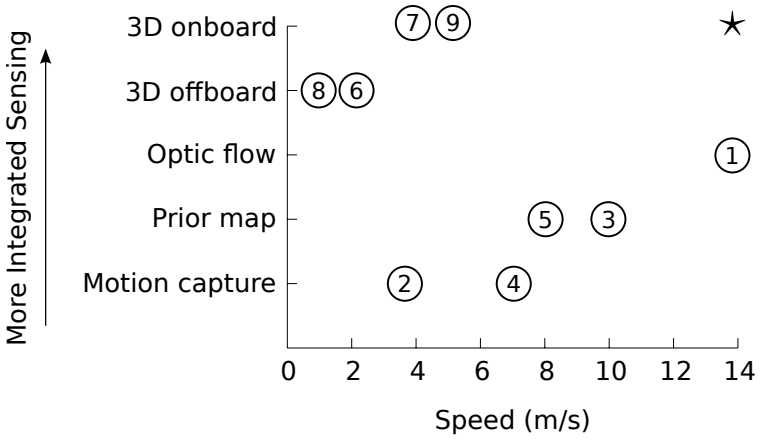
1.2 Related Work

There is a large body of work in the UAV field, ranging in scale from airliners to bumblebees and in autonomy from radio-controlled to fully automatic. Here we discuss the most relevant works for obstacle avoidance.

1.2.1 Obstacle Avoidance with Micro Aerial Vehicles

Small electric vehicles weighing under 5kg (often called MAVs) are an area of current interest, delivering platforms that are safer and can operate in more cluttered locations than larger UAVs, while also retaining enough payload to perform complex tasks. Stabilization and non-aggressive maneuvering of these small vehicles is relatively well understood in open flight environments. Simple PID (proportional, integral, derivative) controllers implemented on low cost commodity hardware[1] demonstrate good stabilization and waypoint navigation performance on these systems.

[1] 3D Robotics, Inc. (`http://3drobotics.com`,) among others, provide platforms with this capability.



⋆:	This thesis	–
1:	Beyeler, 2009	[12]
2:	Mellinger, 2010	[71]
3:	Bry, 2012	[16]
4:	Barry, 2012	[8]
5:	Richter, 2013	[86]
6:	Ross, 2013	[91]
7:	Shen, 2013	[98]
8:	Dey, 2015	[26]
9:	Oleynikova, 2015	[83]

Figure 1-1: Plot of related work in small UAV obstacle avoidance examining speed and sensor integration. The ⋆ indicates this thesis and labels are in order of publication year. Despite varying flight hardware, we believe speed is a good comparison metric because the systems are limited by sensing and computation as much as wingspan or body-length.

Effectively avoiding obstacles, however, remains an open question. In pursuit of this goal, researchers have used a wide variety of sensors, such as external motion capture, monocular vision, stereo vision, and LIDAR to detect potential dangers. Figure 1-1 maps some of the related work in the field based on sensor integration and speed.

In Motion Capture Environments

In motion capture environments, rotorcraft and fixed wings have shown impressive obstacle avoidance capabilities, demonstrating accurate dynamics models and good trajectory tracking. Abbeel's work on learning autonomous helicopter maneuvers [1] was an early demonstration of these techniques. Mellinger and Kumar demonstrate quadrotor flight through small openings [71] and in formation [72, 52]. Quadrotors have further been shown to perform agressive flips [65] and interact with objects including catching and juggling a ball [14, 78], balancing a pendulum [39], cooperatively throwing a ball [87], and building structures [63]. Variable pitch quadrotors have shown improved performance over these systems at the cost of further complexity [23].

The dynamics of fixed wing flight have proven more difficult than quadrotors for similarly dramatic maneuvers. On these systems, Sobolic showed automatic transition between hover

and level flight [103] and Cory demonstrated autonomous perching on a glider, landing on a wire with impressive 5cm accuracy [22]. We have previously demonstrated flight through a gap smaller than the aircraft's wingspan [7].

Systems that have a Known Map

While impressive, the work described above relies on sets of well-calibrated cameras that localize all relevant vehicles and objects in a relatively small volume. Performing these maneuvers outside of motion capture environments remains a substantial barrier to the practical use of these systems and algorithms.

Without motion capture, but using a prior map of the environment, Bry and Richter demonstrate localization and map-based obstacle avoidance on fixed-wing and quadrotor vehicles, respectively [16, 86]. Their work relies on a laser rangefinder to localize the aircraft in flight, allowing it to use a prior map on the environment. Should obstacles appear between their offline mapping and flight phases, the aircraft may collide.

Using a magnetometer to sense a powerline, Moore has extended Cory's work to onboard sensing of the landing wire. He has successfully demonstrated perching without the help of motion capture [74, 76].

1.2.2 Vision-Based Techniques for MAVs

The payload of MAVs severely limits researchers' choices for obstacle sensing. Planar laser rangefinders (LIDAR) are heavy relative to the payload of small wingspan aircraft and give insufficient data for three-dimensional obstacle detection. Other active rangefinders, such as the Microsoft Kinect[2], currently do not work outdoors, limiting their usefulness to indoor navigation tasks. For these reasons, the field has focused lightweight cameras and computer vision algorithms.

[2]Microsoft, Inc. `https://dev.windows.com/en-us/kinect`

Stereo Vision Techniques

We are by no means the first to consider stereo vision on small aircraft. For obstacle detection, Byrne et al. show that augmented stereo can work on embedded flight systems [17]. Their approach is to use an image segmentation technique to reduce false-positive detections from correspondence alone. However, like many existing techniques, they are able to run their system at only 5-10Hz, substantially slower than is required for fast flight near obstacles. Hrabar presents a technique utilizing both optical flow and stereo, demonstrating that the combination can outperform each individually [43]. Yang and Pollefeys, among others, implemented stereo on GPUs for significant processing gains [115]. They also suggest a method that eliminates the need for per-frame rectification, which further increases efficiency and framerate.

Meier et al. integrates stereo vision based obstacle detection with a full suite of IMU and vision-based state estimation, and autonomous waypoint navigation [69]. They do not show, however, fast navigation around obstacles as they are limited by the rate of their stereo processing (15 fps) and only attempt to notify higher level systems of obstacles. Goldberg demonstrates stereo processing at relatively high framerate using similar processors and cameras to this thesis, but is still limited to 46 fps [35]. Recent work an ultra-light flapping UAV demonstrated a 4.0 gram line-based stereo algorithm running at up to 40 Hz at 128x32 on a 168 Mhz embedded processor [25]. Honegger et al. show that stereo and optical flow matching is possible using small, flight ready FPGA (Field Programmable Gate Array) hardware at similar rates to our system (376x240 at 127 fps or 640x480 at 60 fps) [42, 41] with their latest work demonstrating onboard obstacle avoidance at 5 m/s [83].

Monocular Vision

Monocular vision techniques are attractive because of their simplicity, low cost, payload weight, and availability. In 2005, Michels et al. demonstrated a learning algorithm for high-speed obstacle avoidance on a radio controlled car. While they were only able to process images at 7 Hz, they found relatively low fatal errors rates (around 2%) and thus were

able to make the car navigate autonomously at 5 m/s in cluttered environments [73]. More recent work has shown success learning range classifiers and computing depth in realtime offboard. In Dey et. al.'s latest work they present flights over 100 meters in cluttered wooded environments [26].

Optic Flow

Optical flow techniques work well, controlling stable flight, takeoff, landing [5, 11], and obstacle avoidance [12, 116]. They allow for fast flight, have incredible framerate when using dedicated sensors (up to 4,500 Hz) and the modules are lightweight, self-contained, and low power. Unlike many of the other techniques described here, optic flow has been successful enough to see commercialization on MAVs[3]. For our purposes, however, their limited resolution cannot percieve the complex geometries we want to navigate, such as deliberately flying close to certain obstacles, maneuvering to avoid particular geometries, or approaching and flying through small gaps.

Visual Simultaneous Localization and Mapping (VSLAM)

No discussion of robotic navigation is complete without SLAM. SLAM allows a robot to build a map of its environment and subsequently use that map for obstacle avoidance. Many authors have suggested using VSLAM [24, 38, 100] on MAVs [50]. There are a few fundamental problems with VSLAM on MAVs, notably: (1) computational power and (2) robustness of the navigation solution. Lee et al. attempted to address the first by adding a plane constraint to the system, but this limits the vehicle's navigation to at least somewhat structured environments [58].

Recently, researchers have begun using parallel tracking and mapping (PTAM) [51] as an alternative to traditional SLAM methods. PTAM separates tracking and mapping, allowing the mapping component to run below framerate while tracking runs at framerate, reducing the pressing computational load. Tracking large maps, however, continues to be

[3] senseFly Ltd., `http://sensefly.com`

computationally expensive.

Visual Odometry

Visual odometry is useful to track position and orientation of robotic vehicles, especially in GPS-denied environments. In our case, visual odometry could be used to compute an estimate of the wind speed, which without GPS, is difficult to measure. Early visual odometry work started on ground vehicles, showing promise with low error rates and reasonable framerates [82]. [95] gives a good overview of the techniques and development of the field. Recent work with similar goals to PTAM has shown fast tracking using the same processors used in this thesis. Semi-direct visual odometry (SVO) uses a monocular camera to track features and build a map in realtime [31]. SVO, like PTAM, relies on a usually downward facing camera and texture below the MAV. Both systems are not yet tested at the speeds we fly, but show promise for monocular visual odometry approaches.

Arguing for Fast Vision

Using photorealistic ray-traced synthetic images, Handa et al. directly investigate the advantages of high framerate systems. In their 3D-tracking experiment with perfect lighting, they show substantial benefits from increasing rates up until approximately 100Hz, beyond which tracking continues to improve but at a less substantial rate [37]. This work encourages us to attempt these rates on our airframe. Recently, Srinivasan et al. described an algorithm for egomotion using high frame rate (120 fps) processing [105]. Murali and Birchfield suggest that vision data is highly redundant and that low resolution representations may allow for fast processing with a limited reduction in performance [79]. The types of features they extract, however, are limited in scope and are not yet capable of the detection performance required for our task.

1.2.3 For Stabilization and Localization

Researchers have been using vision-based techniques for stabilization of MAVs for over a decade [88]. Approaches range from insect-inspired non-metric [81], optical flow, and angular-rate based techniques [104] to involved systems integrating monocular images and IMU data [29, 93].

Vision-aided Inertial Navigation Systems (VINS)

Most monocular vision systems integrate inertial sensors from an on-board inertial navigation system (INS) for position estimation [114, 20, 84]. These systems can benefit from predicting feature locations in images via inertial measurement unit (IMU) sensors, reducing the search space for features. Shen et al. have presented a combined fast-monocular and slow-stereo VINS system to produce stable state-estimates for a slow moving quadrotor [99]. Their monocular camera system runs at 25 Hz, with the stereo augmenting the state-estimates at 1 Hz. Li et al. have recently demonstrated excellent tracking using online calibration techniques and commodity phone hardware [60, 61]. They demonstrate tracking within 0.5-0.8% of distance traveled for substantial distances.

1.2.4 Avian Flight

The first question one might ask when attempting this task, is "can it be done *fast*?" Birds prove that high-speed, dynamic flight through dense obstacles is possible. Lin et al. show that pigeons are fully capable of dynamic flight through dense obstacles, easily maneuvering to avoid closely spaced objects [62]. Biologists have given some suggestions on nature's techniques. For example, Carruthers et al. suggest that birds may use sensing spread across their wings to stabilize their maneuvers. She suggests that eagles are using a leading-edge flap [19] and fast tail movement [109] during unsteady flight such as takeoff, gust response, and landing. Furthermore, eagles may use inputs from their covert feathers to sense the position of flow separation, degree of stall, and instantaneous lift coefficient [18].

1.2.5 Large and Ultra-Small UAVs

Control of large (5-20kg) UAVs is a topic of active research that has succeeded in developing solutions for autonomous take-off, landing [94, 46], aerobatics [34], navigation, map-making [47], and obstacle avoidance [96]. In general, the field has had greater success with larger UAVs, buoyed by their ability to carry substantially more sophisticated sensors and computers.

Alternatively, researchers have developed small (less than 25 grams) UAVs that are capable of absorbing impacts and even exploring environments using collisions [30]. These systems typically use optical flow sensing with simple controllers, limited by their tiny computation capacity.

1.2.6 Planning Algorithms

Even given some notion of where obstacles are, planning a feasible path through obstacles in real time remains a difficult task. Valid trajectories must be feasible for our aircraft to execute, transition from the previous plan to the new one seamlessly, and, of course, avoid impact. All computation must happen fast enough that we can perform it onboard and keep up with high-speed flight.

This problem is well studied in the literature and a number of solutions offer good balances of computational cost and performance. Simple systems use potential field methods which model obstacles as electrical charges imparting a "force" on the robot [53]. These systems suffer from becoming stuck in local minimums, rely on another system to keep the plane in flight (they often do not reason about the flight dynamics in three dimensions), and usually consider the obstacles and robots as points, ignoring contact geometries.

Shooting and Collocation Methods

One might consider an exhaustive search over the state and action space of the vehicle to determine an optimal path given some cost function. For our application, the state space of the aircraft (at least 12-dimensional) makes this search prohibitively expensive even when

using dynamic programming methods [27]. Shooting methods do not perform an exhaustive simulation, but instead simulate the system forward (usually a small amount) to find a good trajectory [13]. In contrast, collocation methods force the optimizer to find both a feasible state and action set, as well as minimize a cost function. Specifically, direct collocation uses cubic polynomials between discretized knot points to parameterize the trajectories [107].

Probabilistic Roadmaps (PRMs)

Probabilistic roadmaps (PRMs) generate trajectories by first, offline, building many paths through state space. After generating paths that transit a substantial portion of the state space, the PRM can transition to online operation. In the online mode, new initial and goal states are queried. The PRM attempts to run a small, fast optimization to connect the initial state to its graph of trajectories. It then performs another small optimization to branch from the existing paths to the final state [49]. In this way, the PRM can connect states that are far apart by using its existing "highways" to speed up the computation. PRMs can, however, be difficult to apply to nonholonomic systems because they assume an undirected graph for motions through state space [108]. The computation becomes more expensive, although not infeasible, when one requires directed routes.

Rapidly-Exploring Random Trees (RRTs)

RRTs randomly explore state space through small extend operations that add to an existing tree [56]. The algorithm is biased towards expansion into unexplored areas and is not hindered by nonholonomic systems or complex dynamics. The field has seen a plethora of expansions on the idea, including RRT*, an RRT which rewires its internal structure to achieve asymptotic optimality [48].

Trajectory Libraries

Another method for trajectory planning is to string together a pre-built library of small trajectories. For example, one might have a library consisting of "fly straight," "turn left,"

and "turn right." At each time step, we choose between those options to determine where to fly next. Atkeson first suggested using local trajectories connected together to reduce the computational load of dynamic programming [2]. Later, he demonstrated that idea used in a learning context for planning with humanoids [3] and again on real hardware for navigating a marble maze [106]. In the marble maze example, the fundamental obstacles for robust implementation are clear: trajectory generation, choice of distance function, and dependency on model performance.

Frazzoli et. al give a detailed description of a library-based control system, showing that it can be computationally efficient and stable [32]. They use tools from hybrid systems to switch between maneuvers and trim trajectories, where the first are motion plans and the second are stable regions of state space. Bachrach suggests using bundles of trajectories as a way to represent the environment in addition to using them for control [4].

Transitioning between trajectories in libraries safely can be a difficult task. Verification of controllers can guarantee safe transitions by ensuring that the next controller is capable of stabilizing the ending state of the previous controller [68]. In recent work, Majumdar has shown a full working system that uses these guarantees to aggressively avoid obstacles [67].

Differentially Flat Systems

Differential flatness has become popular as a way to reduce the computational load of creating trajectories. A system is differentially flat if one can find a set of *flat outputs* from which the full state and input vectors can be determined without integration [57, 102]. More formally, given a system of the form $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ with $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{u} \in \mathbb{R}^m$, one can find flat outputs $\mathbf{y} = h(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, \ddot{\mathbf{u}}, ..., \mathbf{u}^{(k)})$ such that the following $\mathbf{g}$ and $\mathbf{g}'$ exist:

$$\mathbf{x} = \mathbf{g}\left(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}, ..., \mathbf{y}^{(j)}\right)$$

and

$$\mathbf{u} = \mathbf{g}'\left(\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}, ..., \mathbf{y}^{(j)}\right).$$

Mellinger and Kumar demonstrated trajectory generation using this method in their well-received work on minimum snap trajectories [70]. The primary drawback of these systems is that for underactuated plants, such as an aircraft or quadrotor, the trajectories cannot specify all the state variables. For example, in [70], the flat outputs are the x, y, z positions and the yaw angle ψ. The pitch and roll of the vehicle cannot be specified using that controller, which can make close-obstacle flight, like a knife-edge maneuver, difficult or impossible to perform without switching between controllers.

1.2.7 System Identification

System identification, or the creation and identification of a model of the system dynamics, can enable a large variety of powerful feedback control techniques. We identify a model based on [103], integrating the flat plate dynamics from [22]. While there are a variety of techniques for selecting parameters (see [45] or [112] for an overview), we use a gray-box technique based on [64].

1.2.8 Feedback Control Algorithms

Open-loop trajectories or planned paths are insufficient for flight because they are, in general, unstable. Modeling errors, wind gusts, and other disturbances will cause our aircraft to deviate from the planned trajectory. Online, we must use feedback control to track our plans.

There is a substantial body of control literature, but here we focus on nonlinear systems. Our system is fundamentally nonlinear because, for example, as angle-of-attack (AOA) increases, the wing will generate more lift until flow separation causes a stall. At that point, increased AOA will not continue to increase lift. We want to be capable of generating plans that push the performance envelope into these aggressive, unstable, and difficult to model regions.

Linear Quadratic Regulators

The linear quadratic regulator (LQR) is a well-known technique for performing optimal control with a linear dynamics model and quadratic cost [59]. Many of the control tasks described above, such as [1, 87, 6, 7, 21] use LQR or variations of it.

One such variation, time-varying LQR (TVLQR) involves taking Taylor approximations along an open-loop trajectory to generate a linear time-varying dynamics model along the trajectory [111]. Using those approximations, LQR can be applied to nonlinear systems about known trajectories.

Model Predictive Control

In model predictive control (MPC), we are given a model of a system and a reference trajectory. We then run an optimization program, the type of which varies by model class and robustness requirements, to generate (often optimal) inputs for the system. After applying the first input to the plant, we observe the real system's response and repeat the simulation and optimization process at time $t + 1$ [33]. MPC can have good robustness properties [10] and while linear MPC can be fast (problems of our size[4] can be solved in under 5ms) [113], non-linear MPC remains intractable in many situations. Some work such as [101] suggests that simplifications like linearization around a reference trajectory can make non-linear MPC more tractable.

Robust Control Methods

H-infinity methods use the H_∞ norm to characterize the worst-case scenario for a system with uncertainty:

$$||G(s)||_\infty = \max_\omega \bar{\sigma}(G(j\omega))$$

where $\bar{\sigma}$ denotes the largest singular value of the frequency response and $G(j\omega)$ is a frequency response description of our system [9]. Thus, we find the maximum frequency response on the largest singular value, characterizing the most vulnerable portion of the response to

[4] Our problems include about 12 states and three control inputs.

uncertainty. If we are sure that the system is stable in this case to a bounded uncertainty, then we are guaranteed that no other part of the system will have a less-stable (worse) response to uncertainty. In this way, for the linear case, we can build systems that are sure to be stable under bounded uncertainty by analyzing the worst case and ensuring that our bounded uncertainty is insufficient to cause instability.

Gain Scheduling

Gain scheduling aims to combine multiple controllers, often of the same type, with different gains at different operating points. One picks a (slow moving) "scheduling variable" that determines when to switch between the different gains [97]. This type of system can provide substantial flexibility, easily combining multiple linear regions. With sophisticated controllers, gain scheduling can also provide rich feedback control.

1.2.9 Work Avoiding Trees

10 years ago, Langelaan and Rock attempted to build a flight system through a forest, but were only able to perform experiments on a ground vehicle in an artificial forest [53]. They attempted to use a vision-based bearing only (no range estimates) SLAM technique. The system's map building demanded substantial computation that made attempting the experiments on flight vehicles infeasible [55, 54].

Roberts et al. have a recent take on flying through forests, estimating tree location via vision. Their system relies on tree trunks to be vertical, potentially limiting the system to relatively tame forests. They use optical flow to predict what portions of the images are close to the vehicle and are therefore important to process immediately [89]. The most successful work to date is Dey's recent monocular vision system that estimates depth from a variety of monocular features. They cite over 100 meter trials and present over 1 km of flight data [26].

1.3 Thesis Organization

Chapter 1 introduces related work, including techniques for control, vision, and trajectory generation. We note similar hardware platforms used, the sensors acquired, and other technqiues for flying near natural and man-made obstacles. In Chapter 2, we introduce a novel stereo vision algorithm, pushbroom stereo, that is capable of detecting obstacles at high framerate with a small, lightweight CPU. We detail the algorithm, present flight results, and compare to another stereo system implemented on an FPGA. Chapter 3 presents our autopilot mechanism, complete with online planning, feedback control, and high-level control. Chapter 4 presents results from the system flying near, and avoiding, natural obstacles. We conclude with a discussion of the results and note future directions in Chapter 5.

Chapter 2

Pushbroom Stereo for High-Speed Obstacle Detection

Our system necessitates fast vision. On a standard 30 frames-per-second (fps) system, traveling at 14 m/s means that with a detection horizon of 10 meters, we would have 21 frames to detect and avoid an obstacle. Based on these numbers, we decided to pursue fast vision, with a goal of capturing and processing frames in less than 10ms (100+ Hz.)

2.1 Block-Matching Stereo

A standard block-matching stereo system produces depth estimates by finding pixel-block matches between two images. Given a pixel block in the left image, for example, the system will search through the epipolar line[1] to find the best match. The position of the match relative to its coordinate on the left image, or the disparity, allows the user to compute the 3D position of the object in that pixel block.

[1]Standard calibration and rectification techniques provide a line, called the epipolar line, on which the matching block is guaranteed to appear.

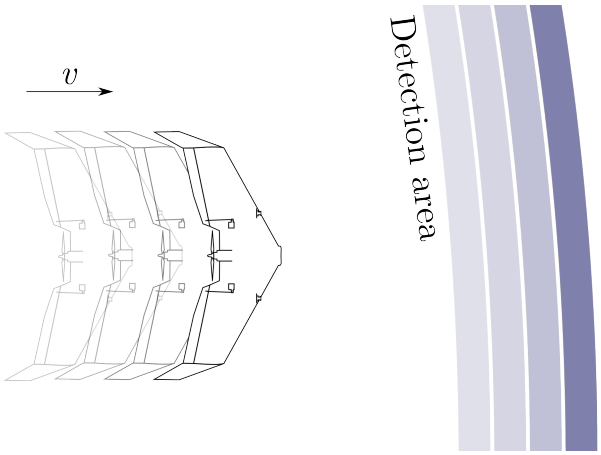


Figure 2-1: By detecting at a single depth (dark blue) and integrating the aircraft's odometry and past detections (lighter blue), we can quickly build a full map of obstacles in front of our vehicle.

2.2 Pushbroom Stereo

One can think of a standard block-matching stereo vision system as a search through depth. As we search along the epipolar line for a pixel group that matches our candidate block, we are exploring the space of distance away from the cameras. For example, given a pixel block in the left image, we might start searching through the right image with a large disparity, corresponding to an object close to the cameras. As we decrease disparity (changing where in the right image we are searching), we examine pixel blocks that correspond to objects further and further away, until reaching zero disparity, where the stereo base distance is insignificant compared to the distance away and we can no longer determine the obstacle's location.

Given that framework, it is easy to see that if we limit our search through distance to a single value, d meters away, we can substantially speed up our processing, at the cost of neglecting obstacles at distances other than d. While this might seem limiting, our cameras are on a moving platform (in this case, an aircraft), so we can quickly recover the missing depth information by integrating our odometry and previous single-disparity results (Figure 2-1). The main thing we sacrifice is the ability to take the best-matching block as our stereo match; instead we must threshold for a possible match.

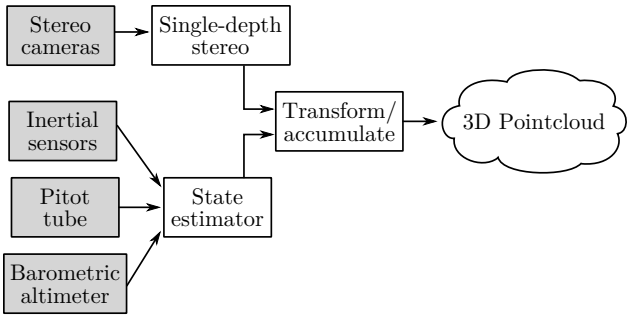


Figure 2-2: Pushbroom stereo overview. Note that the system does not require GPS.

We give this algorithm the name "pushbroom stereo" because we are "pushing" the detection region forward, sweeping up obstacles like a broom on a floor (and similar to pushbroom LIDAR systems [80]). We note that this is distinct from a "pushbroom camera," which is a one-dimensional array of pixels arranged perpendicular to the camera's motion [36]. These cameras are often found on satellites and can be used for stereo vision [40].

2.3 Odometry

Our system requires relatively accurate odometry over short time horizons. This requirement is not particularly onerous because we do not require long-term accuracy like many map-making algorithms. In our case, the odometry is only used until the aircraft catches up to its detection horizon, which on many platforms is 5-10 meters away. We demonstrate that on aircraft, airspeed measurement (from a pitot tube) is sufficient. On a ground robot, we expect that wheel odometry would be adequate.

2.4 Implementation

2.4.1 Pushbroom Algorithm

Like other block-matching algorithms, we use sum of absolute differences (SAD) to detect pixel block similarity. In addition to detecting matching regions, we score blocks based on

their abundance of edges. This allows us to disambiguate the situation where two pixel blocks might both be completely black, giving a good similarity score, but still not providing a valid stereo match. To generate an edge map, we use a Laplacian with an aperture size (`ksize`) of 3. We then take the summation of the 5x5 block in the edge map and reject any blocks below a threshold for edge-abundance.

After rejecting blocks for lack of edges, we score the remaining blocks based on SAD match divided by the summation of edge-values in the pixel block:

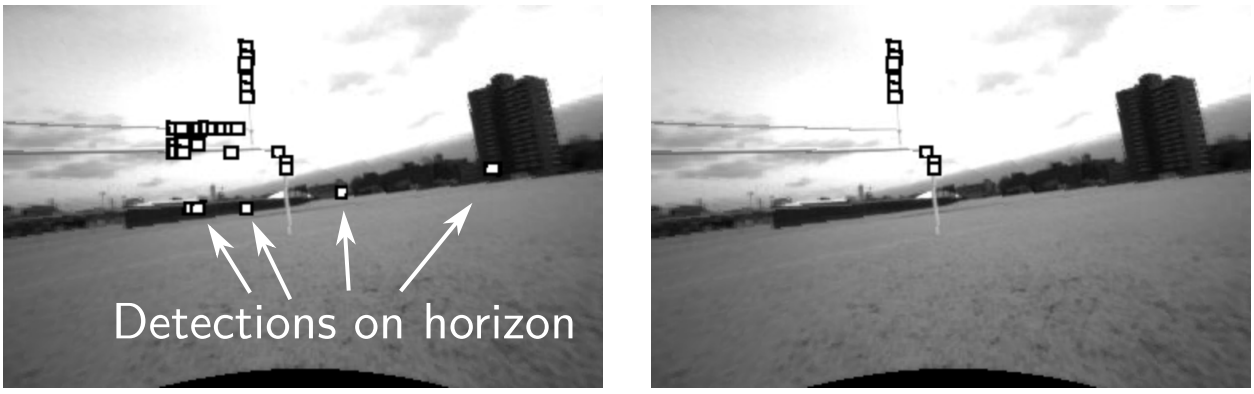
$$S = \frac{\overbrace{\sum_{i=0}^{\text{5x5}} |p(i)_{left} - p(i)_{right}|}^{\text{Sum of absolute differences (SAD)}}}{\sum_{i=0}^{\text{5x5}} L\left(p(i)_{left}\right) + L\left(p(i)_{right}\right)}$$

where $p(i)$ denotes a pixel value in the 5x5 block and L is the Laplacian. We then threshold on the score, S, to determine if there is a match.

We have deliberately chosen a design and parameters to cause sparse detections with few false positives. For obstacle avoidance, we do not need to see every point on an obstacle but a false positive might cause the aircraft to take unnecessary risks to avoid a phantom obstacle.

All two-camera stereo systems suffer from some ambiguities. With horizontal cameras, we cannot disambiguate scenes with horizontal self-similarity, such as buildings with grid-like windows or an uninterrupted horizon. These horizontal repeating patterns can fool stereo into thinking that it has found an obstacle when it has not.

While we cannot correct these blocks without more sophisticated processing or additional cameras, we can detect and eliminate them. To do this, we perform additional block-matching searches in the right image near our candidate obstacle. If we find that one block in the left image matches blocks in the right image at different disparities, we conclude that the pixel block exhibits local self-similarity and reject it. While this search may seem expensive, in practice the block-matching above reduces the search size so dramatically that we can run



(a) Without horizontal invariance filter.

(b) With horizontal invariance filter.

Figure 2-3: All stereo systems suffer from repeating textures which cannot be disambiguated with only two cameras. Here, we demonstrate our filter for removing self-similarity. Detected pixel blocks are marked with squares. Note that the filter removes all self-similar regions including those on obstacles, limiting our ability to detect untextured, horizontal obstacles.

this filter online. Figure 2-3 demonstrates this filter running on flight data.

2.4.2 Hardware Platform

We implemented the pushbroom stereo algorithm on a quad-core 1.7Ghz ARM, commercially available in the ODROID-U3 package, weighing under 50 grams[2]. Our cameras' resolution and stereo baseline can support reliable detections out to approximately 5-10 meters, so we use 4.8 meters as our single-disparity distance in this experiment. We detect over 5x5 pixel blocks, iterating through the left image with 8 parallel threads.

We use two Point Grey Firefly MV[3] cameras, configured for 8-bit grayscale with 2x2 pixel binning, running at 376x240 at 120 frames per second. A second ODROID-U3, communicating over LCM [44], runs our state-estimatator (a Kalman filter from [16]) and connects to our low-level interface, a firmware-modified APM 2.5[4], which provides access to our servo motors, barometric altimeter, pitot tube airspeed sensor, and 3-axis accelerometer, gyroscope, and magnetometer suite. Further information about our flight platform (Figure 2-4)

[2]Hardkernel co., Ltd. `http://hardkernel.com`
[3]Point Grey Research, Inc. `http://www.ptgrey.com`
[4]3D Robotics, Inc. `http://3drobotics.com/`

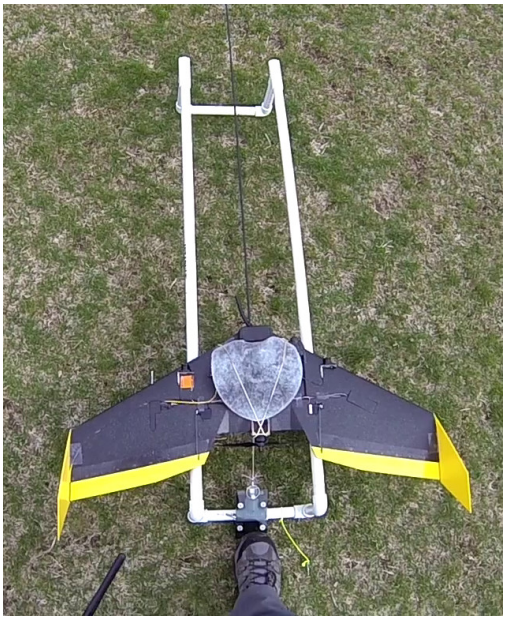


Figure 2-4: Aircraft hardware in the field. We use a small catapult for consistent launches near obstacles.

is in Chapter 4.

2.5 Results

2.5.1 Single-Disparity Stereo

To determine the cost of thresholding stereo points instead of using the best-matching block from a search through depth, we walked with our aircraft near obstacles and recorded the output of the onboard stereo system with the state-estimator disabled[5]. We then, offline, used OpenCV's [15] block-matching stereo implementation (`StereoBM`) to compute a full depth map at each frame. We then removed any 3D point that did not correspond to a match within 0.5 meters of our single-disparity depth to produce a comparison metric for the two systems.

With these data, we detected false-positives by computing the Euclidean distance from

[5]Our state-estimator relies on the pitot-tube airspeed sensor for speed estimation, which does not perform well below flight speeds.

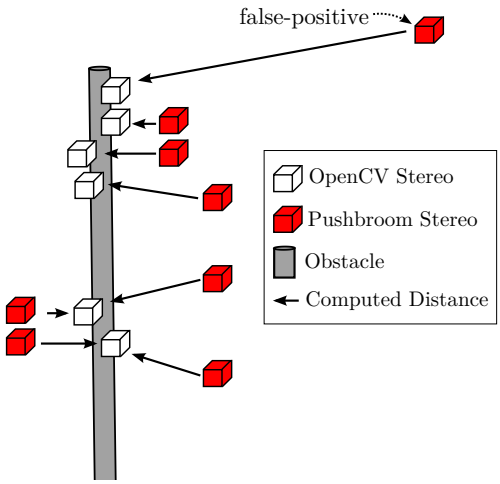


Figure 2-5: Sketch of our evaluation strategy for single-disparity stereo. We detect false-positives by computing the distance from single-disparity stereo's output (red) to the nearest point from OpenCV's `StereoBM` (white). False positives stand out with large distances (labeled box).

each single-disparity stereo coordinate to the nearest point produced by the depth-cropped StereoBM (Figure 2-5). Single-disparity stereo points that are far away from any StereoBM points may be false-positives introduced by our more limited computation technique. StereoBM produces a large number of false negatives, so we do not perform a false-negative comparison on this dataset (see Section 2.5.2 below.)

Our ground dataset includes over 23,000 frames in four different locations with varying lighting conditions, types of obstacles, and obstacle density. Over the entire dataset, we find that single-disparity stereo produces points within 0.5 meters of StereoBM 60.9% and within 1.0 meters 71.2% of the time (Figure 2-6). For context, the aircraft's wingspan is 0.86 meters and it covers 0.5 meters in 0.03 to 0.07 seconds.

2.5.2 Flight Experiments

To test the full system with an integrated state-estimator, we flew our platform close to obstacles (Figure 2-7) on three different flights, recorded control inputs, sensor data, camera images, and on-board stereo processing results. Figures 2-8 and 2-9 show on-board stereo

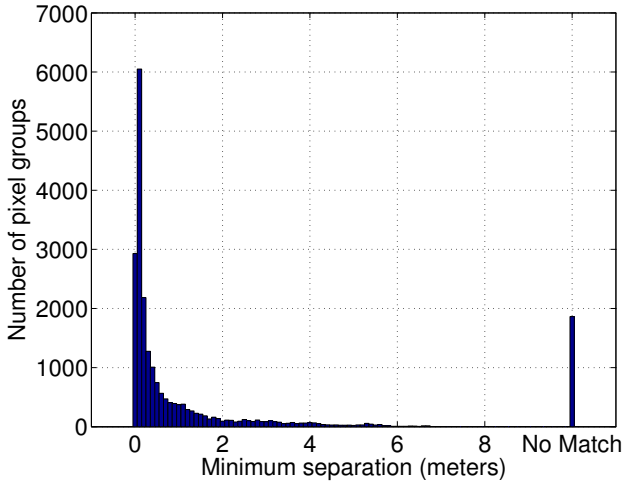


Figure 2-6: Results of the false-positive benchmark described in Figure 2-5 on 23,000+ frames. *No Match* indicates single-disparity points where there was no matching StereoBM point on the frame. We find that 8.2% of detected pixels fall into this category.

detections as the aircraft approaches an obstacle.

During each flight, we detected points on every obstacle in real time. Our GPS-denied state estimate was robust enough to provide online estimation of how the location of the obstacles evolved relative to the aircraft. While these flights were manually piloted, we present fully autonomous flights in Chapter 4.

To benchmark our system, we again used OpenCV's block-matching stereo as a coarse, offline, approximation of ground truth. At each frame, we ran full block-matching stereo, recorded all 3D points detected, and then hand-labeled regions in which there were obstacles to increase StereoBM's accuracy.

We compared those data to pushbroom stereo's 3D data in two ways. First, we performed the same false-positive detection as in Section 2.5.1, except we included *all 3D points seen and remembered* as we flew forward. Second, we searched for false-negatives, or obstacles pushbroom stereo missed, by computing the distance from each StereoBM coordinate to the nearest 3D coordinate seen and remembered by pushbroom stereo (Figure 2-11a).

Figures 2-10 and 2-11b show the results of the false-positive and false-negative benchmarks on all three flights respectively. Our system does not produce many false-positives,

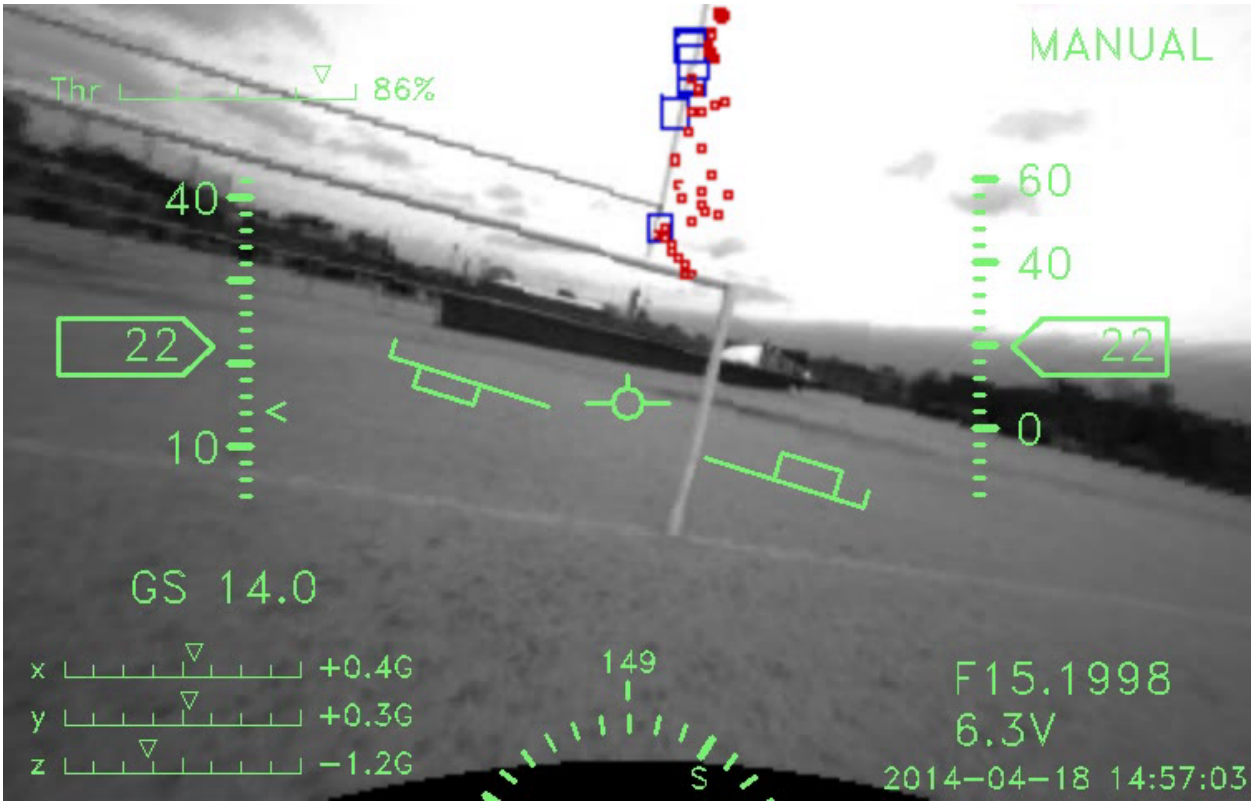


Figure 2-7: In-flight snapshot of single-disparity stereo detections on a goalpost (blue boxes) and past detections integrated through the GPS-denied state estimate and reprojected back on the image (red dots). Overlay includes relevant flight data such as airspeed in MPH (left) and altitude in feet (right).

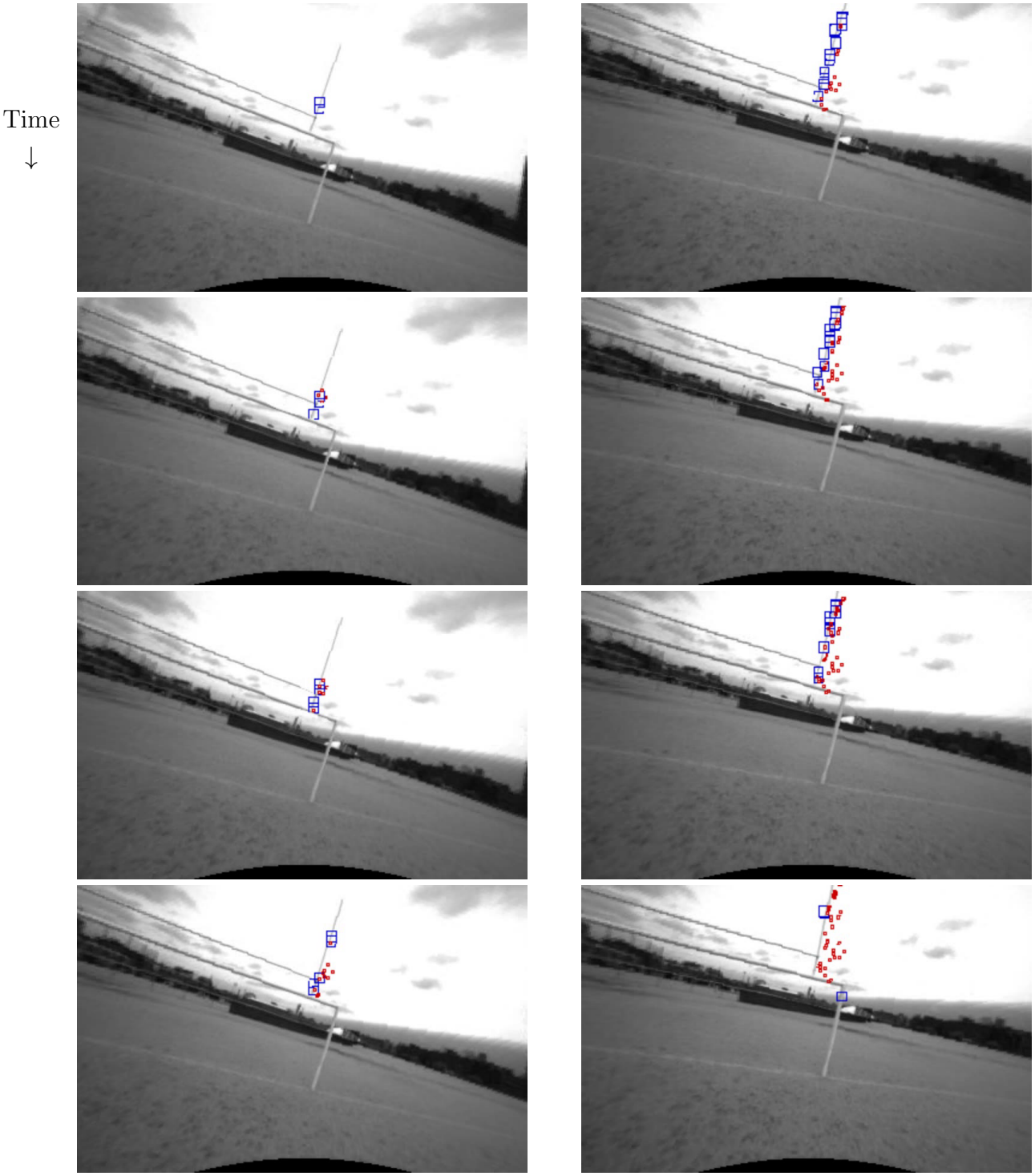


Figure 2-8: Sequence of stills from an obstacle detection. Each image is 0.02 seconds (20ms) after the previous. The entire set captures 0.16 seconds. Here, the fieldgoal is detected in the first frames (blue boxes). Afterwards, the position of those detections is estimated via the state estimator and reprojected back onto the image (red dots).

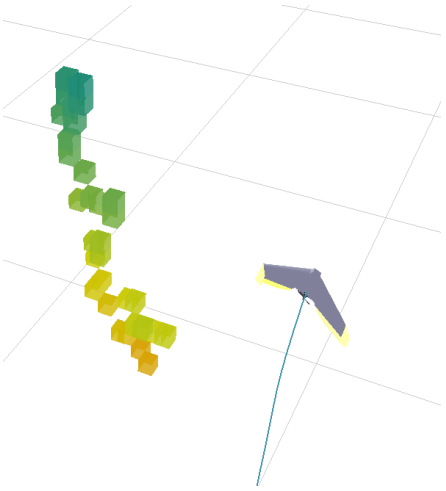


Figure 2-9: Obstacle detection from Figure 2-8 rendered in a 3D visualizer. While we do not endeavor to build maps, our system outputs pointclouds providing compatibility with many existing planning, visualization, and control tools.

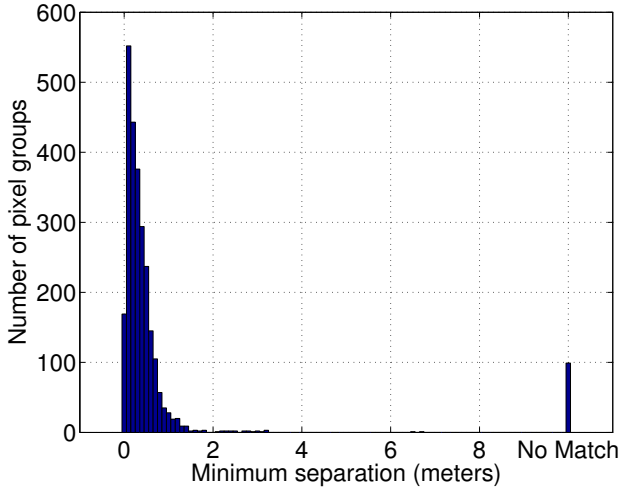
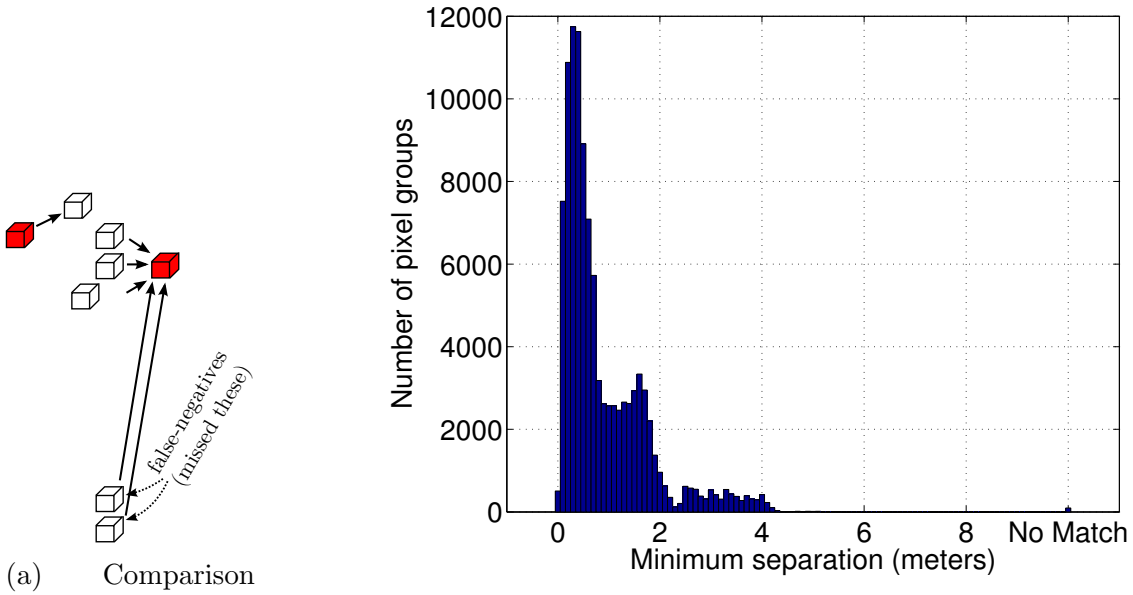


Figure 2-10: Results of the comparison as described in Figure 2-5 (a). Our system produces few outliers (74.8% and 92.6% within 0.5 and 1.0 meters respectively), even as we integrate our state estimate, and the obstacle positions, forward. *No Match* indicates points that pushbroom stereo detected but there were no block-matching stereo detections on that frame.



(a) Comparison technique.

(b) False-negative comparison.

Figure 2-11: Results of the false-negative benchmark on flight data. In this comparison, we compute distance from each StereoBM point (white) to the nearest pushbroom stereo coordinate (red). False-negatives stand out with large distances. Pushbroom stereo performs well, detecting an obstacle within 2.0 meters of StereoBM 91.3% of the time.

with 74.8% points falling within 0.5 meters and 92.6% falling less than one meter from OpenCV's StereoBM implementation. For comparison, a system producing random points at approximately the same frequency gives approximately 1.2% and 3.2% for 0.5 and 1.0 meters respectively.

As Figure 2-11 shows, pushbroom stereo detects most of the points on obstacles that StereoBM sees, missing by 1.0 meter or more 32.4% of the time. A random system misses approximately 86% of the time by the same metric. For context, the closest our skilled pilot ever flew to an obstacle in this experiment was about two meters.

These metrics demonstrate that the pushbroom stereo system scarifies a limited amount of performance for a substantial reduction in computational cost, and thus a gain in speed. Finally, we note that all data in this thesis use identical threshold, scoring, and other parameters, despite changing the detection distance, lens focal length, and camera calibration between Chapters 2 and 4.

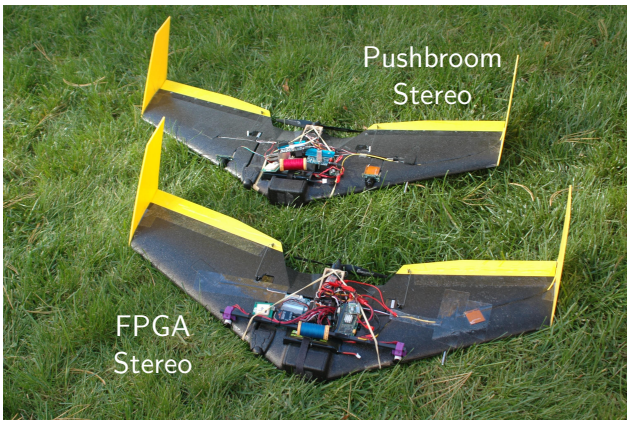


Figure 2-12: Experimental aircraft platforms holding the pushbroom stereo system (back) and the FGPA stereo system (front). Cameras are mounted on the front of the wings at the same baseline (14 inches) on both airframes. Covers over the electronics were removed for this photograph.

2.6 Comparison to FPGA Stereo

We compare the above system to a Field Programable Gate Array (FPGA) stereo implementation which computes a dense depth map at every frame in hardware. We run both systems at 120 frames per second at 376x240 pixel resolution, on duplicate flight platforms (Figure 2-12) flying at over 13 m/s (29 MPH).

The FPGA system, created and built by Dominik Honegger and Helen Oleynikova, uses a modified version of semi-global matching (SGM) [40] for stereo processing and is detailed in [41]. We flew the FPGA system in the same flight pattern as detailed above and recorded results for comparison.

Figure 2-13 demonstrates the primary difference between the two systems. The FPGA system (blue dots) produces many matches, increasing in number as the distance to the obstacle decreases. The pushbroom system detects nothing until the threshold distance (4.8 meters), around which it finds matches (green stars). Past that distance, there are no additional detections, but past detections remain in memory (red crosses).

The FPGA system produces dense stereo data, giving depth on almost the entire obstacle (Figure 2-14a). The pushbroom stereo system is tuned to reject almost all outliers, so all detections can be treated as obstacles without further processing. The dense data delivers

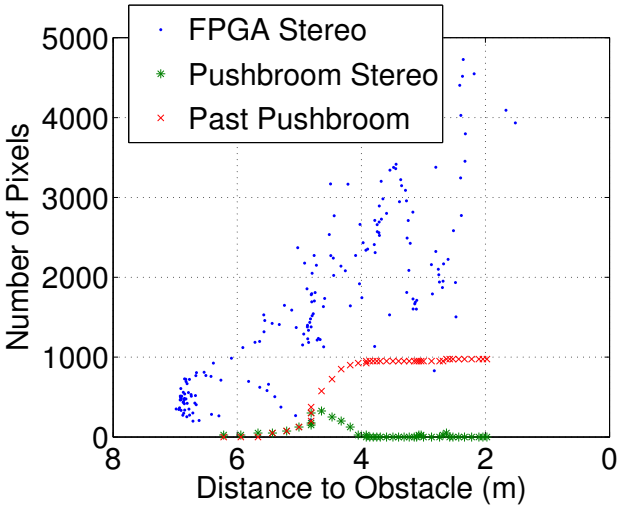
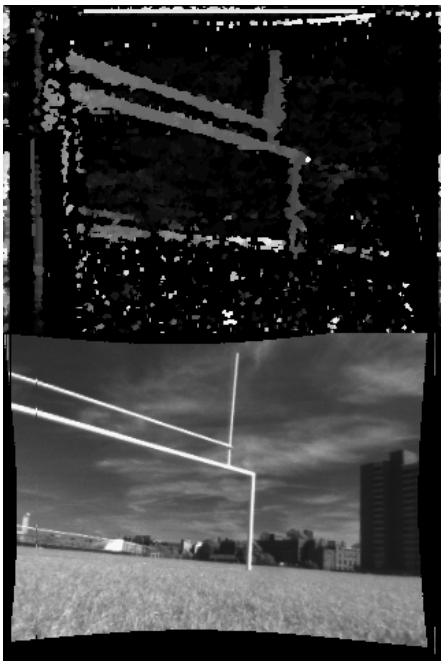
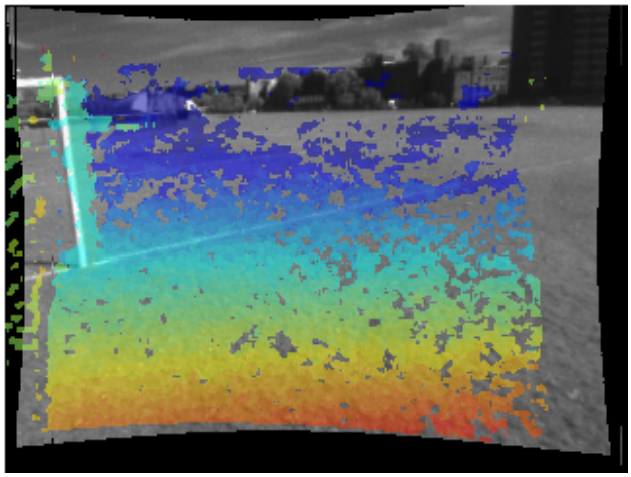


Figure 2-13: Number of pixels detected while flying towards the fieldgoal obstacle. The FPGA system produces an increasing number of detections as the obstacle nears (blue dots), while the pushbroom system only detects the obstacle around the set distance of 4.8 meters (green stars). Past that, pushbroom detections (red crosses) remain in memory for avoidance. Note that the x-axis is reversed allowing time to flow left-to-right.

more information about nearby obstacles, but also requires more intelligent filtering for autonomous operation (Figures 2-14b and 2-15). Finally, pushbroom stereo is substantially easier to replicate since all of its components are off-the-shelf and camera/CPU independent. For example, [85] uses a substantially less powerful processor, different cameras, and controls a boat, but is able to perform online reactive obstacle avoidance using pushbroom stereo.



(a) FPGA produced grayscale depthmap (top) compared with raw image (bottom).



(b) Depth output from the FPGA system overlaid on the right camera image. Depth ranges from red (close) to blue (far).

Figure 2-14: Results from the FPGA stereo system. Note that the FPGA system produces substantially more dense depth estimates than the pushbroom system.

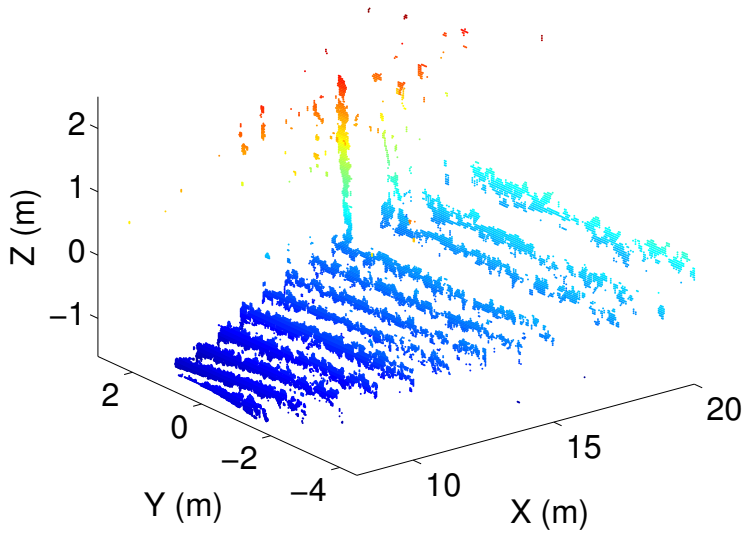


Figure 2-15: 3D pointcloud generated from the depth map in Figure 2-14b. False colored by height ranging from blue (low) to red (high).

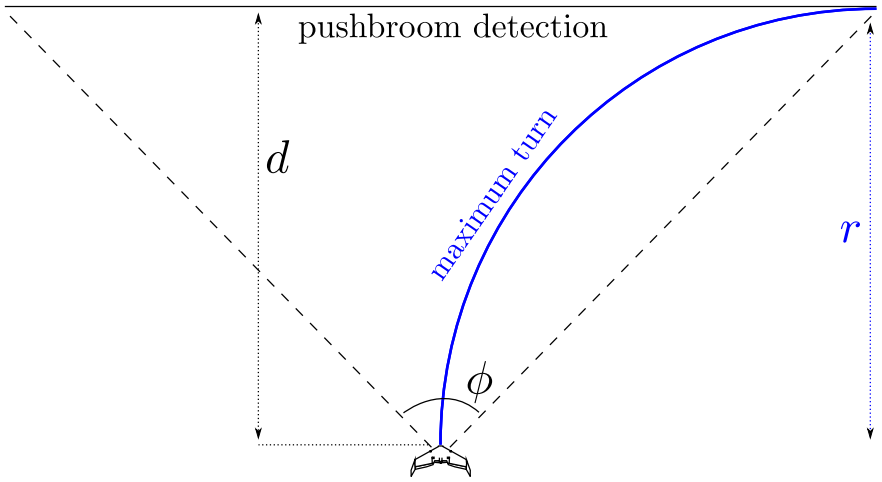


Figure 2-16: Sketch of the 2D obstacle-avoidance case showing the pushbroom detection distance (d), the field of view (ϕ), and the turning radius (r). In this drawing, $d = r$ and $\phi = 90°$.

2.7 Analysis of Obstacle Avoidance Limits

2.7.1 2D without Occlusions

In this section we present an analysis of what types of obstacle fields an aircraft equipped with pushbroom stereo could theoretically fly through. First, we make the following simplifying assumptions:

- 2D, static environment
- Aircraft moves along arcs with a minimum radius r
- Obstacles do not occlude other obstacles
- Pushbroom stereo detects objects in a line at distance d.

The analysis depends on three parameters (Figure 2-16):

1. minimum turning radius, r
2. pushbroom stereo distance, d
3. field of view, ϕ

The relevant ratio in this case is between the sensing range, d, and the minimum turning radius, r. There are three cases to consider: $d = r$, $d > r$, and $d < r$.

Case I ($d = r$): If $d = r$ and $\phi = 90^\circ$, the aircraft can always see obstacles at the edge of its minimum turning radius, so it will never crash because it failed to see an obstacle. Given that the aircraft starts at least d distance from obstacles, it will be able to avoid any obstacle field that has objects spaced at least $2r$ apart. This is easy to see because the system can choose a minimum radius turn and make a 180° turn along the circle with diameter $2r$.

If $\phi > 90^\circ$, the system is capable of seeing areas it cannot reach, so no limits are improved or reduced. Clearly, the aircraft could reach those regions with multiple future maneuvers, but we discount that case since pushbroom stereo removes past obstacles from memory relatively quickly. If, however, $\phi < 90^\circ$, the system will not be able to see the full path for it's minimum radius turn. The result is that it can only avoid limited-width obstacles, as detailed in Case III below.

Case II ($d > r$): The second case occurs when $d > r$, or when the pushbroom detection region is farther than the minimum radius turn (Figure 2-17, left). In this case, the user should change d or r in software so that $d = r$. Otherwise, there is a possibility of missing obstacles that the aircraft is capable of turning into.

Case III ($d < r$): Finally, $d < r$ occurs when the detection horizon is short compared to the minimum turning radius. In this case, the system will always see everything in its path, but might be unable to avoid collision with obstacles. For example, if the plane is heading at a wall, the system will not know until even a minimum radius turn cannot save it from collision.

In the $d < r$ case, we can determine the maximum width of obstacles that the system is capable of avoiding. Figure 2-18 shows the relation, which is the maximum turn the aircraft can take as soon as it detects the obstacle. The width of the obstacle (ignoring the aircraft's own geometry) is $W = 2\left(r - \sqrt{r^2 - d^2}\right)$, derived from the change in x coordinate as the aircraft moves forward along a circle. Clearly, as d increases or r decreases, the maximum width improves.

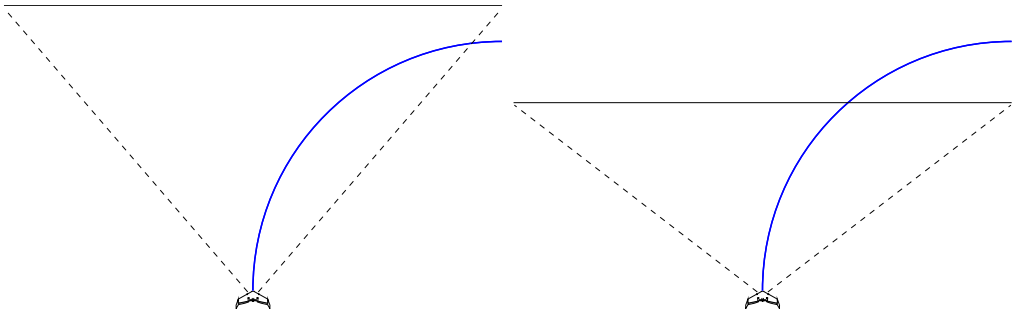


Figure 2-17: Sketches of the two other cases: $d > r$ (left) and $d < r$ (right).

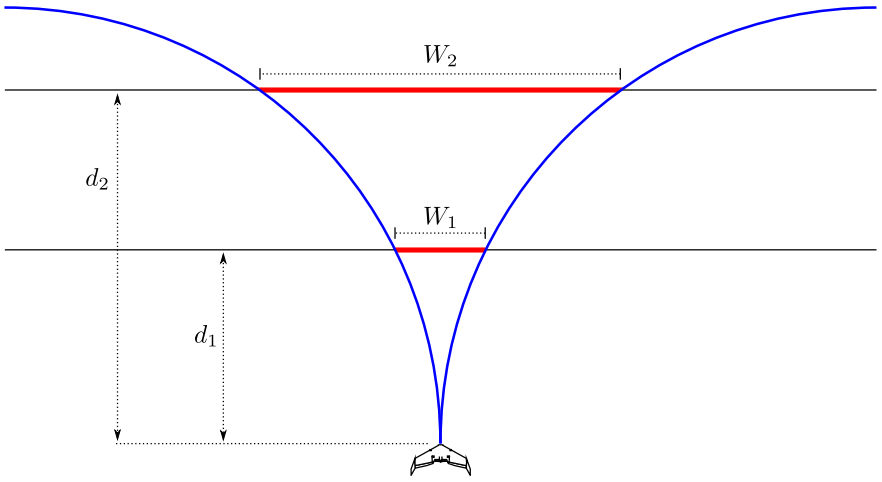


Figure 2-18: The maximum width of an obstacle (excluding the aircraft's geometry) when $d < r$ is $W = 2\left(r - \sqrt{r^2 - d^2}\right)$.

In this case, the minimum ϕ required for W is easily computed from the system's geometry: $\phi = 2\tan^{-1}\left(\frac{W}{2d}\right)$. If ϕ is less than that, it will further limit W with the following relation: $W = 2d\tan\frac{\phi}{2}$. We can combine these results to find an expression for W:

$$W = \min\left[2\left(r - \sqrt{r^2 - d^2}\right), 2d\tan\frac{\phi}{2}\right]$$

The system described in Chapter 4 has $d = 10m$ and $\phi = 92°$. For the real aircraft using the trajectory library also detailed below, $r_{nominal} = 70.8m$ when turning *without losing altitude* and $r_{aggressive} = 7.7m$ when losing altitude. We note that the aircraft is capable of

even tighter turns, but those were not included in our trajectory library. In practice, when flying close to obstacles, maintaining altitude is a primary concern, so the online planning system will limit its execution of altitude-losing turns. If altitude was not an issue, reducing d would be advantageous, but with that concern, $d < r$, so we want to maximize d, up to $70.8m$.

2.7.2 2D with Occlusions

Next we consider obstacles that occlude each other. Assume that we have two obstacles L meters apart and in line with the aircraft's flight path. The first obstacle will occlude the second at the pushbroom detection distance with a "shadow" that has width $S = \frac{O \cdot d}{d - L}$, where O is the width of the occluding obstacle. For the shadow to occlude the second obstacle, the system must have decided not to turn away from the first obstacle at its earliest detection, since pushbroom stereo will always detect a closer obstacle before a further one along the direction of flight. We note that this differs from the analysis above, where the aircraft immediately turns away from obstacles to ensure safety.

In the worst case, we assume that there is some set of non-occluded obstacles, such as a gap between trees, that requires the aircraft to fly the path of minimum deviation around the first obstacle (Figure 2-19). It then must maneuver to avoid the second obstacle and the shadowed region. If the second obstacle was completely in the shadowed region (had width less than S), the system would never detect it. Even so, however, the system could reason about the shadowed region and avoid the danger. If $S > W$ (neglecting the small deviation from O), however, the shadowed region would be too large for the aircraft to avoid, potentially causing a collision. Thus, to ensure safety the following must be true for all obstacles:

$$\frac{O \cdot d}{d - L} < \min \left[2 \left(r - \sqrt{r^2 - d^2} \right), 2d \tan \frac{\phi}{2} \right].$$

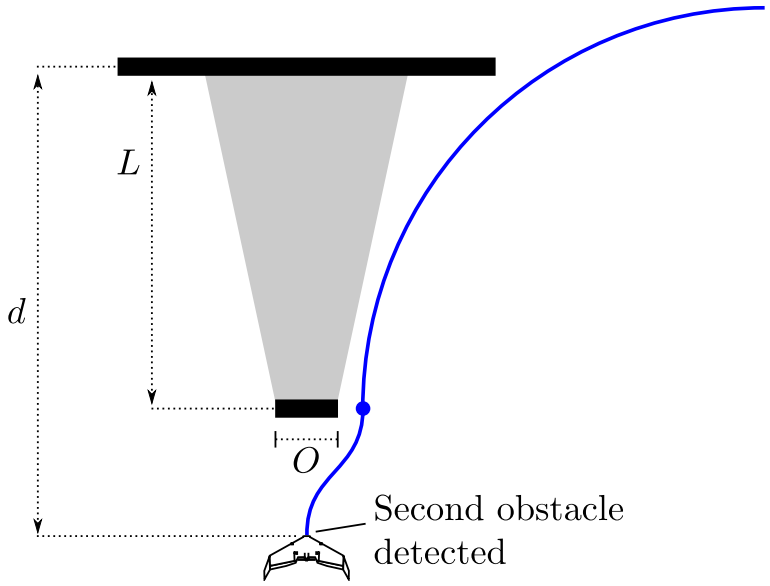


Figure 2-19: Sketch of the obstacles with occlusions case. The gray region shows the occluded area.

Chapter 3

Aircraft Control

We use a model-based control system consisting of trajectories selected from a library in real-time based on a pointcloud produced by the pushbroom stereo vision system. Model-based control allows us to use sophisticated planning techniques and, soon, perform verification of our system. Each open-loop trajectory has a precomputed time-varying linear quadratic regulator (TVLQR) associated with it for closed-loop control. The control is performed with the full state estimate of the vehicle produced by the onboard state estimator running in the loop. This chapter details the aircraft model, trajectory library generation, control, and integration with the vision system.

3.1 Aircraft Model

3.1.1 Model Structure

We use a 12-state model of the aircraft with 3 control inputs. The state variables are:

$$\mathbf{x} = \begin{bmatrix} x & y & z & \phi & \theta & \psi & U & V & W & P & Q & R \end{bmatrix}^T$$

which are defined in Table 3.1. The coodinate system is defined in Figure 3-1.

We use a flat-plate model inspired by Cory's work on his fixed-wing glider [22]. We model

$\mathbf{x}$	state vector
$\mathbf{u}$	control vector
x	position on the x-axis
y	position on the y-axis
z	position on the z-axis
ϕ	roll
θ	pitch
ψ	yaw
U	forward velocity in the body frame (x-axis)
V	velocity to the right in body frame (y-axis)
W	downwards velocity in body frame (z-axis)
P	angular rotation about the x-axis
Q	angular rotation about the y-axis
R	angular rotation about the z-axis
$\dot{x}$	time derivative of x
α_w	angle of attack of the wing
α_e	angle of attack of an elevon
α_l	angle of attack of a winglet
v_{e_x}	x-component of the velocity of an elevon
ρ	density of air

Table 3.1: Notation

the aircraft with three fixed flat plates representing the wing and winglets and two smaller moving flat plates for the elevons (Figure 3-2). We model the propeller as a thrust-generating element, but ignore its aerodynamic drag, blade flapping, and other higher order effects.

Following Cory (but modifying for 3D), we model the coefficient of lift ($\boldsymbol{F}_L$) and drag ($\boldsymbol{F}_D$) as:

$$C_L = 2\sin(\alpha)\cos(\alpha) \tag{3.1}$$

$$C_D = 2\sin^2(\alpha) \tag{3.2}$$

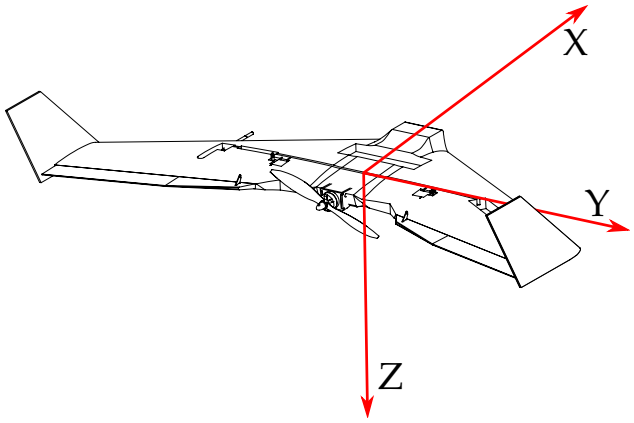


Figure 3-1: Model coordinate system.

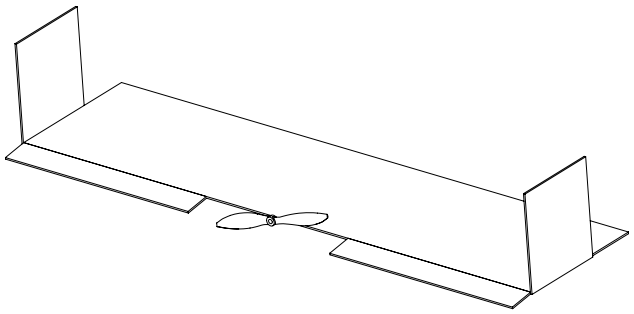


Figure 3-2: Sketch of the flat plate model with a single flat plate for the wing, two flat plate elevons, two flat plate winglets, and a thurst model.

where α is the wing's angle of attack. Given standard lift and drag forces:

$$\boldsymbol{F}_L = \frac{1}{2}\rho|\boldsymbol{v}|^2 C_L S \tag{3.3}$$

$$\boldsymbol{F}_D = \frac{1}{2}\rho|\boldsymbol{v}|^2 C_D S \tag{3.4}$$

where $\boldsymbol{v} = \begin{bmatrix} U & V & W \end{bmatrix}^T$, ρ is the air density, $\boldsymbol{v}$ is the wing's velocity in the x- and z-axes, and S is the flat plate's area.

Combining equations 3.1–3.4 we find:

$$\boldsymbol{F}_L = \rho|\boldsymbol{v}|^2 \cos(\alpha)\sin(\alpha) S \tag{3.5}$$

$$\boldsymbol{F}_D = \rho|\boldsymbol{v}|^2 \sin^2(\alpha) S \tag{3.6}$$

Note that the lift force is perpendicular to the aircraft's velocity in the positive z direction and the drag force is anti-parallel to the velocity [21]. To account for this, we rotate the lift and drag forces based on the angle of attack to find the wing force, $\boldsymbol{F}_w$.

$$\boldsymbol{F}_w = \begin{bmatrix} -\cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 0 & 0 \\ -\sin(\alpha) & 0 & -\cos(\alpha) \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_D \\ 0 \\ \boldsymbol{F}_L \end{bmatrix} \tag{3.7}$$

Computing Angle-of-Attack

We start with computing the angle of attack for the main wing:

$$\alpha_w = \tan^{-1}\left(\frac{W}{U}\right) \tag{3.8}$$

To compute α for the elevons, we must account for any additional velocity from the aircraft's pitching motion (because the airframe does not pitch about the center of the elevons):

$$\boldsymbol{x}_e = \begin{bmatrix} e_{a_x} + \dfrac{e_c}{2} - \dfrac{e_c}{2}\cos(u_e) \\ e_{a_y} \\ -\dfrac{e_c}{2}\sin(u_e) \end{bmatrix} \tag{3.9}$$

where $\boldsymbol{x}_e$ is the elevon's position, e_{a_x} is the distance from the aircraft's center to the elevon's center along the x-axis, e_c is the elevon's chord, u_e is the control input for the elevon, and e_{a_y} is the distance from the aircraft's center to the elevon's center along the y-axis. For velocity:

$$\boldsymbol{v}_e = \boldsymbol{v} + \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \times \boldsymbol{x}_e \tag{3.10}$$

With the velocity of the elevon, we can compute its angle of attack:

$$\alpha_e = \tan^{-1}(v_{e_z}/v_{e_x}) \tag{3.11}$$

where v_{e_x} and v_{e_z} index the x and z components of the elevon's velocity ($\boldsymbol{v}_e$) respectively. Finally, we compute the forces from the elevons:

$$\boldsymbol{F}_e = \begin{bmatrix} -\cos(\alpha_e) & 0 & \sin(\alpha_e) \\ 0 & 0 & 0 \\ -\sin(\alpha_e) & 0 & -\cos(\alpha_e) \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_D \\ 0 \\ \boldsymbol{F}_L \end{bmatrix} \tag{3.12}$$

being careful to substitute $\boldsymbol{v}_e$ for $\boldsymbol{v}$ and α_e for α in equations 3.5 and 3.6.

Unlike the aircraft's main wing, which is centered at the aircraft's center of mass, the elevons produce a moment:

$$\boldsymbol{M}_e = \boldsymbol{x}_e \times \boldsymbol{F}_e \tag{3.13}$$

The angle of attack on the winglets is similar, but rotated 90°. We again start by computing the flat plate's velocity:

$$\boldsymbol{v}_l = \boldsymbol{v} + \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \times \boldsymbol{x}_l \tag{3.14}$$

In this case, $\boldsymbol{x}_l$, the position of the winglet, is a constant, so it is easy to measure on the aircraft, giving:

$$\alpha_l = \tan^{-1}(v_{l_y}/v_{l_x}) \tag{3.15}$$

Note that here we use v_{l_y} in the numerator because the winglet is rotated compared to the main wing or the elevons. We compute the winglet force, $\boldsymbol{F}_l$, by substituting the apporpriate terms in equation 3.7.

Computation of the winglet's moment follows simply:

$$\boldsymbol{M}_l = \boldsymbol{x}_l \times \boldsymbol{F}_l \tag{3.16}$$

Other Terms

The model accounts for body drag in the forward direction:

$$\boldsymbol{F}_{body\_drag} = \begin{bmatrix} -\frac{1}{2}\text{sign}(U) b_{d_x} \rho U^2 \\ 0 \\ 0 \end{bmatrix} \tag{3.17}$$

The gravity vector needs to be rotated into the body frame:

$$\boldsymbol{F}_{gravity} = \begin{bmatrix} c_y c_p & c_y s_p s_r - s_y c_r & c_y s_p c_r + s_y s_r \\ s_y c_p & s_y s_p s_r + c_y c_r & s_y s_p c_r - c_y s_r \\ -s_p & c_p s_r & c_p c_r \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{3.18}$$

where s and c are shorthand for sin and cos, and the subscripts, r, p, and y are shorthand for roll, pitch, and yaw respectively.

Input Models

We use a linear model for each servo that maps servo command to elevon deflection:

$$\begin{aligned} e_l &= m_l u_l + l_0 \\ e_r &= m_r u_r + r_0 \end{aligned} \tag{3.19}$$

where e_l is the deflection of the left control surface in radians, m_l and l_0 are measured parameters that are different for each airframe, and u_l is the length of the servo control signal pulse in micro-seconds (ranging from 1000 to 2000).

We model the motor and propeller's thrust as a linear function of control input as well:

$$\boldsymbol{F}_{thrust} = \begin{bmatrix} k_m u_m + k_0 \\ 0 \\ 0 \end{bmatrix} \tag{3.20}$$

where k_m, k_0, are coefficients fit experimentally and u_m is the bounded control input. In this model, we choose to ignore torque produced by the propeller.

Combining the above forces and torques, we can write the dynamics:

$$m\ddot{\boldsymbol{x}} = \boldsymbol{F}_w + \boldsymbol{F}_{e_L} + \boldsymbol{F}_{e_R} + \boldsymbol{F}_{l_L} + \boldsymbol{F}_{l_R} + \boldsymbol{F}_{thrust} + \boldsymbol{F}_{body\_drag} + \boldsymbol{F}_{gravity} \tag{3.21}$$

and the total moments:

$$I\dot{\boldsymbol{\omega}} = \boldsymbol{M}_{e_L} + \boldsymbol{M}_{e_R} + \boldsymbol{M}_{l_L} + \boldsymbol{M}_{l_R} + \boldsymbol{M}_{rate\_dependent} \tag{3.22}$$

where $\boldsymbol{M}_{rate\_dependent}$ is a parameter (see Table 3.2).

Parameter	Value (*units*)	Identification Method
Mass (m)	0.648412 kg	Measured
Wing span (w_s)	0.8636 m^2	Measured
Wing chord (w_c)	0.2097 m	Measured
Elevon span (e_s)	0.31115 m^2	Measured
Elevon chord (e_c)	0.0402 m	Measured
Elevon moment arm (X) (e_{a_x})	0.12495 m	Measured
Elevon moment arm (Y) (e_{a_y})	0.276225 m	Measured
Winglet area (t_{area})	0.01944076 m^2	Measured
Linear component, left servo (m_l)	0.002195 ($rad/\mu s$)	Measured
Affine component, left servo (l_0)	-3.017 (rad)	Measured
Linear component, right servo (m_r)	-0.0019 ($rad/\mu s$)	Measured
Affine component, right servo (r_0)	2.811 (rad)	Measured
Linear component, thrust (k_m)	0.01004 $\mu s/N$	Experiment
Affine component, thrust (k_0)	12.17 N	Experiment
Moment of inertia (J_x)	0.0153 $kg \cdot m^2$	CAD and Experiment
Moment of inertia (J_y)	0.0052 $kg \cdot m^2$	CAD and Experiment
Moment of inertia (J_z)	0.0184 $kg \cdot m^2$	CAD and Experiment
Elevon lift factor	1.30	Fit
Elevon drag factor	0.101	Fit
x-axis body drag factor (b_{d_x})	0.0443	Fit
x-axis rate-dependent moment (M_P)	-0.0740	Fit
y-axis rate-dependent moment (M_Q)	-0.1150	Fit
z-axis rate-dependent moment (M_R)	0	Fit

Table 3.2: Model parameters, their values, and the identification method used to find them.

3.1.2 System Identification

The model described above has 23 parameters we need to identify. Some are easily measured, like the aircraft's mass, and others are more difficult, such as the body's drag coefficient. Table 3.2 shows every model parameter, its identified value, and the identification method.

Elevon Deflection

We measure the deflection of the elevons by hand (Figure 3-3) and fit a simple linear model (Equation 3.19 and Figure 3-4). We also note the minimum and maximum command and deflection for each elevon and ensure the controller does not exceed these bounds. In practice, the parameters are relatively constant between airframes.

Figure 3-3: Measuring servo deflection for a given input.

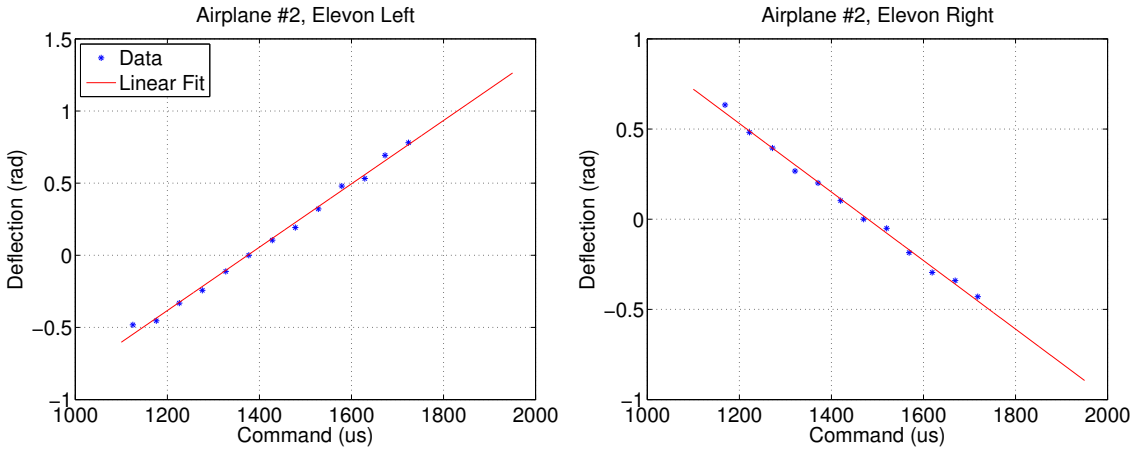


Figure 3-4: Linear model fitting elevon command (PWM pulse length in microseconds) to elevon deflection in radians.

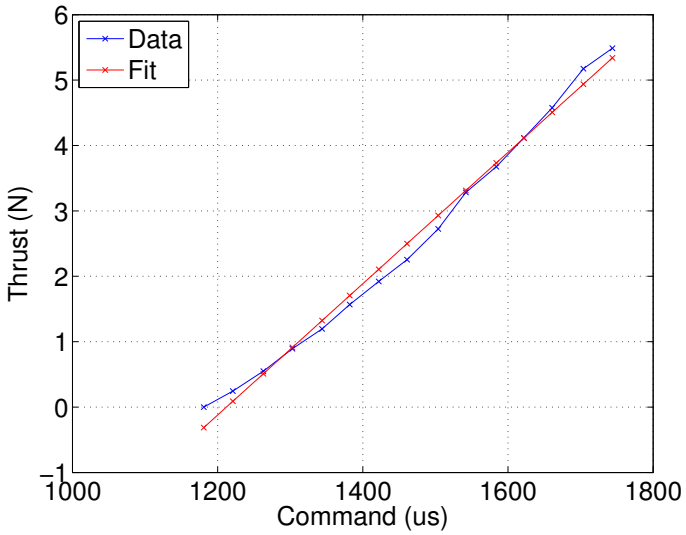


Figure 3-5: Thrust data (blue) with a linear fit (red). The x-axis shows the commanded pulse length in mirco-seconds to the speed controller. Clearly the linear model is not valid below 1250 μs because the system can not generate neagtive thrust.

Thrust Force

We fit a linear thrust model using a simple apparatus to measure thrust force when stationary. The setup placed the aircraft pointing straight down, suspended by a scale. The weight of the was recorded with the motor off. We then increased throttle, recording the additional weight of the plane, along with the throttle setting, until reaching 100%. A more accurate model would be obtained by performing the same experiment in a wind tunnel at flight speed but we deemed that unnecessary. Figure 3-5 shows the measured values and accompanying linear fit for the model.

Moments of Inertia

We determined the moments of inertia using the average of two techniques. First, we built an accurate CAD model (Figure 3-7) of the vehicle and numerically computed moments of inertia for that model. Second, we used a bifilar pendulum setup to measure the moment of inertia about each axis. Figure 3-6 demonstrates this setup, which produced results within 13% of the CAD estimate on all three axes and on stock material of a known, simple, shape.

Figure 3-6: Measuring the moment of inertia with a bifilar pendulum (roll, pitch, and yaw axes respectively). The technique involves measuring the period of oscillations about the axis in quesiton.

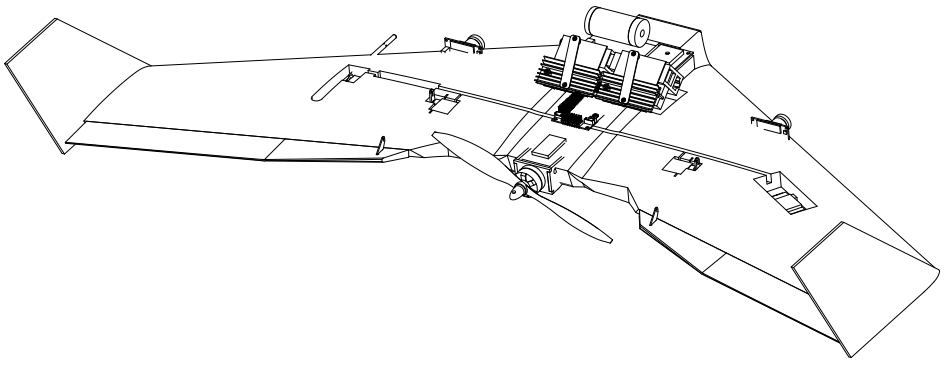


Figure 3-7: CAD model used to estimate moments of inertia.

Parameter Fitting

We use a gray-box fitting technique to identify the values of parameters that are difficult to measure directly. With the aircraft model above, we optimized the parameters designated as "Fit" in Table 3.2 using MATLAB's System Identification Toolbox [64].

Since we do not have ground-truth data from a motion capture system, we are forced to rely on data from our state estimator. That limits our ability to fit data for global x and y coordinates, so we restrict the optimization to fitting roll, pitch, yaw, and airspeed. We use a prediction-error minimization framework (PEM) in which computes total error over a segment of data (as opposed to one-step acceleration error like equation-error methods).

This optimization is sensitive to a number of factors. Firstly, one must estimate the delay in the system carefully. To do this, we used a motion capture system[1] with a known delay (3 ms) and placed a marker on one of the elevons. We then commanded a movement and recorded the input time and delay until the motion capture system measured a movement on the control surface. In our system, the closed-loop delay was 20 ms.

Secondly, PEM is relatively sensitive to initial conditions. To address this, we used the System Identification Toolbox to simultaneously fit initial conditions on every state other than roll, pitch, yaw, and airspeed when performing the optimization. To ensure that this method did not cause us to over-fit our data, we set the optimization to run on data sets between 0.5 and 1.5 seconds. We found that shorter lengths would cause the system to use the selection of initial conditions to over-fit the data, giving useless parameter estimates. Longer segments, however were sensitive to errors in the initial condition and limits of the model class, so could not fit the data well. Overall, our limited ability to measure states other than the rotation angles and airspeed were significant limiting factors in obtaining an accurate model of the system's dynamics. While the model presented here is sufficient for control, we expect future work could improve it, starting with a more careful use of the state estimator's angular-rate outputs for initial conditions and extending lift and drag estimates beyond a flat plate, perhaps by using radial basis functions similar to [77] or an

[1]Northern Digital Inc. 3D Investigator. `http://www.ndigital.com/msci/products/3d-investigator`

airfoil simulation environment like [28].

3.2 State Estimation

Estimating the state of the aircraft online is critical to robust, stable control. This experiment exclusively uses onboard sensors, ruling out any possibility of motion capture or other external state estimation apparatus. Furthermore, GPS can be unreliable in the presence of dense obstacles, so we exclude it from consideration[2].

Reliable and stable GPS-denied state estimation with inertial sensors is currently an open problem. In this case, however, we do not need good long-term estimates of our position. Pushbroom stereo only requires relative position estimates that are accurate for 1-2 seconds, which we can achieve using existing techniques and onboard sensing. In particular, with a 3-axis accelerometer and gyroscope, we can reliably estimate roll, pitch, yaw, and their derivatives using the Kalman filter from [16][3]. By integrating a barometric altimeter for absolute measurement of altitude, we can measure z. With a pitot tube airspeed sensor for forward velocity and a zero side-slip assumption, we have reliable estimates of U, $V = 0$, and W (recall that U, V, and W are the x, y, and z velocities in the body frame). Thus we have reliable estimates for all state variables excepting x and y and can use pushbroom stereo. Remarkably, we did not need to modify the default parameters for most of the state estimator's gains (Table 3.3).

3.2.1 Managing Covariance with Unbounded x and y

Without reliable, global estimates of position, the state estimator, rightly, reports a rapid and unbounded growth in covariance for those variables. Over long periods of time, this growth causes the estimator to become unreliable in the other states as slight abnormalities

[2]The aircraft does have a GPS onboard for logging and debugging purposes, but it is never used in the state estimator or control systems. The author notes that some flights were flown with the GPS disconnected to prove that this was the case.

[3]Available as Pronto with changes from this thesis integrated at: `https://github.com/ipab-slmc/pronto-distro`

Parameter	Value (*units*)	Selection Method
sigma0_vb	0.15 m/s	Default
sigma0_chi_xy	3.0 deg	Default
sigma0_chi_z	3.0 deg	Default
sigma0_delta_xy	0.5 m	Default
sigma0_delta_z	1.0 m	Default
sigma0_gyro_bias	0.0 $^\circ/s$	Disabled
sigma0_accel_bias	0.0 m/s^2	Disabled
q_gryo	0.5 $^\circ/s$	Default
q_accel	0.2 m/s^2	Default
q_gyro_bias	0.0 $^\circ/s^2$	Disabled
q_accel_bias	0.0 $m/s^2/s$	Disabled
timestep_dt	140 Hz	IMU rate
airspeed_r	15 m/s	Trial and Error
altimeter_r	5.0 m	Trial and Error
sideslip_r	5.0 m/s	Trial and Error

Table 3.3: State estimator parameters.

accelerations and other measurements are explained with large changes in position. To limit this growth, we artificially reset the covariance for x and y at the beginning of each trajectory. In this way, we accept that we are measuring *relative* position from the start of the trajectory, instead of a global position. For convenience, we do not force the estimator to actually reset its estimate of x and y, just the covariance. Section 3.4.2 explains how we move trajectories into relative position coordinates. When executing a time-invariant trajectory, such as straight and level flight, we reset the position covariance on a schedule.

To preserve the maximum amount of information in the estimator, when the estimator is already running, we only modify the two rows and two columns corresponding to x and y in the covariance matrix. If the estimator is previously offline or for some other reason has never reported its state, we are forced to completely reset its covariance matrix.

3.3 Trajectory Libraries

Our system relies on trajectories computed offline and then selects a trajectory to execute online, similar to [32]. We rely on two methods to build individual open-loop trajectories:

trajectory optimization and a direct-from-data approach. We then apply time varying linear quadratic regulators (TVLQR) to the open-loop trajectories for create a feedback system which we execute online.

3.3.1 Trim Conditions

Flight conditions with zero acceleration are considered trim conditions. Ignoring battery capacity, these conditions, such as straight and level flight, constant velocity climbs, and gentle turns are stable forever. Using the model above, we can set up a feasibility search to find trim conditions. Recall that the time derivative of our state vector has six acceleration terms. In the global frame:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} & \underbrace{\ddot{x} & \ddot{y} & \ddot{z} & \ddot{\phi} & \ddot{\theta} & \ddot{\psi}}_{\text{accelerations}} \end{bmatrix}^T$$

We can search for a trim condition by setting those terms to zero and searching over both the state and control:

$$\text{find } \mathbf{x}, \mathbf{u}$$

$$\begin{aligned} s.t. \\ &\text{accelerations} = 0, && \Leftarrow 6 \text{ nonlinear contstraints} \\ &\mathbf{u} \geq \mathbf{u}_{min}, && \Leftarrow 3 \text{ linear contstraints} \\ &\mathbf{u} \leq \mathbf{u}_{max} && \Leftarrow 3 \text{ linear contstraints} \end{aligned}$$

In practice, the particular result will depend on the initial guess, but any result will be a trim condition of the model for straight and level flight. This problem solves in under 0.1 seconds. By constraining $\dot{z} > 0$ or $\dot{\psi} > 0$, one can search for a climb or turn respectively.

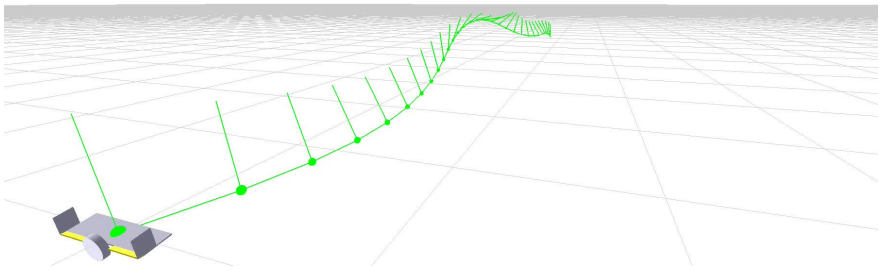


Figure 3-8: Knife-edge trajectory optimized and simulated in Drake.

3.3.2 Trajectory Optimization

To build trajectories with non-zero, time-varying accelerations directly from the aircraft model, we use a direct-collocation method like [107] implemented in Drake [110]. Each trajectory requires a hand-designed (but not particularly complicated) set of constraints or cost function to generate the desired result. In other words, one might add a constraint on the final position to design a trajectory that climbs, dives, or turns. Figure 3-8 shows a knife-edge trajectory, with a roll contraint of 90° at $t = t_F/2$, being optimized.

3.3.3 Trajectories from Data

Building trajectories from data involves flying the aircraft by hand along the desired trajectory. The advantage of this method is that the open-loop path is correct (does not have errors introduced by the model), up to the accuracy of the state estimator. Obviously, it can be difficult to fly the exact trajectory required, but in practice we found that a small number of attempts was sufficient. To control the trajectory online, we use our model to build a TVLQR controller around the flight states.

3.4 Feedback Control

3.4.1 Time Invariant Control for Trim Conditions

Errors in the model, disturbances such as wind, state estimation error, etc., require feedback control to keep the aircraft flying along the desired path. To stabilize trim conditions, we can use a standard, time invariant LQR controller:

First, we define error coordinates:

$$\bar{\mathbf{x}} = \underbrace{\mathbf{x}}_{\text{current state}} - \underbrace{\mathbf{x_0}}_{\text{trim state}}$$

$$\bar{\mathbf{u}} = \underbrace{\mathbf{u}}_{\text{current input}} - \underbrace{\mathbf{u_0}}_{\text{trim input}}$$

where $\bar{\mathbf{x}}$ gives the error in the state vector and $\bar{\mathbf{u}}$ gives the additional control action beyond that specified by the trim condition. Next, we can linearize about the trim condition resulting in a linear model of the standard form $\dot{\bar{\mathbf{x}}} = A\bar{\mathbf{x}} + B\bar{\mathbf{u}}$.

Given that linearizion, we can apply LQR to produce K, a gain matrix that stabilizes the trim condition online. In practice, this system is very capable. Figure 3-9 shows the system recovering from inverted flight.

3.4.2 Time Varying Control

To provide feedback control for time-varying trajectories, we use a time varying linear quadratic regulator (TVLQR) as in [111]. This controller uses the same error coordinates as above, but we discretize along the trajectory to produce a time-varying linear model of the form $\dot{\bar{\mathbf{x}}} = A(t)\bar{\mathbf{x}} + B(t)\bar{\mathbf{u}}$. By building an LQR controller at each discretization point along the trajectory, we can build a time-varying gain $K(t)$. We compute the additional control action online with this gain: $\bar{\mathbf{u}} = -K(t)\bar{\mathbf{x}}$, giving a total control of $\mathbf{u} = \mathbf{u_0}(t) - K(t)\bar{\mathbf{x}}$.

We take advantage of the fact that the aircraft's dynamics are invariant to x, y, z, and ψ (yaw), substantially reducing the number of trajectories required in a library. Thus, when a

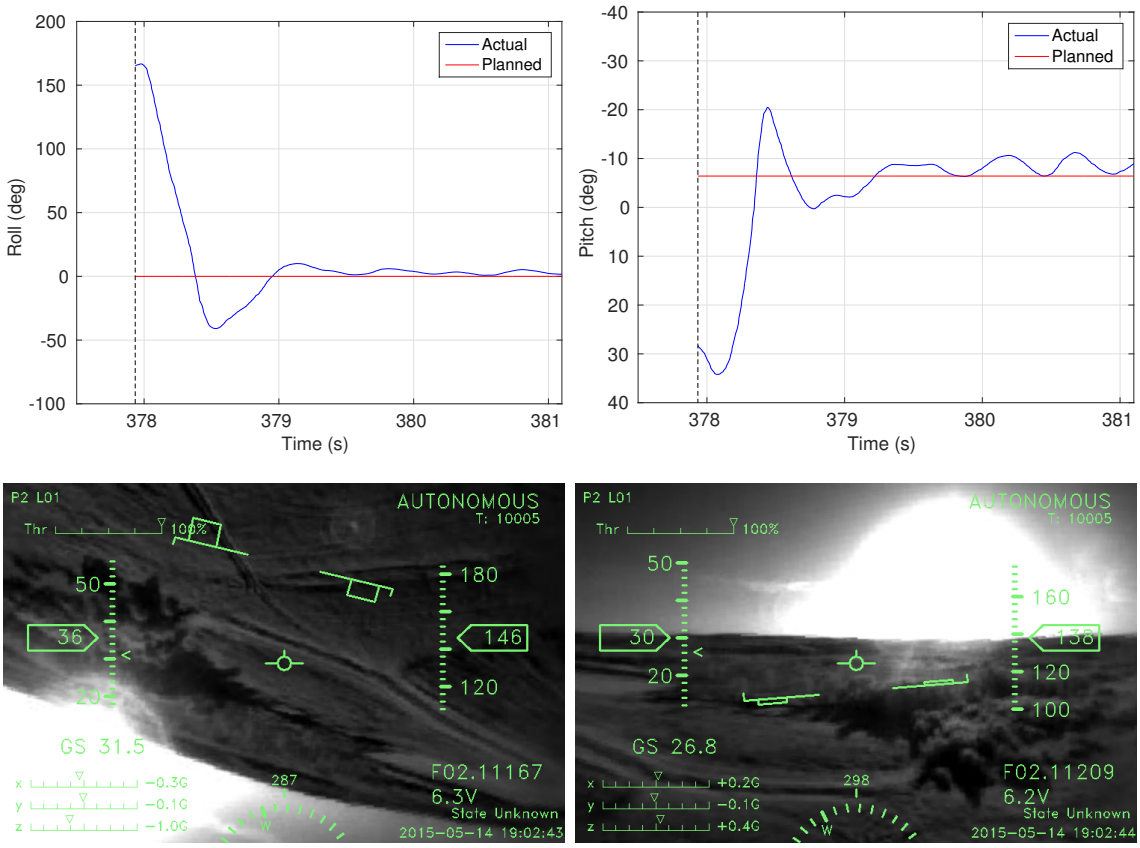


Figure 3-9: Top: roll and pitch traces for autonomous recovery from inverted flight. The controller is enabled at the black vertical line. Bottom: Onboard view at the beginning and end of the recovery.

trajectory is selected online, we record those four initial states and transform the trajectory along those coordinates.

Time Varying Control Results

To analyze the tracking of this system, we flew the aircraft away from obstacles, activated trajectories manually, and recorded the states. Note that since all experiments were performed in the field while traveling at substantial speed, we do not have ground truth or motion capture data for these flights. The plots below are based on the state estimator.

Figure 3-10 shows the system's tracking of a knife-edge maneuver. The tracking is not perfect and likely could be improved with further system identification and careful tuning. In practice, it tracks sufficiently well for complicated obstacle avoidance maneuvers. Other trajectories are shown in Chapter 4.

Control Results for Trajectories from Data

Figures 3-12 and 3-13 present tracking for a left turn trajectory that was built by replaying flight data online. The tracking in this case is worse than that from above when using the same controller and gains. Since these control gains were not tuned for this case, we expect that we could improve tracking without substantial effort. Regardless, both systems are capable of generating trajectories useful for obstacle avoidance.

3.5 Online Planning

3.5.1 Online Planning Algorithm

Given a trajectory library as described in Section 3.3 and a point cloud from pushbroom stereo (Chapter 2), we chose trajectories online to avoid obstacles. This selection process need not be optimal, but must run in realtime on embedded hardware. To that end, we discretize each trajectory in time, check distance to each point in the point cloud, and chose the trajectory that maximizes the closest encounter to the obstacles (maximizes the

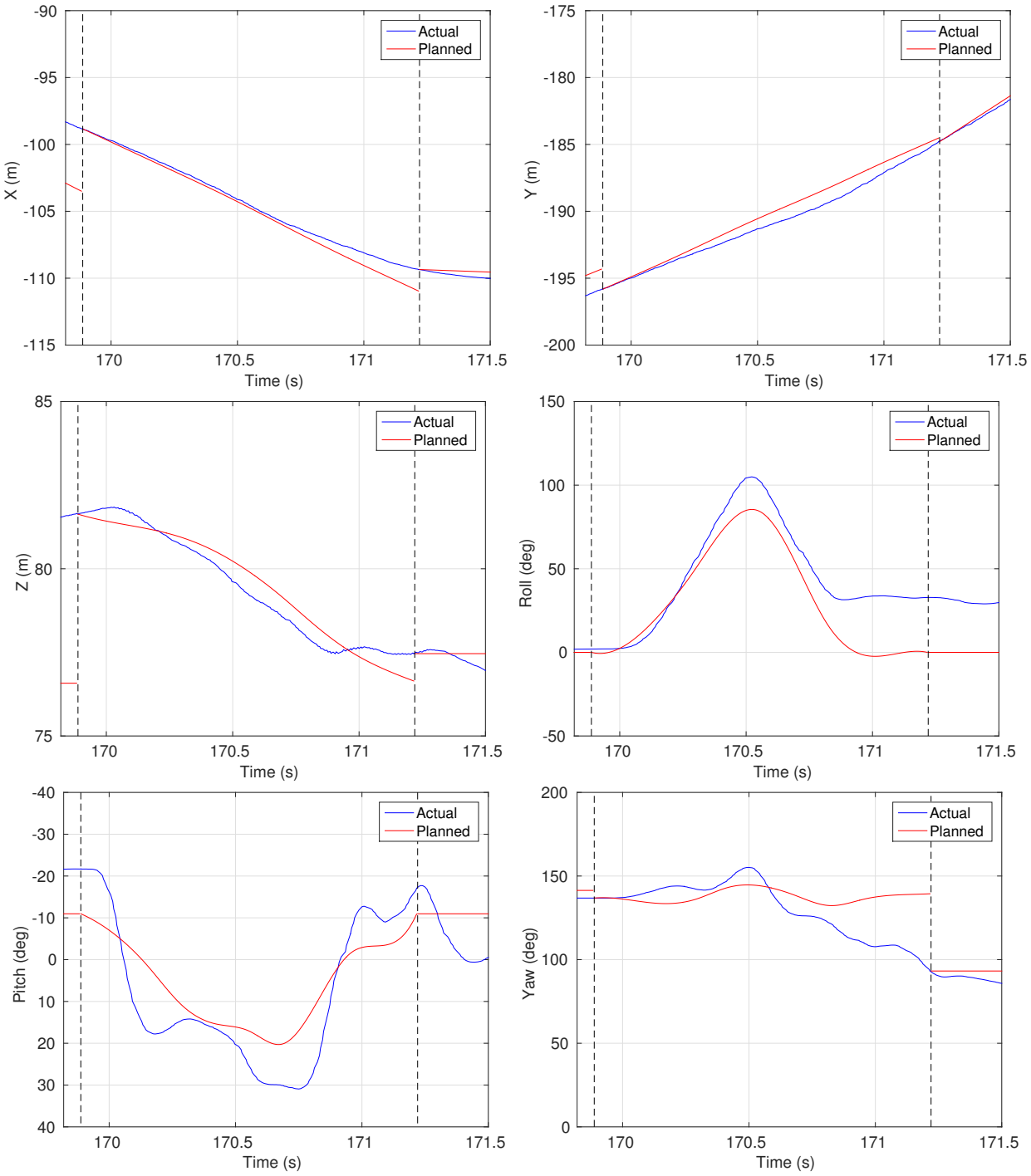


Figure 3-10: Tracking of a knife-edge trajectory with gains on all dynamically-relevant states. The knife-edge trajectory starts and ends at the dashed vertical lines. Figure 3-11 shows the control actions associated with this trajectory.

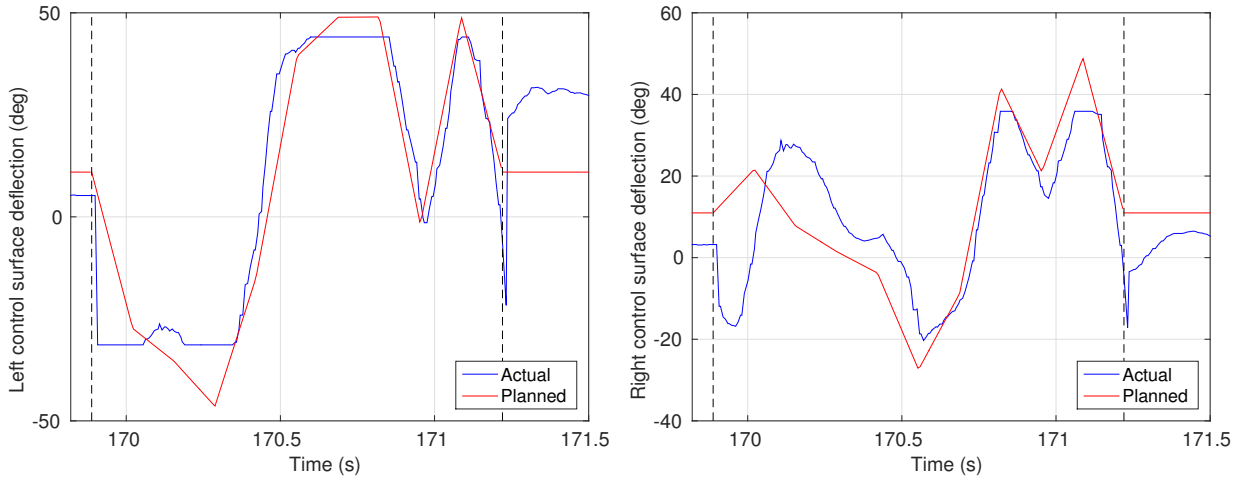


Figure 3-11: Control actions for the knife-edge trajectory.

minimum distance). Figure 3-14 outlines the system and Listing 1 shows the trajectory selection algorithm.

3.5.2 Point Cloud Management

To enable fast nearest-neighbor searches through a potentially large point cloud, we use an octree structure implemented in PCL [92]. Pushbroom stereo relies on our state estimate which is not accurate in position over long time horizons. Thus, the obstacle information we collect is local, and we should avoid accumulating it in the octree for long periods of time. To address this issue, we build two octrees simultaneously, and seamlessly swap between them on a clock. In the implementation described here, an octree lives for 4 seconds before being discarded. To ensure that we never miss a potentially dangerous obstacle when discarding an octree, we never swap octrees unless the new one has been accumulating points for at least half of its nominal life (2 seconds in this case). Based on our stereo range (10 meters), the stall speed of the aircraft (7-8m/s), and its limited turning radius, the aircraft will have flown well past any obstacles detected prior to 2 seconds earlier.

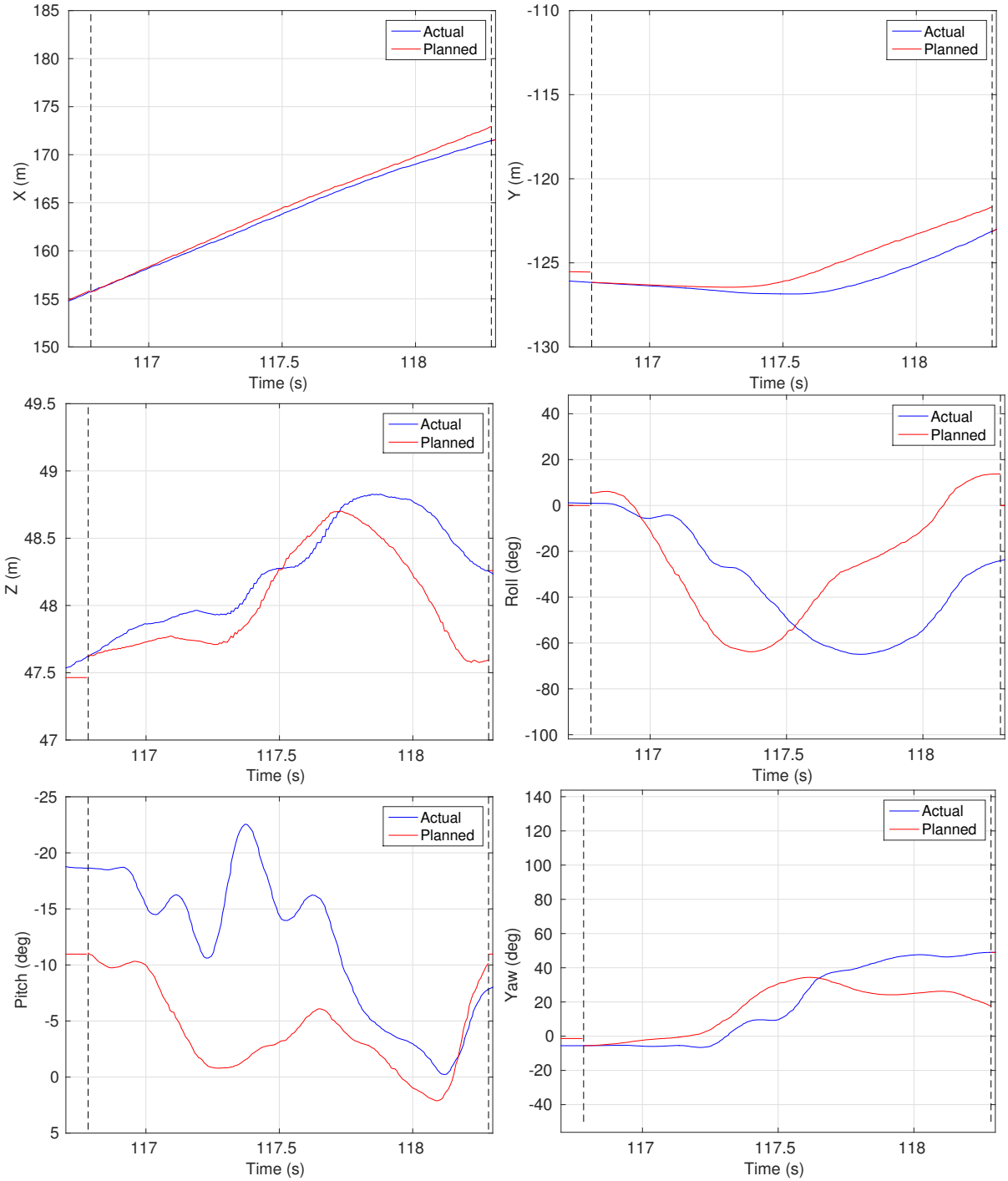


Figure 3-12: Tracking for a left turn generated from flight data, using the same TVLQR controller and gains as the trajectories generated from trajectory optimization. Clearly there is some delay in the execution of the roll, but the yaw tracking (the most important for avoidance) tracks well enough for obstacle avoidance maneuvers. Figure 3-13 shows the control actions associated with this trajectory.

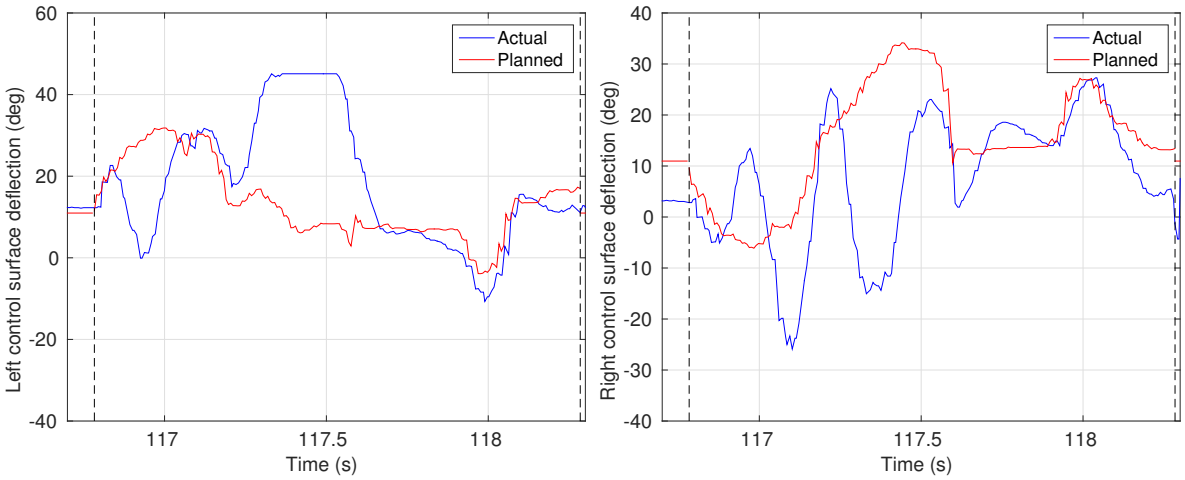


Figure 3-13: Control actions for a left turn generated from flight data. The gains are the same as in Figure 3-11, but could clearly be optimized for this case. Regardless, the trajectory executes sufficiently well for obstacle avoidance.

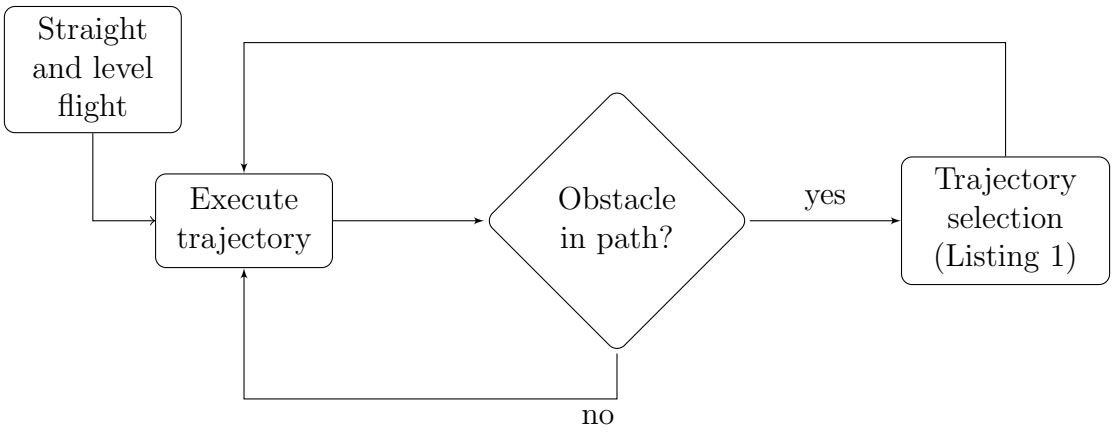


Figure 3-14: Diagram of the online planning system. An obstacle is deemed to be in the path if the minimum distance between the trajectory and the point cloud is below a threshold.

input :
L trajectory library
P point cloud
d current trajectory's distance to point cloud
m minimum improvement to switch trajectories
s safety distance
output:
L_i, selected trajectory
foreach *trajectory* T_i *in* L **do**
  `// compute the distance between this trajectory and the point cloud`
  $D_i = \min(\text{distance}(T, P))$
  if $D_i > s$ *and* D_i - $d > m$ **then**
    return T_i `// this trajectory is safe, so use it`
  end
end
`// no trajectories were completely safe`
$i = \text{IndexOfMaximum}(D)$
if D_i - $d > m$ **then**
  return T_i `// found a better trajectory than we had before`
else
  return *0* `// do not change trajectories`
end

Algorithm 1: Trajectory selection algorithm.

3.5.3 Online Planning Computation

To test the online planning system in isolation, we flew the aircraft in free space and triggered the presence of virtual obstacles manually. The virtual obstacles are triggered by a switch on the safety pilot's remote and reported via the same mechanism pushbroom stereo uses. In these experiments, with 10 trajectories in the library, system chooses a trajectory an average of 18.9 ms after the report of an obstacle and takes a control action within 0.5 ms of that (N = 12 executions over 2 flights). Further results and analysis are presented in Chapter 4.

3.6 State Machine

The aircraft control system uses a state machine to manage the initial takeoff, climb to cursing altitude, and triggering transitions between trajectories. The state machine's transition diagram (implemented with SMC[4]) is shown in Figure 3-15.

[4]State Machine Compiler by Charles W. Rapp `http://smc.sourceforge.net`

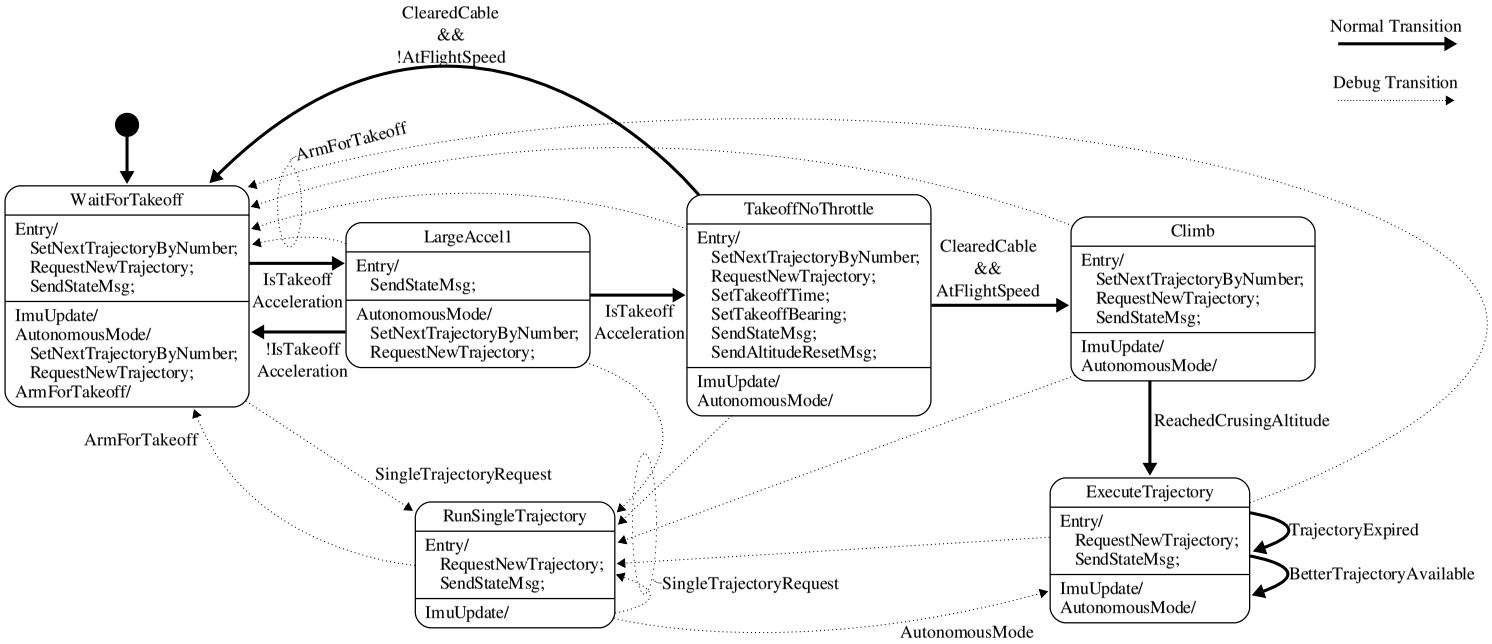


Figure 3-15: State transition diagram. The system starts in the `WaitForTakeoff` state, transitions through `LargeAccel1`, `TakeoffNoThrottle`, and `Climb` before running the main autonomous state, `ExecuteTrajectory`. The `RunSingleTrajectory` state is provided for debugging and testing. Transitions activated by the user for debugging are marked with dotted lines.

Chapter 4

Experiments Near Obstacles

We built a highly-dynamic aircraft to test the above algorithms in the field. The airframe exhibits a roll-rate of over 300 degrees per second and a top diving speed of 22+ m/s (50+ MPH), requiring precise and fast control systems.

4.1 Aircraft Platform

Our hardware (Figure 4-1) is based on a Team Black Sheep Caipirinha[1], modified to carry a substantial sensing, vision, and computation platform. We use two ODROID-U3[2] single-board computers, each with a quad-core ARM Cortex-A9 running at 1.7Ghz with 2GB of RAM and 64GB of eMMC solid state hard disk. Our IMU platform is a firmware-modified APM 2.5[3,4] providing a MPU-6000 based 3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer suite with an additional pitot tube airspeed sensor, barometric altimeter, uBlox GPS (unused for control or estimation), and radio/servo I/O. The aircraft has a 2-cell 7.4V LiPo 2000mAh battery, supporting approximately 15 minutes of flight time, and has a takeoff weight of 664 grams. Appendix C has a complete list of components.

[1]Team Black Sheep, `http://team-blacksheep.com/products/prod:tbscaipirinha`
[2]Hardkernel Co. Ltd, `http://hardkernel.com`
[3]3D Robotics, Inc. `http://3drobotics.com/`
[4]Firmware available: `https://github.com/andybarry/ardupilot/tree/arduread`

Figure 4-1: Aircraft hardware on launcher.

The stereo system is based on two Point Grey Firefly MV cameras[5] connected via USB 2.0 to one of the ODROID-U3s. The cameras are configured to use 2x2 pixel binning, giving 120 frame-per-second (fps) grayscale video at 376x240 resolution[6]. The cameras are not hardware synchronized, but instead rely on USB to stay in sync. We note that it is not possible to achieve both hardware synchronization and 120fps on these cameras.

We built a total of three nearly identical aircraft, all of which were flight tested and have a stall speed of approximately 7-8 m/s (15 MPH) and a top speed of around 22 m/s (50 MPH). All test flights utilize an onboard self-spooling 250 meter safety tether.

[5]Part #FMVU-03MTC-CS. `http://www.ptgrey.com`

[6]This mode is no longer listed in the current datasheet of the Firefly MV camera but does still exist on newly purchased cameras.

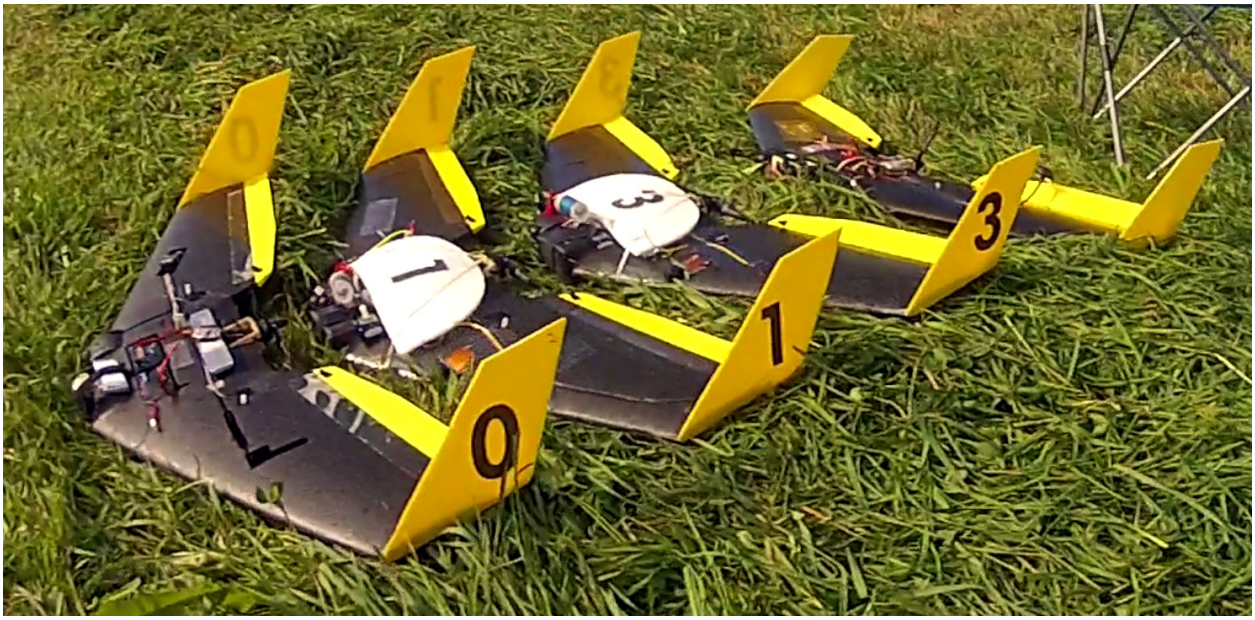


Figure 4-2: Set of flight aircraft. Numbers 1 and 3 are equipped for autonomous flight.

4.2 Experimental Setup

In addition to the experiments described above, we performed an obstacle-avoidance experiment near trees in an outdoor environment. The experiment is fully autonomous, including an automatic takeoff, climb, flight, and obstacle avoidance. We manually land the aircraft after the experiment is complete[7]. Figure 4-3 shows a diagram of the experiment's flight phases.

4.2.1 Obstacles

We present results from both artificial and natural obstacles. The artificial obstacles are approximately 15 ft high poles in a small PVC apparatus to hold them vertical. The natural obstacles are trees located at our field site (Figure 4-4). We avoid a variety of trees, including a number of fully grown trees and one large dead tree.

[7]We have performed successful autonomous landings, but it proved experimentally easier to land manually.

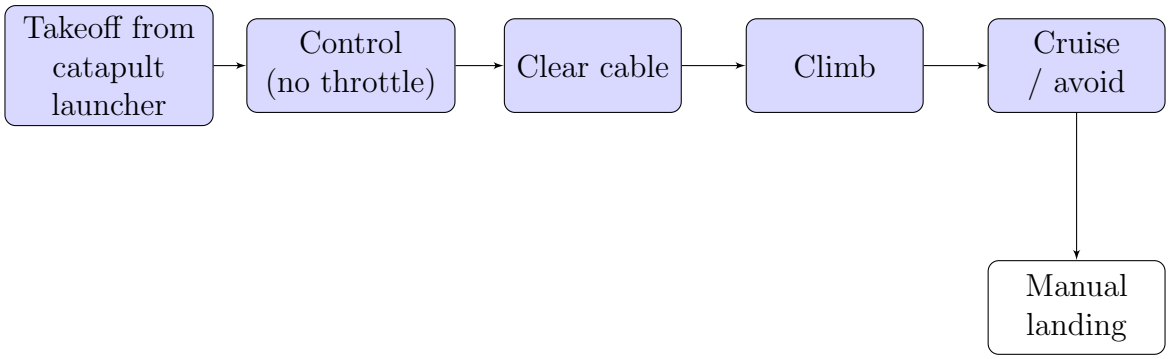


Figure 4-3: Experimental phases. Autonomous modes are labeled in blue. Section 3.6 presents the state machine responsible for these actions.

Figure 4-4: Example of an artificial obstacle (left) and natural obstacles (right).

Figure 4-5: Bungee launcher for the aircraft.

4.2.2 Takeoff and Climb

The system launches from a commercial bungee launcher[8] (Figure 4-5) and performs stabilization control until it has cleared the launch cable (t = 0.8 s). It then starts the motor and climbs. We automatically detect a launch event with a simple acceleration-based filter (See Section 3.6 for more information about aircraft states). As a safety precaution, the aircraft will not engage the throttle unless the airspeed indicates that it is moving over 8.0 m/s (17.9 MPH). Once the motor is running, the aircraft autonomously climbs to a specified cruising altitude and transitions to a sense-and-avoid state using data from pushbroom stereo.

4.2.3 Field Site

All flights near trees were performed at Westview Farms Creamery in Monson, MA with permission of the field's owner. The location features a number of large mowed fields in addition to some trees. Figure 4-6 shows an airiel view of the field site.

[8]Fan Jets USA `http://www.rc-electric-jets.com/`

Figure 4-6: Field site for obstacle avoidance tests. The trees above the red marker are the trees avoided in the experiments below. Imagery © 2015 Google, map data © 2015 Google.

4.3 Trajectories in Library

For most of the experiments near obstacles, we used a trajectory library with only seven trajectories (Table 4.1). Surprisingly, this library was sufficient for most of our flights, although we do not deny that a larger library could have improved performance (see Chapter 5). Despite the fact that, in this table, no time-varying trajectories produced via trajectory optimization are present, the aircraft was able to track those trajectories (see Section 3.4.2). Experimentally, it proved easier to manually fly the limited number of time-varying trajectories by hand.

4.4 Results

Over one field season, starting in March 2015 and ending in October, we performed 20 days of field testing in Monson, MA, starting with basic flight control and ending with obstacle avoidance near trees. In eight of those twenty days we tested near obstacles, starting with

Number	Description	Type	Length (sec)	Produced via
1	Straight	TI	∞	Model
2	Climb	TI	∞	Model
3	Takeoff (no throttle)	TI	∞	Model
4	Gentle left	TI	∞	Model
5	Gentle right	TI	∞	Model
6	Left jog	TV	2.45	Flight data
7	Right jog	TV	2.49	Flight data

Table 4.1: Trajectory library used in most obstacle avoidance experiments. TI stands for "time-invariant," which indicates the trajectory is at a trim condition (see Section 3.4.1). TV stands for "time varying" (see Section 3.4.2).

artificial poles and moving on to trees. In all, we flew the planes on approximately 136 flights. We performed 16 flights where the aircraft autonomously avoided an obstacle at flight speed. To put that in perspective, each one of those flights had the potential to destroy the aircraft in the event of a failure. Over the course of flight testing, we lost one airframe due to a tether malfunction that was unrelated to the obstacle avoidance system. The system failed to avoid an obstacle on 9 flights. Section 4.5 details the types of failures and suggests future work for their mitigation.

4.4.1 Aggregate Analysis

Obstacle type	Total flights	Successes	Success ratio
Artificial	4	4	100%
Pair of trees	4	4	100%
Many trees	18	8	44%

Table 4.2: Statistics for experiments near obstacles.

On 16 successful flights (Table 4.2), the aircraft flew over 1.5km, detected 7,951 stereo matches, executed 163 trajectories and spent 131 seconds flying in autonomous mode with an average speed of 12.1 m/s (27 MPH). Figure 4-9 presents the distribution of trajectories executed when flying near natural obstacles (see Table 4.1 for the trajectory library.) Figure 4-7 demonstrates the aircraft's onboard view during six of these avoidance maneuvers and Figure 4-8 shows a chase-camera view.

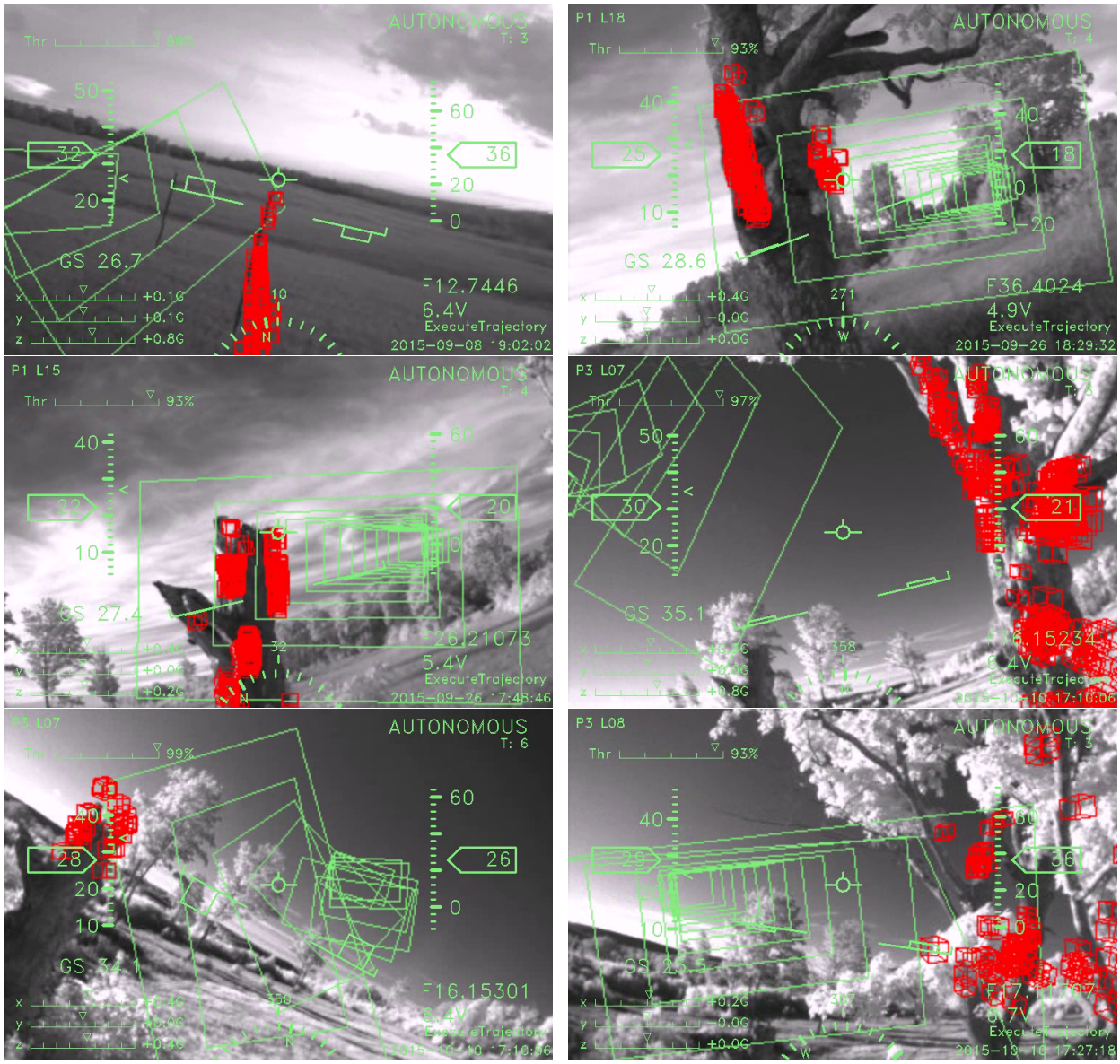


Figure 4-7: Onboard view of 6 different avoidance maneuvers. The red voxels show detected obstacles and the green squares show the planned trajectory as it is being executed. The top left view shows an artificial obstacle and the remaining 5 are trees.

Figure 4-8: Chase camera view of the autonomous aircraft avoiding a tree.

Table 4.2 presents statistics for our system in 3 different obstacle fields. The first, artificial obstacles, are approximately 15ft high poles. We placed one or two poles in the middle of an open field and flew the aircraft at them. The second, a pair of tress, was one live and one dead tree next to each other. We flew the aircraft at these trees and succeeded at avoiding them by going around or between them. Finally, we flew the aircraft at the same two trees but from a different angle, where there was an additional tree on the right and more trees in the path behind the initial two.

4.4.2 Analysis of a Single Avoidance Maneuver

In this section we analyze a single flight and avoidance maneuver in depth. In this maneuver, the aircraft avoided a small tree on the aircraft's right by choosing a maneuver to the left, then it saw a large tree on the left and avoided to the right, going between the two obstacles. While rolling, it detected a horizontal branch on the larger tree and chose a trajectory that avoided that branch by flying under it. Finally it cleared the two obstacles and flew into free space. Once in free space, the safety pilot switched the aircraft to manual mode and landed

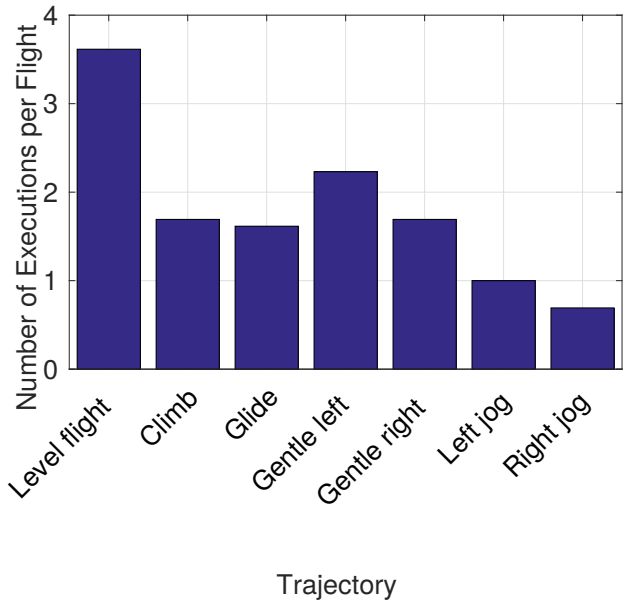


Figure 4-9: Graph of number of times each trajectory in the library was executed per successful flight near trees. Level flight is executed the most, since it is the default trajectory when there are no obstacles.

it. The aircraft covered 104.5 meters in autonomous mode, identified 459 stereo matches, flew at an average speed of 12.3 m/s (27.5 MPH), and experienced a maximum acceleration of 7.9 Gs on launch. Figure 4-10 shows this sequence of maneuvers from the onboard camera's view, Figure 4-11 shows an offboard view, Figure 4-12 presents the trajectories and obstacles in a 3D visualizer, and Figures 4-13 and 4-14 show the flight's planned trajectories and estimated tracking for position rotation, and control.

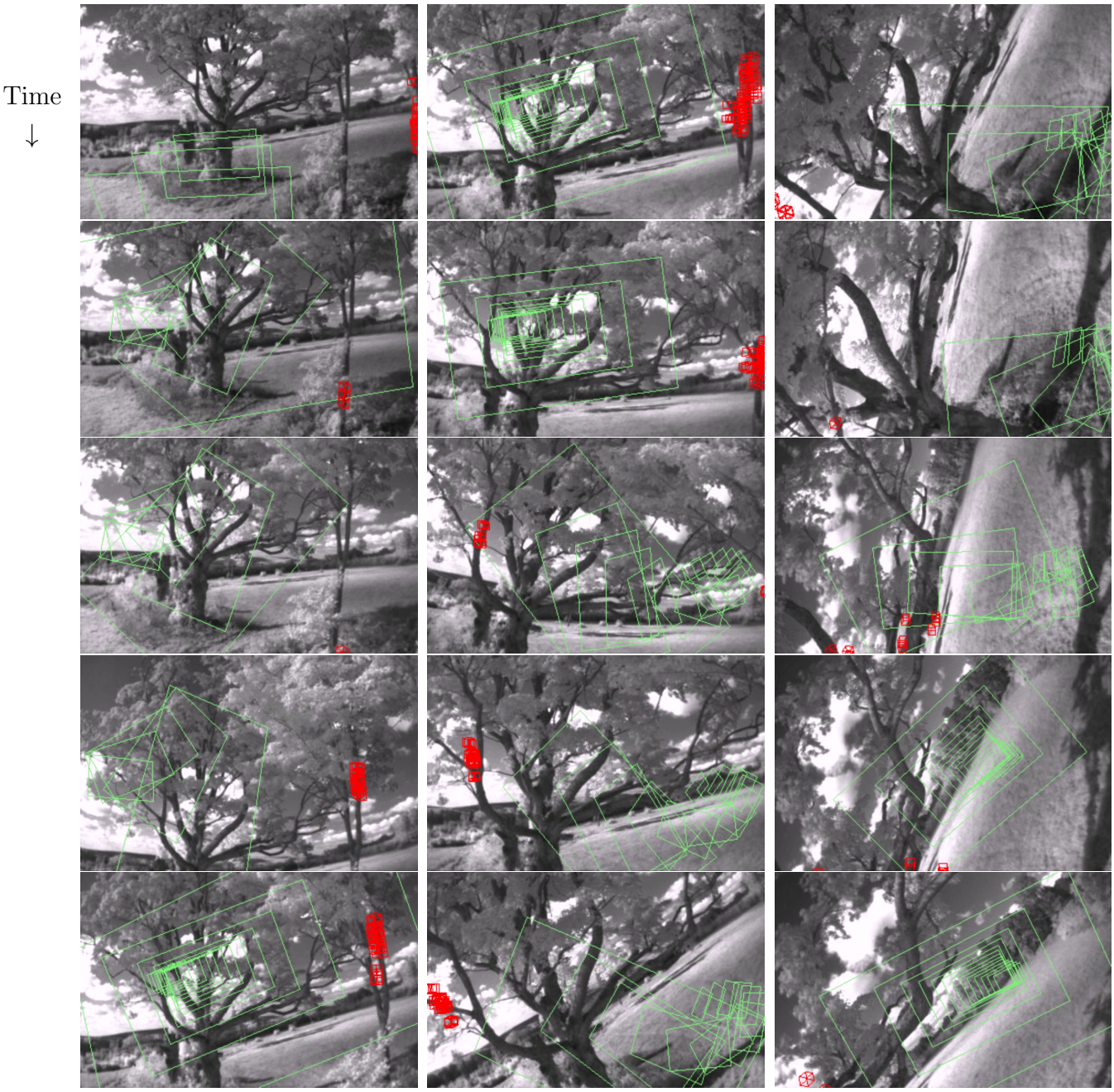


Figure 4-10: Autonomous obstacle avoidance from the onboard view. Time progresses down in columns and red voxels mark detected obstacles. The current trajectory is plotted as green boxes. Each frame is 0.0833 seconds (83.3ms) apart (1/10 actual framerate). The entire sequence in this figure lasts for 1.166 seconds and is sampled from a set of 140 frames.

Figure 4-11: Aircraft avoiding obstacles as viewed from a camera mounted on the tree seen in the left of the images in Figure 4-10. Each snapshot is 0.083 seconds apart and the total sequence spans 1.4 seconds.

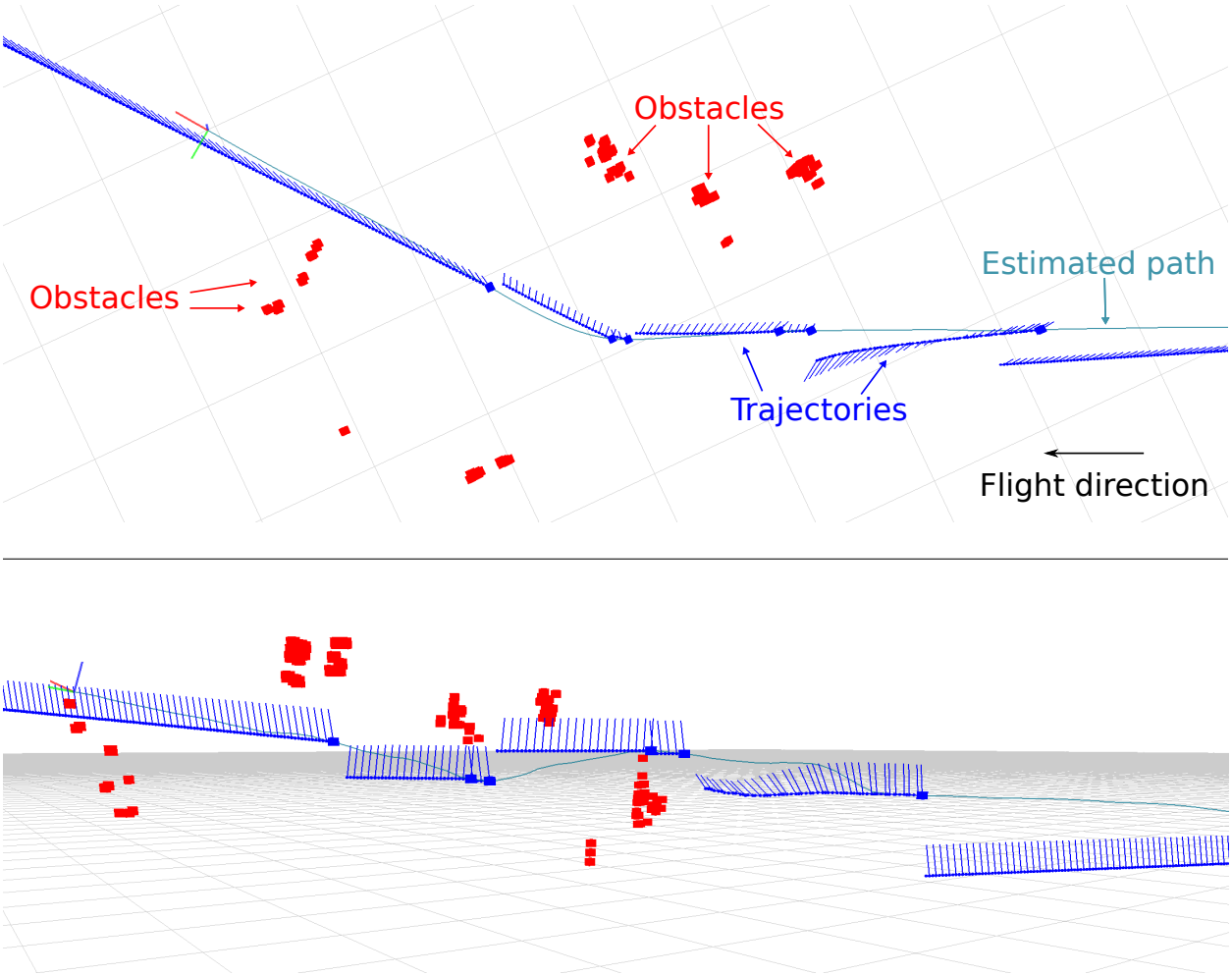


Figure 4-12: 3D visualization (top and side views) of the flight in Figure 4-10. Red boxes represent obstacles in the pointcloud, blue lines represent executed trajectories (including roll). The lighter solid line is the estimated path of the aircraft. The triad near the left represents the state of the aircraft near the end of the maneuver.

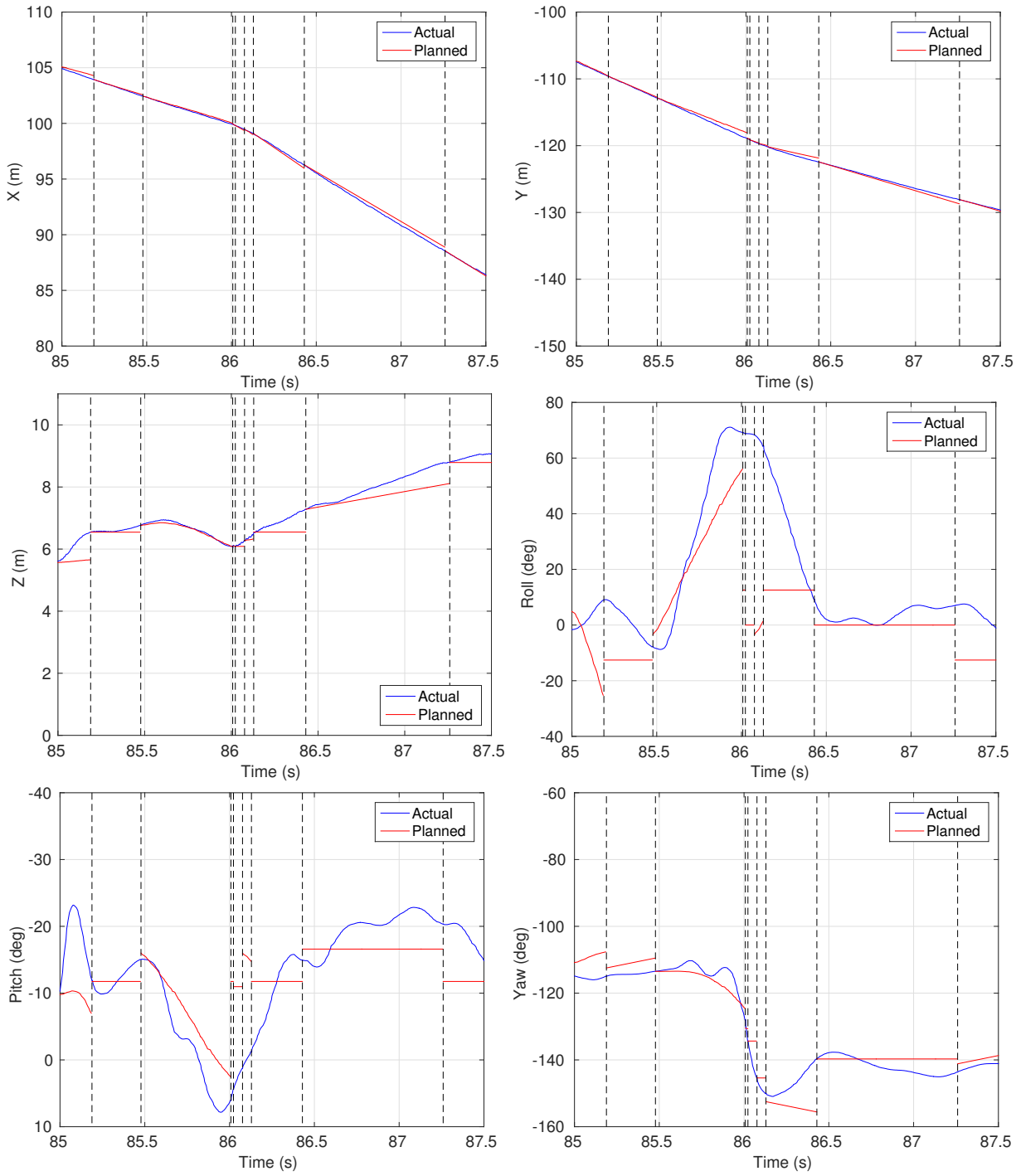


Figure 4-13: Planned and estimated tracking for an obstacle avoidance maneuver. Vertical dashed lines indicate a trajectory change. The first obstacle detection happens at $t = 85.15$ seconds. Notice the significant turn at $t = 86$ seconds where the aircraft dramatically alters its yaw angle to avoid an obstacle.

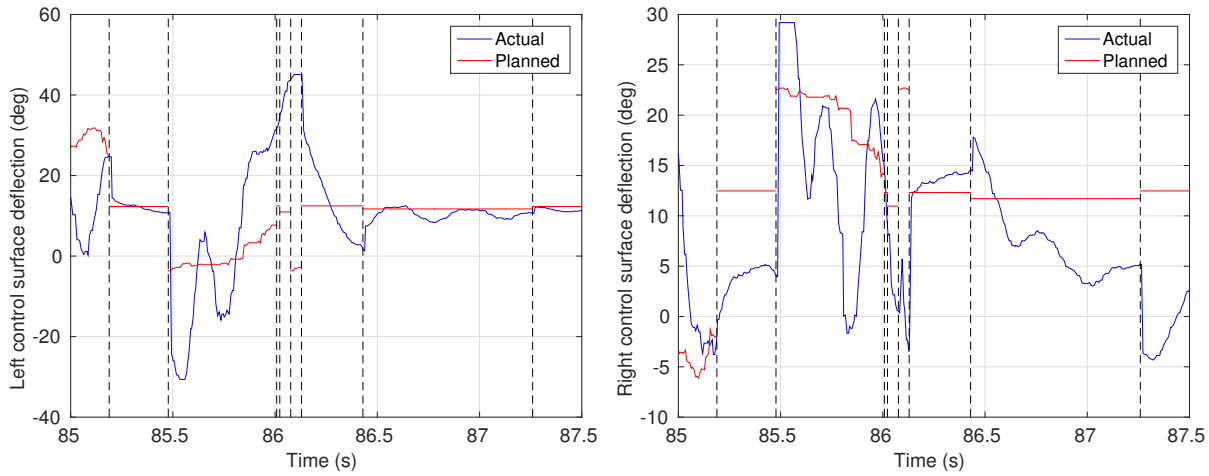


Figure 4-14: Planned and actual control inputs for an obstacle avoidance maneuver. The first obstacle detection happens at $t = 85.15$ seconds. We omit throttle here since it is commanded at, and runs at, approximately 100% throughout the entire maneuver.

Failure Type	Occurrences	Percentage of Failures (%)
Vision failures	*5*	*50%*
Failed to see obstacle	1	10
Poor calibration	2	20
No video data / unknown vision failure	2	20
Control failures	*5*	*50%*
Insufficiently rich maneuver library	2	20
Trajectory initial state	2	20
Loss of control	1	10
Total	10	100%

Table 4.3: Breakdown of system failures.

4.5 Failure Analysis

We pushed the system until it started to fail in the most complicated case. These failures provide insight into the shortcomings of the sensing, planning, and control framework. Overall, the failures break down into two main categories: failures of control and failures of the vision system (Table 4.3).

4.5.1 Control failures

A control failure is characterized by a situation where the vision system indicates that the aircraft is about to hit an obstacle, but the control system does not take appropriate action in time. Figure 4-15 presents an onboard view of this case.

In these data, control failures were caused by two primary issues: (1) an insufficiently rich maneuver library, and (2) trajectory initial state. The first is rather easy to understand. In some cases the maneuver library used was insufficient to avoid the obstacle. Figure 4-16 demonstrates a case where the aircraft approached the canopy of a tree. The best maneuver is likely to completely change direction, either by turning left or right at maximum rate. In this case, the maneuver library did not have such a maneuver, so the aircraft attempted to go through the canopy and failed.

The second case, trajectory initial state, requires more analysis. In these cases, the system correctly identifies an obstacle ahead, plans to execute a reasonable trajectory around the

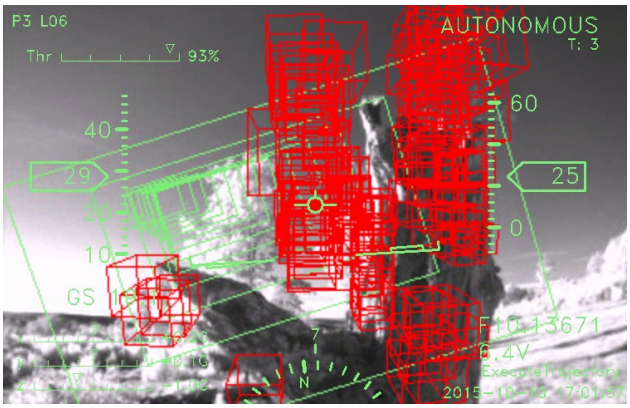


Figure 4-15: An example of a control failure. The vision system has clearly marked the dead tree ahead as an obstacle (red voxels), but the control system has failed to take appropriate corrective action in time. In this particular flight, the aircraft clipped its right wing but stabilized itself and continued flying.

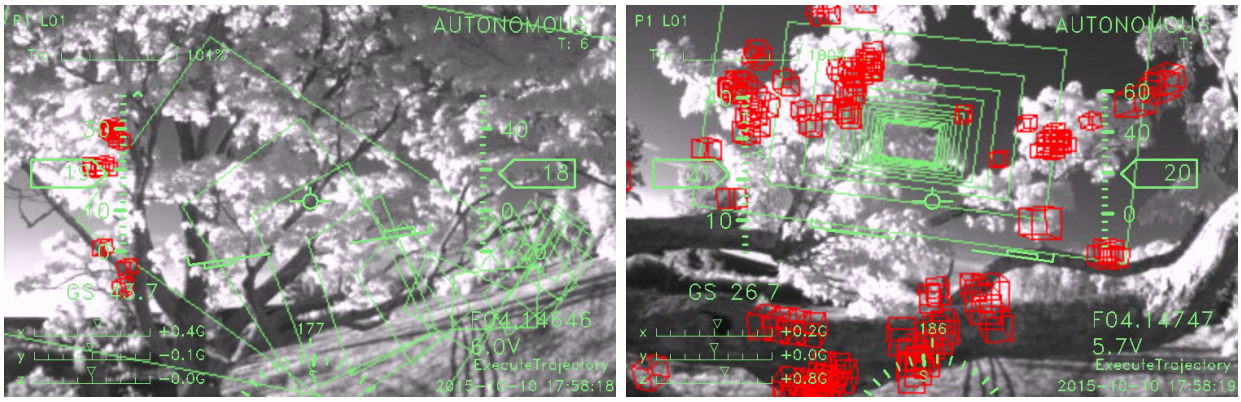


Figure 4-16: Failure case where the trajectory library was insufficient to avoid the obstacle. The left image shows the first frame where an obstacle is detected. The right image shows the aircraft about halfway through the tree's canopy and 0.2 seconds from impact.

obstacle, and then fails to fly the planned path. In this case, the problem lies with a limitation in our trajectory library framework. In particular, our trajectory library only has trajectories that start in one state (straight and level flight). Thus, if the aircraft begins a trajectory away from that state, we rely on the TVLQR controller to bring it progressively closer to the trajectory. In most cases, this strategy works well, but there are some that are troublesome.

In particular, in one flight the aircraft was rolled to the left when it saw an obstacle. The realtime planning system chose an aggressive left turn as an avoidance maneuver in response. As soon as the trajectory begin executing the aircraft rolls *right*. This happens because the controller is attempting to bring the starting state of the aircraft (rolled left) to the trajectory's starting state (level). Then, as the trajectory begins rolling left, the aircraft begins to stop its roll to the right, but it is already too late. Having taken the wrong initial action, the online planner detected that a climb trajectory had more clearance, and the aircraft attempted a climb. In this case, the aircraft's right wingtip clipped an obstacle, but the control system was able to recover and remain flying. In the other case, the aircraft hit the obstacle directly and was damaged. Figure 4-17 shows the various state variables and control actions during this flight.

Overall, a trajectory library with only one starting state worked surprisingly well in the field. There is clearly great potential for success with small trajectory libraries moving forward. To add robustness, however, more sophisticated handling of the starting condition is required. Our results are encouraging since they suggest that the number of starting states required for good performance may be quite small. The fatal situation is one in which the control action required is *opposite* of that initiated at the beginning of the trajectory. Thus, one might need starting states for left and right turns, climbs and dives, and level flight, but not for thousands of states in between.

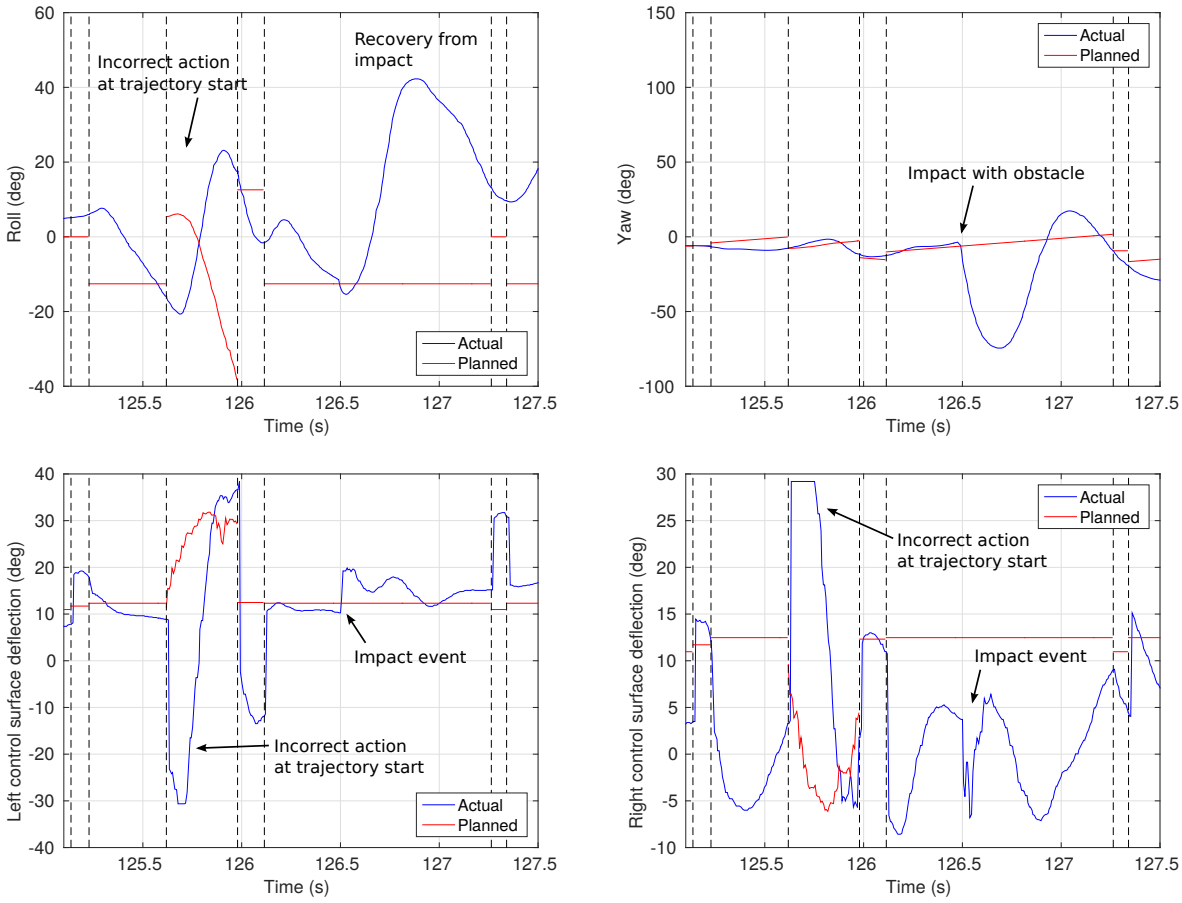


Figure 4-17: Roll, yaw, and control actions during a "trajectory initial condition" failure. In the top left (roll) the large discrepancy at the beginning of the trajectory is apparent, along with the incorrect control action taken (bottom plots). At approximately $t = 126.5$ the aircraft's right wingtip collides with an obstacle, producing a substantial yaw (about 74°), but the controller is able to recover and continue flying.

4.5.2 Vision failures

In all but one flight where the vision system failed and we had recorded data, post-flight analysis shows that poor stereo calibration was at fault. We note, however, that on two of the five flights with vision failures, the system lost power before transferring the video recording from RAM to disk, and thus it is difficult to determine why the vision system failed during those flights. The flight where pushbroom stereo failed to see the obstacle was headed directly at a tree canopy. The leaves in the canopy offer less contrast than trunks and branches, so the system has more difficultly detecting them.

A more secure mounting technique for the cameras, especially one that rigidly connected them together would improve reliability. Furthermore, a system that could detect or even fix minor calibration errors without the need for a full calibration would be useful. This second one would not need to run online, but could be something that ran on every system boot.

Chapter 5

Conclusion

We have presented a complete, working system for outdoor obstacle avoidance with no prior knowledge of the environment and all sensing and processing done onboard the aircraft. To the best of our knowledge, this system can navigate complex natural environments faster than any previous MAV to date.

Pushbroom stereo enables the system to detect obstacles with stereo cameras at 120 frames-per-second, which allows us to perform obstacle avoidance at up to 14 m/s (31 MPH). Critically, pushbroom stereo can achieve this framerate on a lightweight processor allowing us to build a completely self-contained obstacle avoidance unit.

There are a number of weaknesses in our approach that have clear solutions with potential to improve the reliability and performance of the system. In this work, we never explored searching for stereo matches at multiple depths (beyond checking for horizontal invariance,) but the potential for self-correcting estimates and reliability checks for false positive and false negative situations is clear. This multi-depth check is possible with a surprisingly small improvement in processing power because the preprocessing steps of rectification and edge filtering need not be repeated for additional depth checks. Moreover, new lightweight processors offer GPU tools not previously available, so a GPU implementation of pushbroom stereo could further increase framerate or resolution.

Secondly, a trajectory library with multiple starting states would improve control perfor-

mance. Our aircraft collided with obstacles because of this limitation, which again, would not require much (in any) improvement in the onboard hardware. With multiple precomputed trajectory libraries starting in different states, a system could immediately eliminate any trajectories that did not have nearby starting states. Thus, main cost of searching over the pointcloud would not be increased.

In addition, verification of the controllers for each trajectory could provide insight and even safety guarantees when choosing trajectories from the library. Using sums-of-squares programming we can build "funnels" such as those presented in [75] and expanded in [66] that would give us guarantees about when it was safe to switch trajectories, preventing some of the failures described in Section 4.5.1. Majumdar has even shown these techniques working in flight [67]. Our data suggests that the required number of initial conditions are low, but verified funnels would give a more satisfying and complete answer.

Even with verified controllers, however, we could not effectively guarantee safety of the system. Some type of verification for the pushbroom stereo system, be it from statistics, high-fidelity simulation, or other means, is required to build safety metrics for the closed-loop autopilot. Natural features, lighting conditions, calibration error, lens flare, etc. must all be addressed to build metrics that we might ultimately need to be confident when deploying autonomous obstacle avoidance systems in critical or dangerous applications.

This thesis presents the fastest autonomous MAV avoiding complex natural obstacles to date. We hope that others will build on this work to create new, sophisticated autopilots capable of even higher performance and utility in the future.

Appendix A

Open Source Software and External Resources

A.1 Software

- Flight code and pushbroom stereo: `https://github.com/andybarry/flight`
- Simulation and system idenfitication: `https://github.com/andybarry/simflight`
- Drake simulation environment: `http://drake.mit.edu`
- State estimator: `https://github.com/ipab-slmc/pronto-distro`
- APM Firmware: `https://github.com/andybarry/ardupilot/tree/arduread`

A.2 Videos

- High-speed obstacle avoidance:
  - `https://www.youtube.com/watch?v=_qah8oIzCwk`
  - `https://www.youtube.com/watch?v=iksfHQkkq88`
- Pushbroom stereo:
  - `https://www.youtube.com/watch?v=cZE01bJIgvQ`

Appendix B

Parameters and Diagrams

B.1 Stereo Parameters

Parameter	**Value**	**Comments**
Block size	5	Size of pixel blocks to match
Laplacian aperture size	3	Laplacian `ksize`
Interest operator limit	860	Threshold for discarding pixel regions based on Laplacian filtering
Horizontal invariance multiplier	0.5	Scaling for how much more likely a horizontal-invariance check will match. Set to 1 for equal likelyhood, set to 0.5 for 2x decrease in score (higher score means worse match)
Score threshold	54	Threshold to consider a pixel block a match

B.2 Model Parameters

Parameter	**Value (*units*)**	**Identification Method**
Mass (m)	0.648412 kg	Measured
Wing span (w_s)	0.8636 m^2	Measured
Wing chord (w_c)	0.2097 m	Measured
Elevon span (e_s)	0.31115 m^2	Measured
Elevon chord (e_c)	0.0402 m	Measured
Elevon moment arm (X) (e_{a_x})	0.12495 m	Measured
Elevon moment arm (Y) (e_{a_y})	0.276225 m	Measured
Winglet area (t_{area})	0.01944076 m^2	Measured
Linear component, left servo (m_l)	0.002195 ($rad/\mu s$)	Measured
Affine component, left servo (l_0)	-3.017 (rad)	Measured
Linear component, right servo (m_r)	-0.0019 ($rad/\mu s$)	Measured
Affine component, right servo (r_0)	2.811 (rad)	Measured
Linear component, thrust (k_m)	0.01004 $\mu s/N$	Experiment
Affine component, thrust (k_0)	12.17 N	Experiment
Moment of inertia (J_x)	0.0153 $kg \cdot m^2$	CAD and Experiment
Moment of inertia (J_y)	0.0052 $kg \cdot m^2$	CAD and Experiment
Moment of inertia (J_z)	0.0184 $kg \cdot m^2$	CAD and Experiment
Elevon lift factor	1.30	Fit
Elevon drag factor	0.101	Fit
x-axis body drag factor (b_{d_x})	0.0443	Fit
x-axis rate-dependent moment (M_P)	-0.0740	Fit
y-axis rate-dependent moment (M_Q)	-0.1150	Fit
z-axis rate-dependent moment (M_R)	0	Fit

B.3 State Estimator Parameters

Parameter	**Value (*units*)**	**Selection Method**
`sigma0_vb`	0.15 m/s	Default
`sigma0_chi_xy`	3.0 deg	Default
`sigma0_chi_z`	3.0 deg	Default
`sigma0_delta_xy`	0.5 m	Default
`sigma0_delta_z`	1.0 m	Default
`sigma0_gyro_bias`	0.0 $^\circ/s$	Disabled
`sigma0_accel_bias`	0.0 m/s^2	Disabled
`q_gryo`	0.5 $^\circ/s$	Default
`q_accel`	0.2 m/s^2	Default
`q_gyro_bias`	0.0 $^\circ/s^2$	Disabled
`q_accel_bias`	0.0 $m/s^2/s$	Disabled
`timestep_dt`	140 Hz	IMU rate
`airspeed_r`	15 m/s	Trial and Error
`altimeter_r`	5.0 m	Trial and Error
`sideslip_r`	5.0 m/s	Trial and Error

B.4 System Identification Parameters

Parameter	**Value**	**Comments**
Closed loop delay	20 ms	measured (see Section 3.1.2)
Block length	0.5 - 1.5 s	Varies depending on run
Elevon lift min/max	0 / 5	
Elevon drag min/max	0 / 5	
`M_P` factor min/max	-10 / 0	Roll damping moment
`M_Q` factor min/max	-10 / 0	Pitch damping moment
`M_R` factor min/max	-10 / 0	Yaw damping moment
PEM regularization λ	0.01	
PEM regularization R	diag(0.01)	
Roll weight	1	
Pitch weight	85	
Yaw weight	0.75	
Airspeed weight	0.01	Requires low airspeed weight since these data are much more noisy.

B.5 Other Parameters

Parameter	**Value (*units*)**
Transformation from camera frame to body frame	rpy = [-90, 0, -90] deg
Octree	—
Octree life	4.0 s
Filter distance threshold	1.0 m
Filter points required	3
Ground saftey distance	3.0 m
Bearing controller	—
Bang-bang tolerance	0.175 rad
Takeoff	—
Acceleration (x-axis) threshold	45.0 m/s^2
Acceleration max (y-axis) for rejection of takeoff	15 m/s^2
Acceleration max (z-axis) for rejection of takeoff	15 m/s^2
Airspeed minimum for throttle	8.0 m/s
Time to clear launch cable	0.8 s
Obstacle Avoidance	—
Distance for trajectory to be considred safe	5.0 m
Required improvement to switch trajectories	0.1 m

B.6 Diagrams

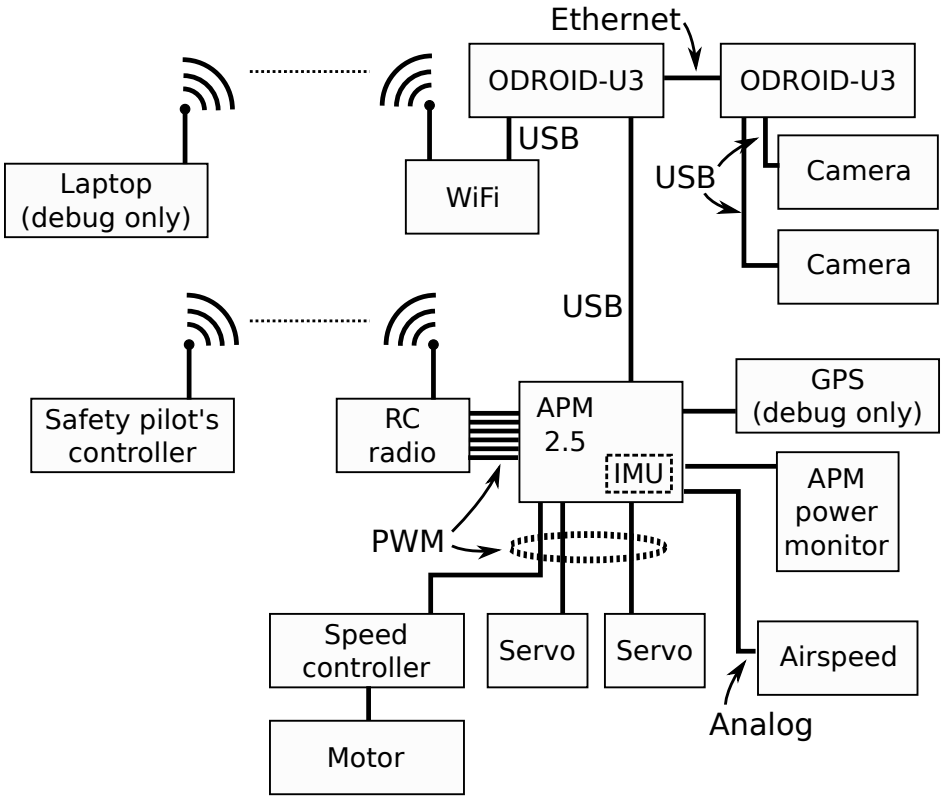


Figure B-1: Information flow diagram for the aircraft.

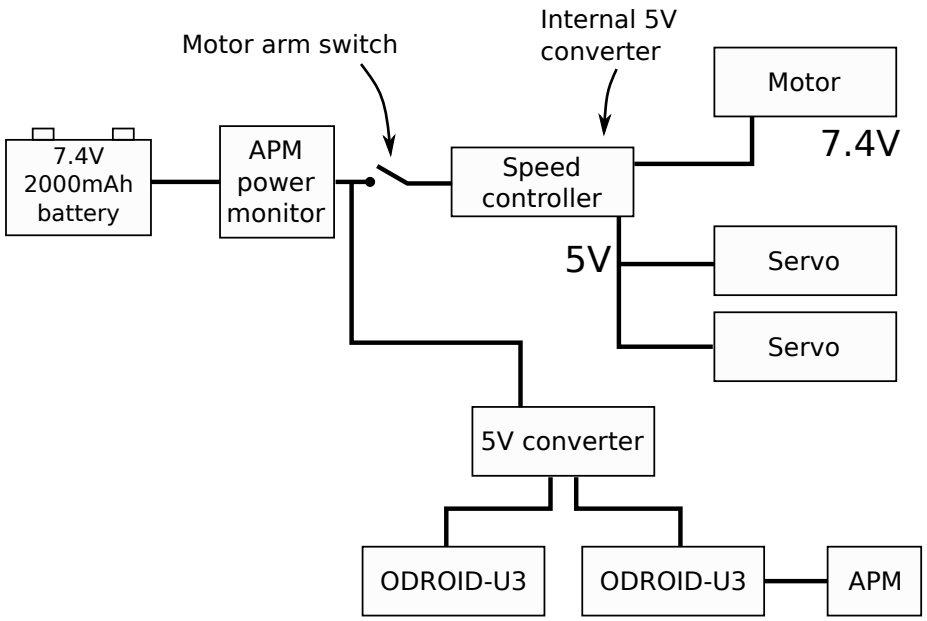


Figure B-2: Power distribution system for the aircraft. Note that the computers operate on an isolated 5V bus so they are insulated from the noise generated by the motor and servos.

Appendix C

Components

Summary	Description / Part Number	Supplier	Quantity	Price	Total
Mechanical					
TBS Caipirinha	-	Team Black Sheep	1	$99.95	$99.95
Winglets	-	Team Black Sheep	2	$0.00	$0.00
Motor / speed controller	”Servos, Motor, ESC and Prop”	Team Black Sheep	1	$129.95	$129.95
Motor mount	-	Team Black Sheep	1	$0.00	$0.00
Graupner E-Prop 8x5	-	Team Black Sheep	1	$6.45	$6.45
Hyperion Atlas Servos	DS 09 GMD	RC Super Sales	2	$29.65	$59.30
Servo horns	-	RC Super Sales	2	$0.00	$0.00
Foam cover	-	RC Foam	1	$20.00	$20.00
Launcher	Jetapult Complete System	FanJets	1	$76.98	$76.98
Launcher hook	-	FanJets	1	$0.00	$0.00
Launcher hook mount	-	FanJets	1	$0.00	$0.00
Teflon tubing, ”0.040”” Teflon Sheathing (yellow)	4 ft.”	CST Sales	1	$1.65	$1.65
Thread	Natural Cotton Thread 273 Yards-Red	Amazon	1	$3.77	$3.77
Numbers decals	mambiSTICKS Themed Stickers, Upper Case Alphabet and Numbers, Black	Amazon	1	$6.92	$6.92
Tiny zip ties	298-1037-ND	Digikey	1	$17.10	$17.10
CA glue	Super-Gold Odorless .5oz CA Adhesive Glue	Amazon	1	$10.49	$10.49

Computation					
ODROID-U3	-	Hardkernel	2	$74.95	$149.90
64GB hard drive	64GB eMMC Module U Linux	Hardkernel	2	$79.95	$159.90
U3 fans	Cooling Fan	Hardkernel	2	$4.95	$9.90
USB WiFi adapter	WiFi Module 2	Hardkernel	1	$8.00	$8.00
WiFi antenna	New A Pair of Laptop Notebook Wifi Bluetooth WWAN Broadband Internal Antenna	Amazon	1	$9.99	$9.99
Angle USB cables	StarTech.com 1 ft Micro USB Cable - A to Left Angle Micro B	Amazon	1	$4.89	$4.89
DC connectors for ODROIDs	CP-012-ND	Digikey	2	$1.12	$2.24
USB A jacks	AE10637-ND	Digikey	3	$0.45	$1.34
USB mini jacks	H2958-ND	Digikey	2	$0.89	$1.78
Sensors and embedded					
3DR APM 2.5	-	3D Robotics	1	$45.85	$45.85
Pitot tube	Airspeed Kit with MPXV7002DP	3D Robotics	1	$0.00	$0.00
Pitot tube sensor	-	3D Robotics	1	$74.85	$74.85
Battery monitor	APM Power Module	3D Robotics	1	$24.99	$24.99
GPS	3DR uBlox GPS	3D Robotics	1	$79.99	$79.99
Spektrum AR6115E RX	-	HorizonHobby	1	$49.99	$49.99
Electrical					
BEC	RMRC 5A Power Regulator - 5 to 6 VOLT (UBEC) RMRC5AUBEC	ReadyMadeRC	1	$9.99	$9.99
Lost model beeper	HobbyKing Discovery Buzzer	Amazon	1	$6.99	$6.99
Micro deans connectors	WS Deans Micro 2R, Red Polarized WSD1222	Amazon	1	$3.95	$3.95
RJ 45 connectors	100 pcs Cat6, Cat5E Crimp Connectors	Amazon	1	$5.54	$5.54
JST connectors	NEEWER Battery Plug JST Connector 10 Pairs	Amazon	1	$4.05	$4.05
Batteries	Turnigy nano-tech 2000mAh 2S1P 20 40C Lipo Receiver Pack	Hobbyking	5	$12.99	$64.95
Cameras					
Cameras	FMVU-03MTC-CS	Point Grey	2	$275.00	$550.00

M12 lens adapters	ACC-01-5000	Point Grey	2	$4.00	$8.00
3.6 mm lenses	”3.6mm 92 Degree Wide Angle CCTV Camera IR Board Lens Focal for 1/3”” CCD”	Amazon	2	$6.90	$13.80
Total					**$1,723.44**

Bibliography

[1] Pieter Abbeel. "An Application of Reinforcement Learning to Aerobatic Helicopter Flight". In: *Proceedings of the Neural Information Processing Systems (NIPS '07)*. Vol. 19. 2006.

[2] Christopher G Atkeson. "Using Local Trajectory Optimizers to Speed Up Global Optimization in Dynamic Programming". In: *Advances in neural information processing systems* (1994), pp. 663–663.

[3] Christopher G Atkeson and Jun Morimoto. "Nonparametric representation of policies and value functions: A trajectory-based approach". In: *Advances in Neural Information Processing Systems* 15 (2003), pp. 1611–1618.

[4] Abraham Galton Bachrach. "Trajectory Bundle Estimation for Perception-Driven Planning". PhD thesis. Massachusetts Institute of Technology, 2013.

[5] D Blake Barber. "Autonomous landing of miniature aerial vehicles". In: *Journal of Aerospace Computing, Information, and Communication* 4.5 (2007), pp. 770–784.

[6] Andrew J. Barry. "Flying Between Obstacles with an Autonomous Knife-Edge Maneuver". MA thesis. Massachusetts Institute of Technology, 2012.

[7] Andrew J. Barry. "Flying Between Obstacles with an Autonomous Knife-Edge Maneuver". In: *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA), Video Track*. 2014.

[8] Andrew J. Barry, Anirudha Majumdar, and Russ Tedrake. "Safety Verification of Reactive Controllers for UAV Flight in Cluttered Environments using Barrier Certificates". In: *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*. 2012.

[9] Declan Bates and Ian Postlethwaite. *Robust multivariable control of aerospace systems*. Delft University Press, 2002. ISBN: 90-407-2317-6.

[10] Alberto Bemporad and Manfred Morari. "Robust model predictive control: A survey". In: *Robustness in identification and control*. Ed. by A. Garulli and A. Tesi. Vol. 245. Lecture Notes in Control and Information Sciences. 10.1007/BFb0109870. Springer Berlin / Heidelberg, 1999, pp. 207–226.

[11] Antoine Beyeler, Jean-Christophe Zufferey, and Dario Floreano. "optiPilot: control of take-off and landing using optic flow". In: *Proceedings of the 2009 European Micro Air Vehicle conference and competition (EMAV '09)*. 2009.

[12] Antoine Beyeler, Jean-Christophe Zufferey, and Dario Floreano. "Vision-based control of near-obstacle flight". In: *Autonomous robots* 27.3 (2009), pp. 201–219.

[13] H.G. Bock and K.J. Plitt. "A multiple shooting algorithm for direct solution of optimal control problems". In: *Proceedings 9th IFAC World Congress Budapest*. Pergamon Press. 1984, pp. 243–247.

[14] Patrick Bouffard, Anil Aswani, and Claire Tomlin. "Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results". In: *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 279–284.

[15] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[16] Adam Bry, Abraham Bachrach, and Nicholas Roy. "State estimation for aggressive flight in gps-denied environments using onboard sensing". In: *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 1–8.

[17] Jeffrey Byrne, Martin Cosgrove, and Raman Mehra. "Stereo based obstacle detection for an unmanned air vehicle". In: *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2006, pp. 2830–2835.

[18] Anna C. Carruthers, Adrian L.R. Thomas, and Graham K. Taylor. "Automatic aeroelastic devices in the wings of a steppe eagle Aquila nipalensis". In: *Journal of Experimental Biology* 210.23 (2007), pp. 4136–4149.

[19] Anna C Carruthers. "Use and Function of a Leading Edge Flap on the Wings of Eagles". In: *AIAA Paper* 43 (2007), p. 2007.

[20] Girish Chowdhary. "GPS-denied Indoor and Outdoor Monocular Vision Aided Navigation and Control of Unmanned Aircraft". In: *Journal of Field Robotics* (2013).

[21] Rick Cory. "Supermaneuverable Perching". PhD thesis. Massachusetts Institute of Technology, 2010.

[22] Rick Cory and Russ Tedrake. "Experiments in Fixed-Wing UAV Perching". In: *Proceedings of the AIAA Guidance, Navigation, and Control Conference*. AIAA. 2008, pp. 1–12.

[23] M. Cutler. "Comparison of Fixed and Variable Pitch Actuators for Agile Quadrotors". In: *AIAA Guidance, Navigation, and Control Conference (GNC)*. (AIAA-2011-6406). Portland, OR, 2011.

[24] Andrew J Davison. "MonoSLAM: Real-time single camera SLAM". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29.6 (2007), pp. 1052–1067.

[25] C De Wagter. "Autonomous flight of a 20-gram flapping wing mav with a 4-gram onboard stereo vision system". In: *Proceedings of the 2014 IEEE/RSJ Int. Conf. on Robotics and Autonomous Systems (ICRA)*. Hong Kong, China, 2014.

[26] Debadeepta Dey. "Vision and Learning for Deliberative Monocular Cluttered Flight". In: *Field and Service Robotics (FSR)*. 2015.

[27] Moritz Diehl. "Fast direct multiple shooting algorithms for optimal robot control". In: *Fast motions in biomechanics and robotics*. Springer, 2006, pp. 65–93.

[28] M. Drela and H. Youngren. "XFOIL 6.94 User Guide". In: (2001).

[29] Jakob Engel, Jurgen Sturm, and Daniel Cremers. "Camera-based navigation of a low-cost quadrocopter". In: *Proceedings of the International Conference on Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ*. IEEE. 2012, pp. 2815–2821.

[30] Dario Floreano. "Aerial Locomotion in Cluttered Environments". In: *Proceedings of the 15th International Symposium on Robotics Research*. 2011.

[31] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. "SVO: Fast Semi-Direct Monocular Visual Odometry". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2014, pp. 15–22.

[32] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. "Robust hybrid control for autonomous vehicle motion planning". In: *Proceedings of the 39th IEEE Conference on Decision and Control, 2000*. Vol. 1. IEEE. 2000, pp. 821–826.

[33] Carlos E Garcia, David M Prett, and Manfred Morari. "Model predictive control: theory and practicea survey". In: *Automatica* 25.3 (1989), pp. 335–348.

[34] Vladislav Gavrilets. "Control Logic for Automated Aerobatic Flight of a Miniature Helicopter". In: *AIAA Guidance, Navigation and Control Conference*. 2002.

[35] Steven B Goldberg and Larry Matthies. "Stereo and IMU Assisted Visual Odometry on an OMAP3530 for Small Robots". In: *Proceedings of the 2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE. 2011, pp. 169–176.

[36] Rajiv Gupta and Richard I Hartley. "Linear pushbroom cameras". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.9 (1997), pp. 963–975.

[37] Ankur Handa. "Real-Time camera tracking: when is high frame-rate best?" In: *Computer Vision–ECCV 2012*. Springer, 2012, pp. 222–235.

[38] Christopher G Harris and JM Pike. "3D positional integration from image sequences". In: *Image and Vision Computing* 6.2 (1988), pp. 87–90.

[39] Markus Hehn and Raffaello D'Andrea. "A flying inverted pendulum". In: *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 763–770.

[40] Heiko Hirschmuller. "Accurate and efficient stereo processing by semi-global matching and mutual information". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2. IEEE. 2005, pp. 807–814.

[41] Dominik Honegger, Helen Oleynikova, and Marc Pollefeys. "Real-time and Low Latency Embedded Computer Vision Hardware Based on a Combination of FPGA and Mobile CPU". In: *International Conference on Intelligent Robots and Systems*. IEEE/RSJ. Chicago, Illinois, USA, 2014.

[42] Dominik Honegger. "Real-time velocity estimation based on optical flow and disparity matching". In: *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2012, pp. 5177–5182.

[43] Stefan Hrabar. "Combined optic-flow and stereo-based navigation of urban canyons for a UAV". In: *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2005, pp. 3309–3316.

[44] Albert S. Huang, Edwin Olson, and David C. Moore. "LCM: Lightweight Communications and Marshalling". In: *International Conference on Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ* (2010), pp. 4057–4062.

[45] Kenneth W Iliff. "Parameter Estimation for Flight Vehicles". In: *Journal of Guidance, Control, and Dynamics* 12.5 (1989), pp. 609–622.

[46] Andrew Johnson, James Montgomery, and Larry Matthies. "Vision guided landing of an autonomous helicopter in hazardous terrain". In: *Proceedings of the 2005 IEEE International Conference on Robotics and automation, 2005. ICRA 2005*. IEEE. 2005, pp. 3966–3971.

[47] Takeo Kanade, Omead Amidi, and Qifa Ke. "Real-time and 3D vision for autonomous small and micro air vehicles". In: *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC)*. Vol. 2. IEEE. 2004, pp. 1655–1662.

[48] Sertac Karaman and Emilio Frazzoli. "Incremental Sampling-based Optimal Motion Planning". In: *Robotics: Science and Systems* (2010).

[49] L.E. Kavraki. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[50] Jong-Hyuk Kim and Salah Sukkarieh. "Airborne simultaneous localisation and map building". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 1. IEEE. 2003, pp. 406–411.

[51] Georg Klein and David Murray. "Parallel tracking and mapping for small AR workspaces". In: *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2007, pp. 225–234.

[52] Aleksandr Kushleyev, Vijay Kumar, and Daniel Mellinger. "Towards A Swarm of Agile Micro Quadrotors." In: *Robotics: Science and Systems*. 2012.

[53] Jack Langelaan and Steve Rock. "Towards autonomous uav flight in forests". In: *Proc. of AIAA Guidance, Navigation and Control Conference*. 2005.

[54] Jack W Langelaan. "State estimation for autonomous flight in cluttered environments". In: *Journal of Guidance, Control, and Dynamics* 30.5 (2007), pp. 1414–1426.

[55] Jacob Willem Langelaan. "State Estimation for Autonomous Flight in Cluttered Environments". PhD thesis. Stanford University, 2006.

[56] S. LaValle. *Rapidly-exploring random trees: A new tool for path planning*. Tech. rep. 98–11. Iowa State University, Dept. of Computer Science, 1998.

[57] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[58] Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. "Mav visual slam with plane constraint". In: *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 3139–3144.

[59] FL Lewis and VL Syrmos. *Optimal control*. Second. John Wiley & Sons, 1995. ISBN: 0-471-03378-2.

[60] M. Li and A.I. Mourikis. "3-D Motion Estimation and Online Temporal Calibration for Camera-IMU Systems". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Karlsruhe, Germany, 2013.

[61] M. Li, B. Kim, and A.I. Mourikis. "Real-time Motion Tracking on a Cellphone using Inertial Sensing and a Rolling Shutter Camera". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Karlsruhe, Germany, 2013.

[62] Huai-Ti Lin, Ivo G Ros, and Andrew A Biewener. "Through the eyes of a bird: modelling visually guided obstacle flight". In: *Journal of The Royal Society Interface* 11.96 (2014).

[63] Quentin Lindsey, Daniel Mellinger, and Vijay Kumar. "Construction with quadrotor teams". In: *Autonomous Robots* 33.3 (2012), pp. 323–336.

[64] L. Ljung. "System Identification Toolbox for Use with MATLAB". In: (2007).

[65] Sergei Lupashin. "A simple learning strategy for high-speed quadrocopter multi-flips". In: *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 1642–1648.

[66] Anirudha Majumdar, Amir Ali Ahmadi, and Russ Tedrake. "Control and Verification of High-Dimensional Systems with DSOS and SDSOS Programming". In: *Proceedings of the 53rd Conference on Decision and Control (CDC)*. 2014.

[67] Anirudha Majumdar and Russ Tedrake. "Funnel Libraries for Robust Realtime Feedback Motion Planning". In: *In preparation* (2016).

[68] Anirudha Majumdar and Russ Tedrake. "Robust Online Motion Planning with Regions of Finite Time Invariance". In: *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*. Cambridge, MA, 2012, p. 16.

[69] Lorenz Meier. "PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision". In: *Autonomous Robots* 33.1-2 (2012), pp. 21–39.

[70] D. Mellinger and V. Kumar. "Minimum snap trajectory generation and control for quadrotors". In: *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 2520–2525.

[71] D. Mellinger, N. Michael, and V. Kumar. "Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors". In: *Proceedings of the 12th International Symposium on Experimental Robotics (ISER 2010)*. 2010.

[72] Daniel Mellinger. "Cooperative grasping and transport using multiple quadrotors". In: *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*. 2010.

[73] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. "High speed obstacle avoidance using monocular vision and reinforcement learning". In: *Proceedings of the 22nd International Conference on Machine Learning*. ACM. 2005, pp. 593–600.

[74] Joseph Moore. "Robust Post-Stall Perching with a Fixed-Wing UAV". PhD thesis. Massachusetts Institute of Technology, 2014.

[75] Joseph Moore and Russ Tedrake. "Control Synthesis and Verification for a Perching UAV using LQR-Trees". In: *Proceedings of the IEEE Conference on Decision and Control*. Maui, Hawaii, 2012, p. 8.

[76] Joseph Moore and Russ Tedrake. "Magnetic Localization for Perching UAVs on Powerlines". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2011).

[77] Joseph Moore, Rick Cory, and Russ Tedrake. "Robust Post-Stall Perching with a Simple Fixed-Wing Glider using LQR-Trees". In: *Bioinspiration and Biomimetics* 9.2 (2014), p. 15.

[78] M. Muller, S. Lupashin, and R. D'Andrea. "Quadrocopter ball juggling". In: *Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2011, pp. 5113–5120.

[79] Vidya N Murali and Stanley T Birchfield. "Autonomous exploration using rapid perception of low-resolution image information". In: *Autonomous Robots* 32.2 (2012), pp. 115–128.

[80] Ashley Napier, Peter Corke, and Paul Newman. "Cross-calibration of push-broom 2D LIDARs and cameras in natural scenes". In: *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013, pp. 3679–3684.

[81] Thomas Netter and Nicholas Francheschini. "A robotic aircraft that follows terrain using a neuromorphic eye". In: *Intelligent Robots and Systems, 2002.* Vol. 1. IEEE. 2002, pp. 129–134.

[82] David Nistér, Oleg Naroditsky, and James Bergen. "Visual Odometry". In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004*. IEEE. 2004, pp. 652–659.

[83] Helen Oleynikova, Dominik Honegger, and Marc Pollefeys. "Reactive Avoidance Using Embedded Stereo Vision for MAV Flight". In: *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 50–56.

[84] Liam O'Sullivan, Peter Corke, and Robert Mahoney. "Image-based visual navigation for mobile robots". In: *Proceedings of the 2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013.

[85] Nathan Pilbrough. *High Speed Reactive Collision Avoidance on an Unmanned Surface Vehicle*. Honours Thesis, Department of Electrical Engineering, University of Cape Town. 2015.

[86] 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)*. Singapore, 2013.

[87] Robin Ritz. "Cooperative quadrocopter ball throwing and catching". In: *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2012, pp. 4972–4978.

[88] Jonathan M Roberts, Peter I Corke, and Gregg Buskey. "Low-cost flight control system for a small autonomous helicopter". In: *Proceedings of the of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 1. IEEE. 2003, pp. 546–551.

[89] Richard Roberts. "Saliency detection and model-based tracking: a two part vision system for small robot navigation in forested environment". In: *Proceedings of SPIE*. Vol. 8387. 2012.

[90] I Michael Ross. *A primer on pontryagin's principle in optimal control*. Collegiate Publishers, 2009.

[91] Stéphane Ross. "Learning Monocular Reactive UAV Control in Cluttered Natural Environments". In: *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013, pp. 1765–1772.

[92] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011.

[93] Inkyu Sa. "Monocular Vision Based Autonomous Navigation for a Cost-Effective MAV in GPS-Denied Environments". In: *Proceedings of the 2013 IEEE/ASME In-*

ternational Conference on Advanced Intelligent Mechatronics (AIM). IEEE. 2013, pp. 1355–1360.

[94] Srikanth Saripalli, James F Montgomery, and Gaurav S Sukhatme. "Vision-based autonomous landing of an unmanned aerial vehicle". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 3. IEEE. 2002, pp. 2799–2804.

[95] Davide Scaramuzza and Friedrich Fraundorfer. "Visual odometry [tutorial]". In: *Robotics & Automation Magazine, IEEE* 18.4 (2011), pp. 80–92.

[96] Sebastian Scherer. "Flying Fast and Low Among Obstacles". In: *Proceedings of the 2007 IEEE International Conference on Robotics and Automation*. IEEE. 2007, pp. 2023–2029.

[97] Jeff S Shamma and James R Cloutier. "Gain-scheduled missile autopilot design using linear parameter varying transformations". In: *Journal of Guidance, Control, and Dynamics* 16.2 (1993), pp. 256–263.

[98] S. Shen. "Vision-Based State Estimation and Trajectory Control Towards Aggressive Flight with a Quadrotor". In: *Robotics: Science and Systems*. Berlin, Germany, 2013.

[99] Shaojie Shen. "Vision-Based State Estimation for Autonomous Rotorcraft MAVs in Complex Environments". In: *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2013.

[100] Robert Sim. "Vision-based SLAM using the Rao-Blackwellised particle filter". In: *IJCAI Workshop on Reasoning with Uncertainty in Robotics*. Vol. 14. 2005, pp. 9–16.

[101] Leena Singh and James Fuller. "Trajectory generation for a UAV in urban terrain, using nonlinear MPC". In: *Proceedings of the 2001 American Control Conference*. Vol. 3. IEEE. 2001, pp. 2301–2308.

[102] Hebertt Sira-Ramírez and Sunil K. Agrawal. *Differentially Flat Systems*. Control Engineering. Marcel Dekker, 2004. ISBN: 0-8247-5470-0.

[103] Frantisek Michal Sobolic. "Agile Flight Control Techniques for a Fixed-Wing Aircraft". MA thesis. Cambridge MA: Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 2009.

[104] MV Srinivasan. "An overview of insect-inspired guidance for application in ground and airborne platforms". In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 218.6 (2004), pp. 375–388.

[105] Natesh Srinivasan, Richard Roberts, and Frank Dellaert. "High Frame Rate Egomotion Estimation". In: *Computer Vision Systems*. Springer, 2013, pp. 183–192.

[106] M. Stolle and C.G. Atkeson. "Policies based on trajectory libraries". In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. IEEE. 2006.

[107] Oskar von Stryk. "Numerical solution of optimal control problems by direct collocation". In: *Optimal Control, (International Series in Numerical Mathematics 111)*. 1993, pp. 129–143.

[108] Petr Švestka and Mark H. Overmars. "Motion planning for car-like robots using a probabilistic learning approach". In: *The International Journal of Robotics Research* 16.2 (1997), pp. 119–143.

[109] Graham K. Taylor. "Flight Control Mechanisms in Birds of Prey". In: *45th AIAA Aerospace Sciences Meeting and Exhibit*. Ed. by AIAA. AIAA. 2007.

[110] Russ Tedrake. *Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems*. `http://drake.mit.edu`. 2014.

[111] Russ Tedrake. "LQR-Trees: Feedback motion planning on sparse randomized trees". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2009, p. 8.

[112] Mark M. Tobenkin. "Robustness Analysis for Identification and Control of Nonlinear Systems". PhD thesis. Massachusetts Institute of Technology, 2014.

[113] Yang Wang and Stephen P Boyd. "Fast Model Predictive Control Using Online Optimization". In: *International Federation of Automatic Control World Congress*. Vol. 17. 2008, pp. 6974–6979.

[114] Stephan Weiss. “Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments”. In: *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 957–964.

[115] Ruigang Yang and Marc Pollefeys. “Multi-resolution real-time stereo on commodity graphics hardware”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 1. IEEE. 2003.

[116] Jean-Christophe Zufferey, Antoine Beyeler, and Dario Floreano. “Near-obstacle flight with small UAVs”. In: *Proceedings of the International Symposium on Unmanned Aerial Vehicles*. Orlando, FL, 2008.