High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo

Andrew J. Barry

Computer Science and Artificial Intelligence Lab

Massachusetts Institute of Technology

Cambridge, MA, USA

abarry@csail.mit.edu

Peter R. Florence

Computer Science and Artificial Intelligence Lab

Massachusetts Institute of Technology

Cambridge, MA, USA

peteflo@csail.mit.edu

Russ Tedrake

Computer Science and Artificial Intelligence Lab

Massachusetts Institute of Technology

Cambridge, MA, USA

russt@csail.mit.edu

Abstract

We present the design and implementation of a small autonomous unmanned aerial vehicle capable of high-speed flight through complex natural environments. Using only onboard GPS-denied sensing and computation, we perform obstacle detection, planning, and feedback control in realtime. We present a novel integrated approach to perception and control using pushbroom stereo, which exploits forward motion to enable efficient obstacle detection and avoidance using lightweight processors on a UAV. 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 detecting obstacles at 120 Hz and 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 using 3D geometric information.

1 Introduction

Recent advances in lightweight processing have enabled sophisticated autonomy and computer vision tasks onboard small flying vehicles. Despite the substantial increase in available computation, however, perceiving and avoiding obstacles during high-speed flight remains a significant challenge. Small vehicles struggle to lift sophisticated laser rangers and often computer vision systems are not fast enough to avert collision.

In this paper, we present a vision and control system that is capable of high-speed flight (10-14

(a) (b)

Figure 1: Autonomous aircraft avoiding a tree (a) and an onboard view (b), where detected obstacles are marked in red.

m/s or 22-31 MPH) in natural environments. 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. We have previously presented the method of generating perception data using pushbroom stereo (Barry and Tedrake, 2015), but this paper presents the first published demonstration of using pushbroom stereo in an aerial autonomous system, integrated with planning and control for obstacle avoidance.

1.1 Organization

In Section 2 we discuss related work followed by Sections 3–4 which introduce and evaluate our stereo vision algorithm. Sections 5–7 present the autonomous system and results. Finally, we offer concluding remarks in Section 8. We note that Section 3 was previously presented in (Barry and Tedrake, 2015) and the combined, autonomous system is presented in the author's doctoral thesis (Barry, 2016).

2 Related Work

2.1 Obstacle Avoidance with Micro Aerial Vehicles

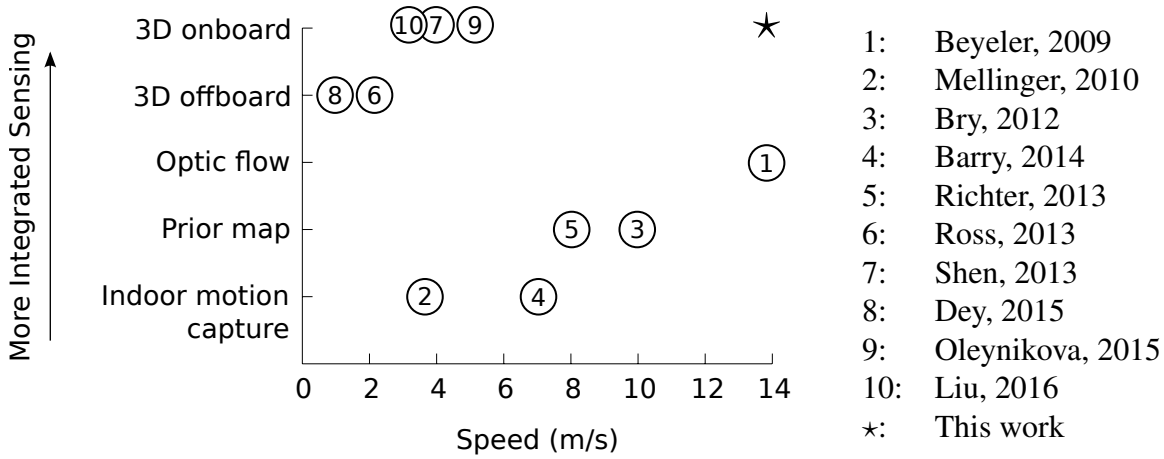


Figure 2: Plot of related work in small UAV obstacle avoidance examining speed and sensor integration. The $\star$ indicates this work and labels are in order of publication year. Despite varying flight hardware, we believe speed is a good comparison metric because the systems are limited by sensing and computation as much as wingspan or body-length.

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.

Effectively avoiding obstacles, however, is an area of growing interest. Recently, DARPA funded a program focused on high speed flight near obstacles (DARPA, 2014). Consumer products are now available with some autonomous obstacle avoidance capabilities[2]. Figure 2 maps some of the

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

[2] DJI Phantom 4 Pro (`http://www.dji.com/phantom-4-pro`)

related work in the field based on sensor integration and speed.

2.1.1 Agile Flight In Motion Capture Environments

Even omitting obstacle detection, agile flight is a required component of obstacle avoidance. Developments of agile flight capabilities have largely taken place in motion capture environments, where rotorcraft and fixed wings have been demonstrated with accurate dynamics models and good trajectory tracking. Abbeel's work on learning autonomous helicopter maneuvers (Abbeel et al., 2006) was an early demonstration of these techniques. Mellinger and Kumar demonstrate quadrotor flight through small openings (Mellinger et al., 2010a) and in formation (Mellinger et al., 2010b, Kushleyev et al., 2012). Quadrotors have further been shown to perform agressive flips (Lupashin et al., 2010) and interact with objects including catching and juggling a ball (Bouffard et al., 2012, Muller et al., 2011), balancing a pendulum (Hehn and D'Andrea, 2011), cooperatively throwing a ball (Ritz et al., 2012), and building structures (Lindsey et al., 2012).

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 (Sobolic, 2009) and Cory demonstrated autonomous perching on a glider, landing on a wire with impressive 5cm accuracy (Cory and Tedrake, 2008). We have previously demonstrated flight through a gap smaller than the aircraft's wingspan (Barry et al., 2014).

2.1.2 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 (Bry et al., 2012, Richter et al., 2013). Their work relies on a laser rangefinder to localize the aircraft in flight, allowing it to use a prior map of the environment. Should obstacles appear between their offline mapping and flight phases, the aircraft may collide.

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[3], currently do not work outdoors, limiting their usefulness to indoor navigation tasks. For these reasons, the field has focused on lightweight cameras and computer vision algorithms.

2.2.1 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 (Byrne et al., 2006). 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 sys-

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

tem at only 5-10 Hz, 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 (Hrabar et al., 2005). Researchers have also implemented stereo on GPU for significant processing gains, and have even suggested a method that eliminates the need for per-frame rectification, which further increases efficiency and framerate (Yang and Pollefeys, 2003).

Meier et al. integrates stereo vision based obstacle detection with a full suite of IMU and vision-based state estimation, and autonomous waypoint navigation (Meier et al., 2012). 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 paper, but is still limited to 46 fps (Goldberg and Matthies, 2011). 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 (De Wagter et al., 2014). 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) (Honegger et al., 2012, Honegger et al., 2014) with recent work demonstrating onboard obstacle avoidance at 5 m/s (Oleynikova et al., 2015).

2.2.2 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 (Michels et al., 2005). 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 (Dey et al., 2015).

2.2.3 Optic Flow

Optical flow techniques work well, controlling stable flight, takeoff, landing (Barber et al., 2007, Beyeler et al., 2009a), and obstacle avoidance (Beyeler et al., 2009b, Zufferey et al., 2008). 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[4]. 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.

2.2.4 Visual Simultaneous Localization and Mapping (VSLAM)

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 (Davison et al., 2007, Harris and Pike, 1988, Sim et al., 2005) on MAVs (Kim and Sukkarieh, 2003). There are a few fundamental problems with VSLAM on MAVs, notably: (1) computational power and (2) robustness of the

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

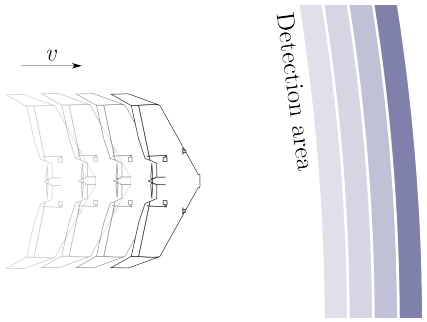


Figure 3: By detecting at a single depth (dark blue) and integrating the aircraft's odometry and past detections (lighter blue), we can quickly build a full map of obstacles in front of our vehicle.

navigation solution.

Recently, researchers have begun using parallel tracking and mapping (PTAM) (Klein and Murray, 2007) 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 computationally expensive.

3 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.) Canonical stereo vision techniques are too slow to meet this goal on lightweight processors, so we explore a modified stereo approach.

3.1 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[5] 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.

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.

3.2 Pushbroom Stereo

In contrast to a conventional block-matching stereo vision system, with pushbroom stereo we limit our search through distance to a single value, d meters away. This 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 fast moving platform (in this case, an aircraft), so we can quickly

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

recover the missing depth information by integrating our odometry and previous single-disparity results (Figure 3). 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.

Pushbroom stereo is an approach that both specifies a stereo vision approach (only search for depth returns of a fixed distance) and an approach to motion over time (forward motion in a relatively static environment). 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 (Napier et al., 2013)).

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 (Gupta and Hartley, 1997). These cameras are often found on satellites and can be used for stereo vision (Hirschmuller, 2005). We also note that our approach is distinct from what is sometimes referred to as "plane sweeping" (Collins, 1996, Gallup et al., 2007) in stereo vision, which is another idea that fits into the conventional stereo method of searching over a wide range of depths for a static (no motion), single-time set of images.

3.2.1 Odometry

Successful implementation of pushbroom stereo 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.

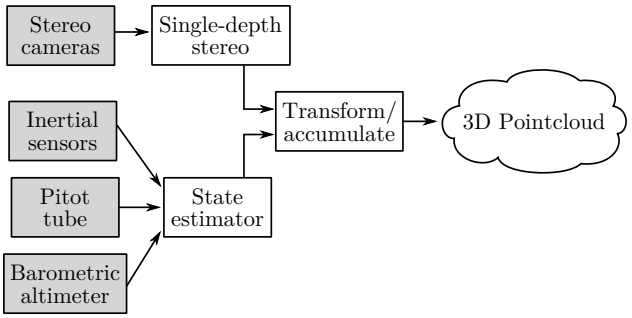


Figure 4: Pushbroom stereo overview. Note that the system does not require GPS.

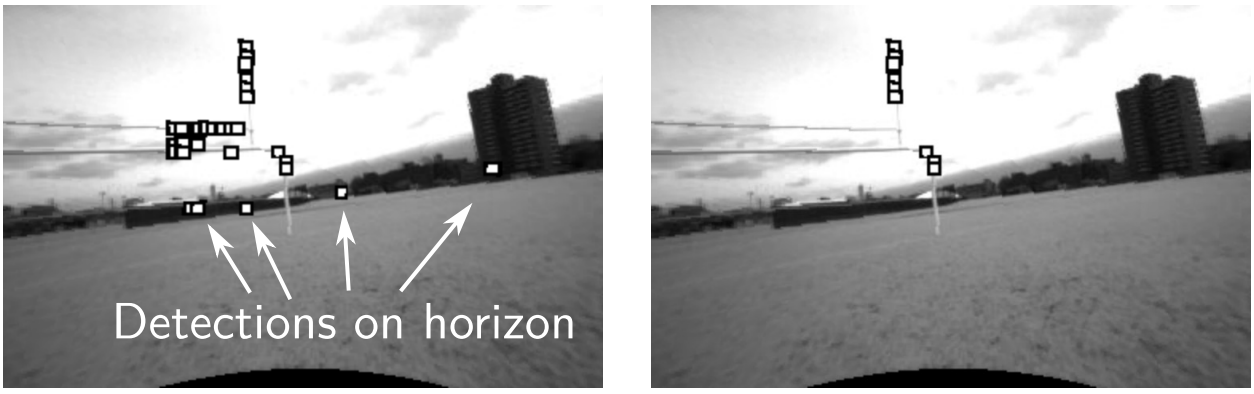
3.3 Implementation

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:

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

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.



(a) Without horizontal invariance filter.

(b) With horizontal invariance filter.

Figure 5: 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.

Since we expect to have many candidate matches on each obstacle, we deliberately chose a design and parameters to cause sparse detections with few false positives. For obstacle avoidance, we do not need to register every point on an object, but instead want to minimize false positives that could cause the aircraft to take unnecessary risks avoiding 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 multiple 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

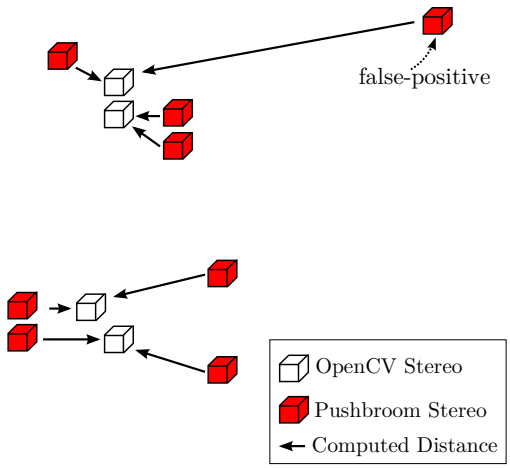


Figure 6: Sketch of our evaluation strategy for single-disparity stereo. We detect false-positives by computing the distance from single-disparity stereo's output (red) to the nearest point from OpenCV's `StereoBM` (white). False positives stand out with large distances (labeled box).

dramatically that we can run this filter online. Figure 5 demonstrates this filter running on flight data.

4 Pushbroom Stereo Evaluation

4.1 Compared to Block-Matching 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 on-board stereo system with the state-estimator disabled[6]. We then, offline, used OpenCV's (Bradski, 2000) 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.

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

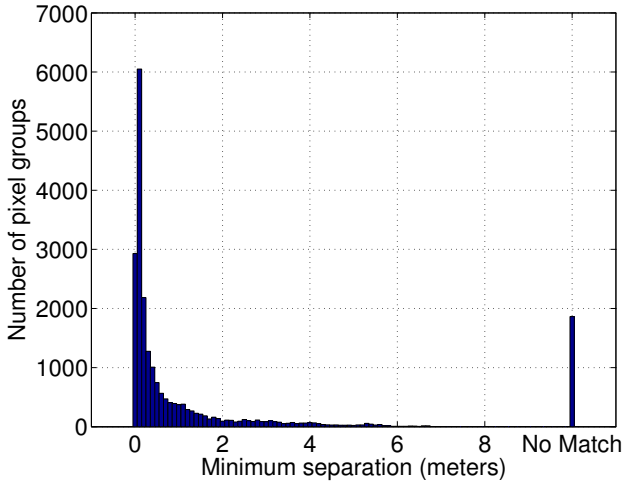


Figure 7: Results of the false-positive benchmark described in Figure 6 on 23,000+ frames. Histogram bin width is 0.1 meters. *No Match* indicates single-disparity points where there was no matching StereoBM point on the frame. We find that 8.2% of detected pixels fall into this category.

With these data, we detected false-positives by computing the Euclidean distance from each single-disparity stereo coordinate to the nearest point produced by the depth-cropped StereoBM (Figure 6). 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 4.2 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 7). For context, the aircraft's wingspan is 0.86 meters and it covers 0.5 meters in 0.03 to 0.07 seconds.

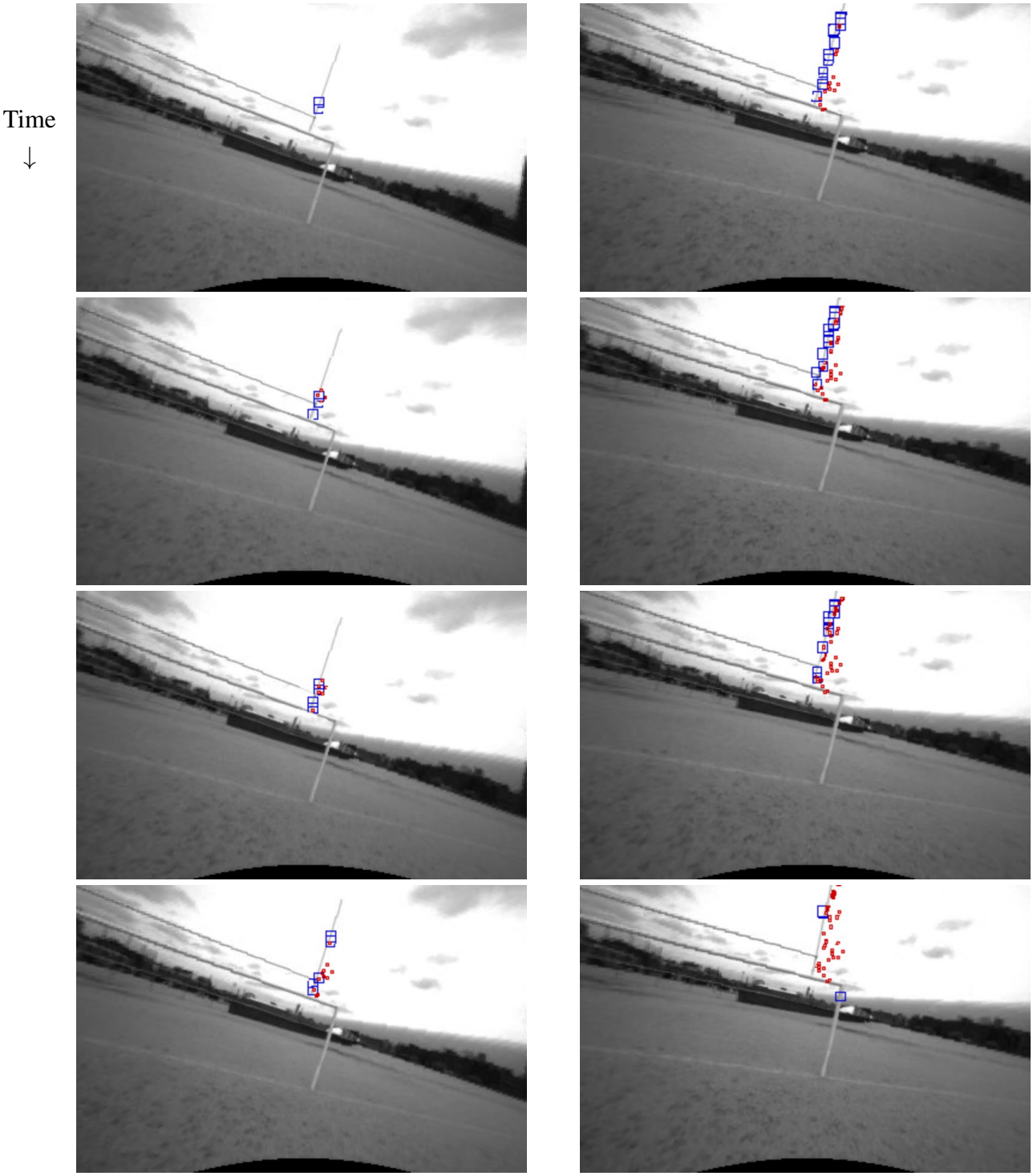


Figure 8: Sequence of stills from an obstacle detection. Each image is 0.02 seconds (20ms) after the previous. The entire set captures 0.16 seconds. Here, the fieldgoal is detected in the first frames (larger blue boxes). Afterwards, the position of those detections is estimated via the state estimator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

4.2 Manual Flight Experiments

To test the full system with an integrated state-estimator, we flew our platform close to obstacles on three different flights, recorded control inputs, sensor data, camera images, and on-board stereo processing results. Figure 8 shows on-board stereo detections as the aircraft approaches an obstacle.

During each flight, we detected points on every obstacle in real time. Our GPS-denied state estimate (see Section 5.2) 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 Section 6.

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 4.1, 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 10a).

Figures 9 and 10b show the results of the false-positive and false-negative benchmarks on all three flights respectively. Our system does not produce many false-positives, with 74.8% points falling

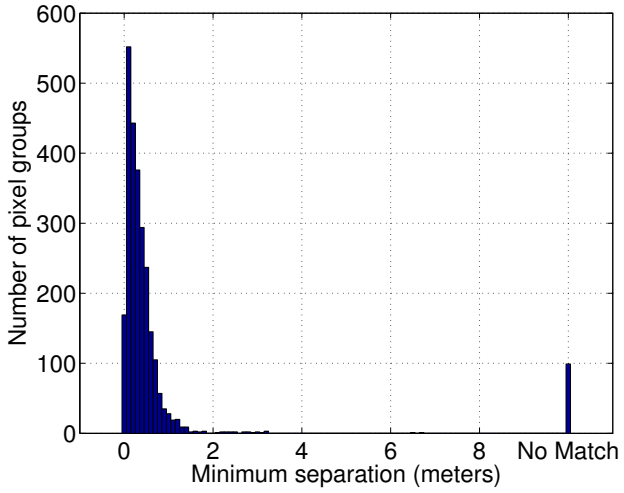
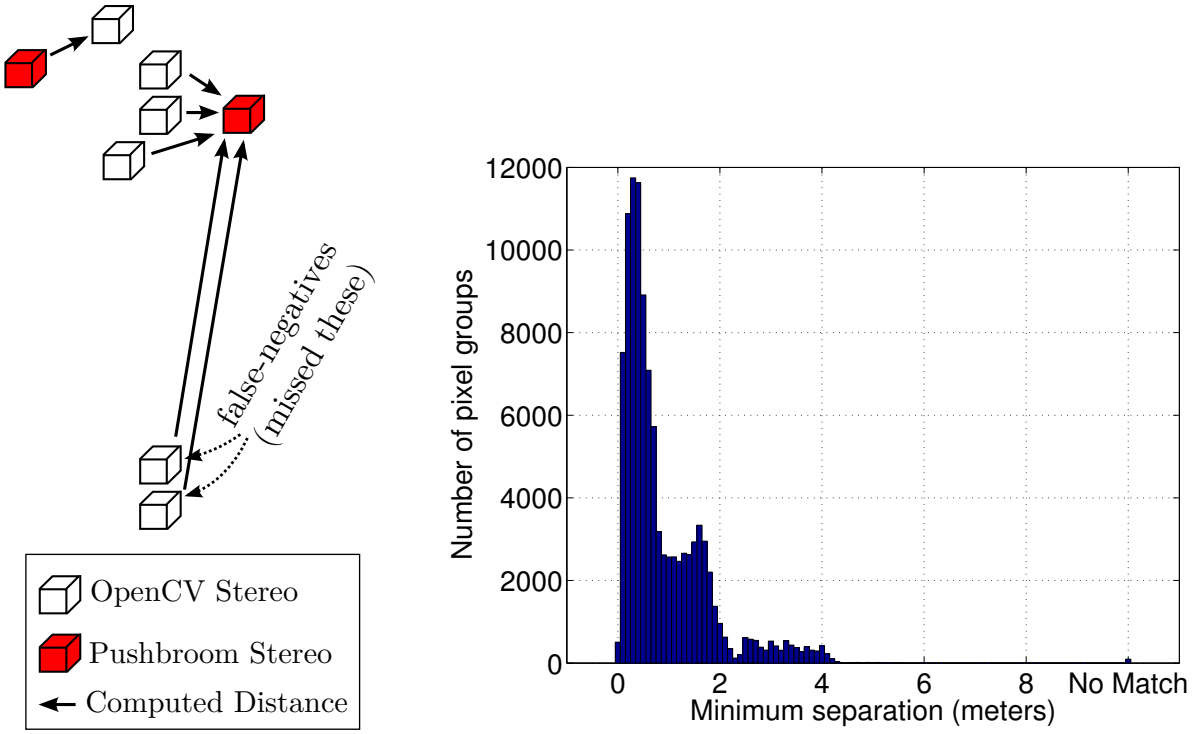


Figure 9: Results of the comparison as described in Figure 6 (a). Our system produces few outliers, with most points falling within the OpenCV StereoBM implementation (74.8% and 92.6% within 0.5 and 1.0 meters respectively), even as we integrate our state estimate, and the obstacle positions, forward. Histogram bin width is 0.1 meters. *No Match* indicates points that pushbroom stereo detected but there were no block-matching stereo detections on that frame.

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. We believe the improved performance on this flight dataset compared to the ground dataset (Section 4.1) is due to the relative complexities of the environments. See (Barry and Tedrake, 2015) for ground data environments.

As Figure 10 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 paper use identical threshold, scoring, and other parameters, despite changing



(a) Comparison technique.

(b) False-negative comparison.

Figure 10: 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. Histogram bin width is 0.1 meters.

the detection distance, lens focal length, and camera calibration between Sections 3 and 6. The complete set of parameters is listed in (Barry, 2016), Appendix B.

5 Autonomous Control

In this section we present the design and implementation of the first fully autonomous aircraft using pushbroom stereo. We focus on a sytem well suited for GPS-denied state estimation and the limited computational payload of a lightweight vehicle. Our solution leverages well-known techniques for state estimation and control, carefully integrated with the perception system. We use

a model-based control system consisting of trajectories selected from a library in realtime based on a pointcloud produced by pushbroom stereo. Each open-loop trajectory has a precomputed linear quadratic regulator (LQR) associated with it for closed-loop control. The control is performed with the full state estimate of the vehicle produced by the onboard GPS-denied state estimator running in the loop.

5.1 Aircraft Model

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

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

where x, y, and z are positions, and ϕ, θ, and ψ are the aircraft's roll, pitch, and yaw respectively. U, V, and W are velocities forward, to the right, and downwards in the body frame, and P, Q, and R are angular rotations about the x, y, and z axes respectively.

We use a flat-plate model inspired by Cory's work on his fixed-wing glider (Cory and Tedrake, 2008). We model the aircraft with three fixed flat plates representing the wing and winglets and two smaller moving flat plates for the elevons. We model the propeller as a thrust-generating element, but ignore its aerodynamic drag, blade flapping, and other higher order effects. Full details are provided in (Barry, 2016), including model parameters in Appendix B.

5.2 State Estimation

This experiment exclusively uses an onboard, GPS-denied system. 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 (Bry et al., 2012)[7]. 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 good estimates for all state variables except for x and y. State estimator parameters used are provided in (Barry, 2016), Appendix B.

5.2.1 Perception in a Local Frame

To enable fast nearest-neighbor searches through a potentially large point cloud, we use an octree structure from (Rusu and Cousins, 2011). 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 every 2 seconds. Due to the sensing horizon (10 meters) and the aircraft's fast forward speed (12-14 m/s), a memory of this length of time is sufficient.

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

5.3 Trajectory Libraries

We use a trajectory library approach, which is a common method for online planing. For example, Frazzoli et. al give a detailed description of a library-based control system for a helicopter, showing that it can be computationally efficient and stable (Frazzoli et al., 2000).

Our system relies on trim and maneuver trajectories computed offline and then selects a trajectory to execute online, similar to (Frazzoli et al., 2000). We generate trim (zero-acceleration) conditions using our model. To generate high acceleration maneuvers, we use a record-from-data approach, where we record state and control information from aggressive manual flight.

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 (Majumdar and Tedrake, 2012). In recent work, Majumdar has shown a full working system that uses these guarantees to aggressively avoid obstacles (Majumdar and Tedrake, 2017).

5.4 Feedback Control

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 use a standard, time-invariant LQR controller. To provide feedback control for trajectories recorded from data, we use a time-varying linear quadratic regulator (TVLQR) as in (Tedrake, 2009). We take advantage of the fact that the aircraft's dynamics are invariant to x, y, z, and ψ (yaw) to substantially reduce the number of trajectories required in the library. Thus, when a trajectory is selected online, we

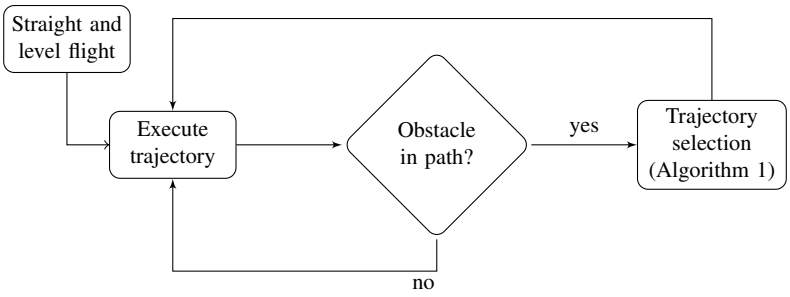


Figure 11: Diagram of the online planning system. An obstacle is deemed to be in the path if the minimum distance between the trajectory and the point cloud is below a threshold.

record those four initial states and transform the trajectory along those coordinates.

5.5 Online Planning

Given a trajectory library and a point cloud from pushbroom stereo, 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 minimum distance). Figure 11 outlines the system and Algorithm 1 shows the trajectory selection algorithm. The navigation direction is implicitly chosen by the ordering of the trajectory library (Table 1), so that the straight trajectory is preferred in the absence of obstacles.

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(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 = 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.

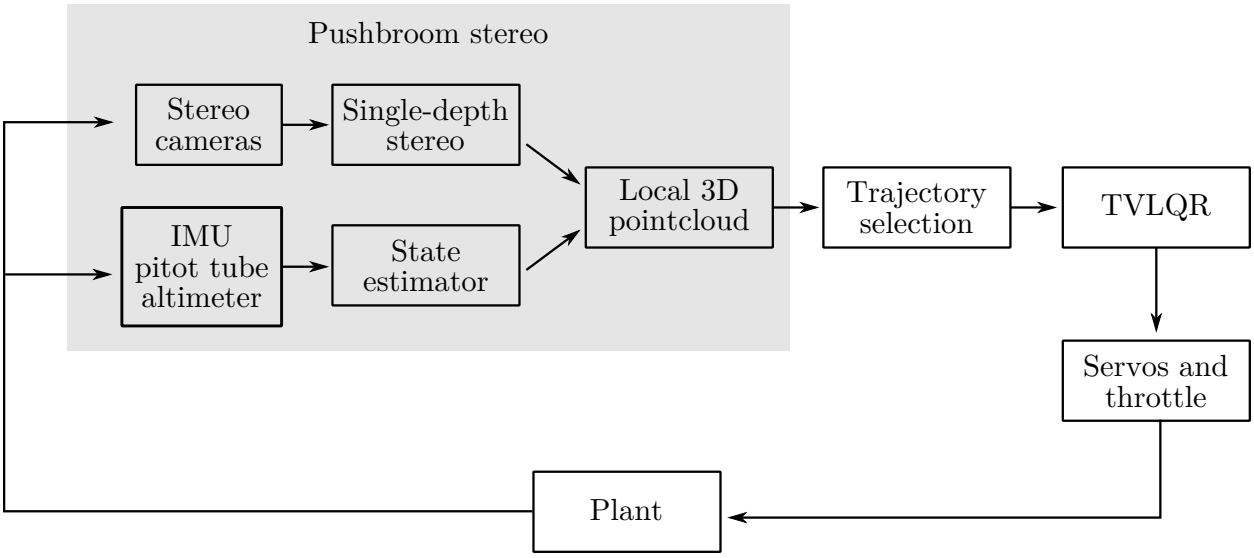


Figure 12: Control system diagram. The shaded region represents the pushbroom stereo subsystem.

6 Experimental Setup

To evaluate the integrated autonomous system capabilities, we performed obstacle avoidance experiments 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[8].

6.1 Aircraft Platform

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.

Our hardware (Figure 13) is based on a Team Black Sheep Caipirinha[9], modified to carry a substantial sensing, vision, and computation platform. We use two ODROID-U3[10] 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[11,12] 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 except for debugging), 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.

[8] We have performed successful autonomous landings, but it proved experimentally easier to land manually.
[9] Team Black Sheep, `http://team-blacksheep.com/products/prod:tbscaipirinha`
[10] Hardkernel Co. Ltd, `http://hardkernel.com`
[11] 3D Robotics, Inc. `http://3drobotics.com/`
[12] Firmware available: `https://github.com/andybarry/ardupilot/tree/arduread`

The stereo system is based on two Point Grey Firefly MV cameras[13] 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[14]. 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 120 fps on these cameras. Empirically we found USB synchronization to be satisfactory at this framerate, as the results show.

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). All test flights utilize an onboard, self-spooling, non-conductive 250 meter safety tether. The tether spools from the aircraft to avoid tangling on obstacles and prevents the aircraft from flying outside the bounds of our testing area.

6.1.1 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 (Figures 14, 15). We avoid a variety of trees, including a number of fully grown trees and one large dead tree.

6.2 Trajectories in Library

For most of the experiments near obstacles, we used a trajectory library with only seven trajectories (Table 1). Surprisingly, this library was sufficient for most of our flights, although we do not deny

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

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

Figure 13: Aircraft hardware on launcher.

Figure 14: Example of an artificial obstacle (left) and natural obstacles (right).

Figure 15: Depictions of the ”pair of trees” and ”many trees” environmental setups, overlaid on satellite imagery from Google Maps. For the ”pair of trees” environment, the launcher was positioned so the vehicle would approach a pair of trees, and then break through into a clearing. For the ”many trees” environment, the launcher was positioned so the vehicle would approach a small grove of trees, pass by the pair of trees, and then proceed past another triplet of trees.

that a larger library could have improved performance (see Section 8).

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 1: Trajectory library used in most obstacle avoidance experiments. TI stands for "time-invariant," which indicates the trajectory is at a trim condition. TV stands for "time-varying".

7 Autonomous System Results

Results from field experiments demonstrated successful obstacle avoidance capabilities of an autonomous aircraft using pushbroom stereo. We pushed the system to its limits, and analyze those boundaries here.

7.1 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 2: Statistics for experiments near obstacles.

We flew the aircraft on more than 130 flights, of which 26 were near obstacles. Over the 16 successful obstacle-avoidance flights, 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). The peak measured speed immediately before an avoidance maneuver

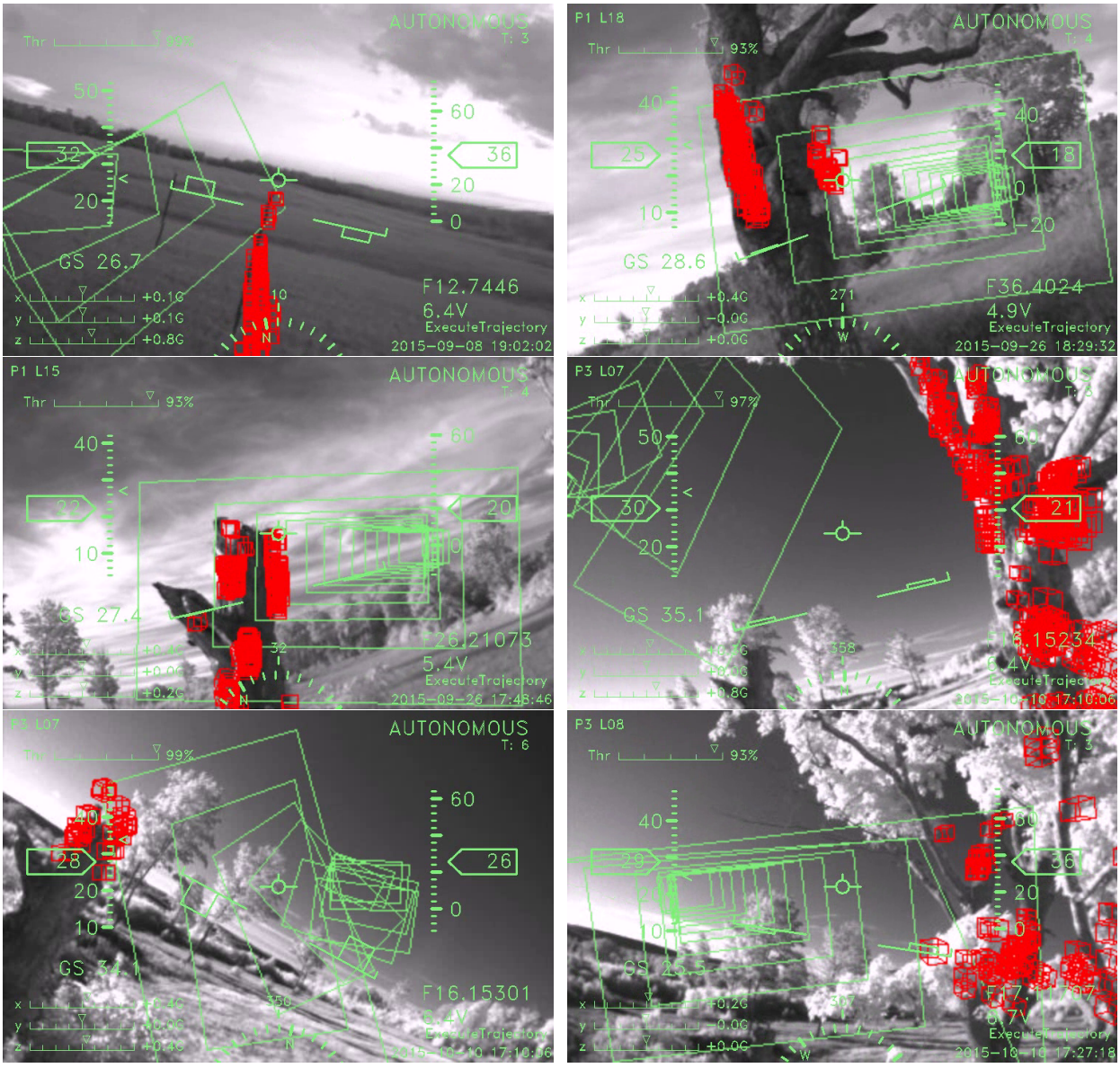


Figure 16: Onboard view of 6 different avoidance maneuvers. The red voxels show detected obstacles and the green squares show the planned trajectory as it is being executed. The top left view shows an artificial obstacle and the remaining 5 are trees.

was 14.0 m/s (31 MPH). Figure 16 demonstrates the aircraft's onboard view during six of these avoidance maneuvers.

Table 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 consisted of a living and dead tree next to each other, with successful paths both around and between the trees. Our final obstacle course consisted of the the same two trees but from a different angle, resulting in an additional tree to the right and more trees in the path behind the initial two.

7.2 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 it. During this flight, 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 17 shows this sequence of maneuvers from the onboard camera's view, Figure 18 shows an offboard view, Figure 19 presents the trajectories and obstacles in a 3D visualizer, and Figures 20 and 21 show the flight's planned trajectories and estimated tracking for position rotation, and control.

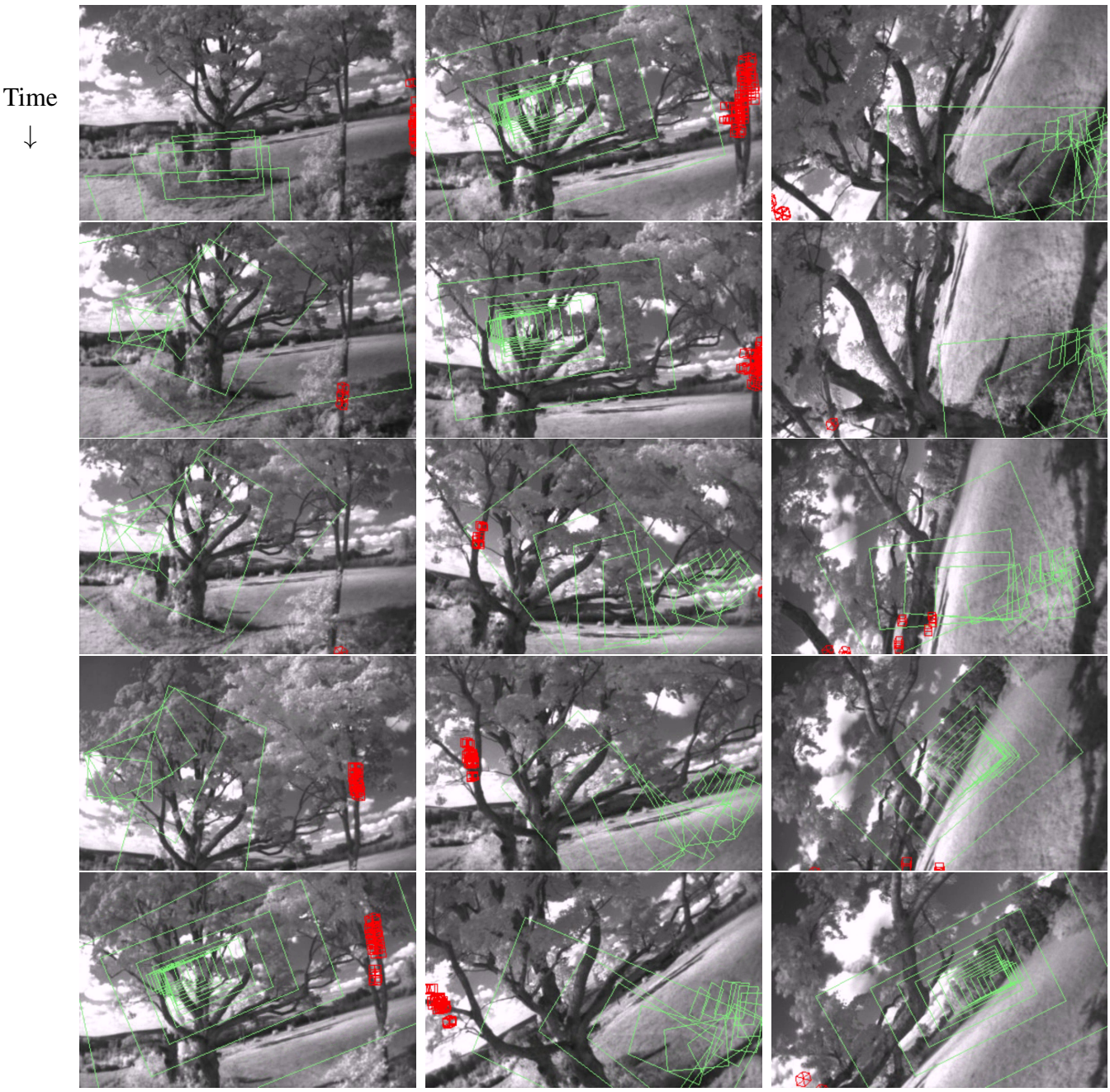


Figure 17: Autonomous obstacle avoidance from the onboard view. Time progresses down in columns and red voxels mark detected obstacles. The current trajectory is plotted as green boxes. Each frame is 0.0833 seconds (83.3ms) apart (1/10 actual framerate). The entire sequence in this figure lasts for 1.166 seconds and is sampled from a set of 140 frames.

Figure 18: Aircraft avoiding obstacles as viewed from a camera mounted on the tree seen in the left of the images in Figure 17. Each snapshot is 0.083 seconds apart and the total sequence spans 1.4 seconds.

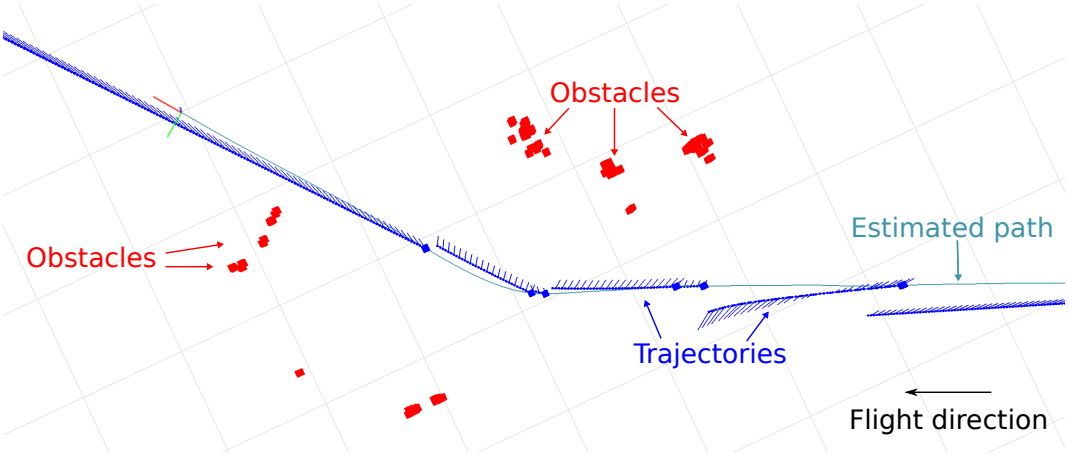


Figure 19: 3D visualization (top view) of the flight in Figure 17. Red boxes represent obstacles in the pointcloud, blue lines represent executed trajectories (including roll). The lighter solid line is the estimated path of the aircraft. The triad near the left represents the state of the aircraft near the end of the maneuver.

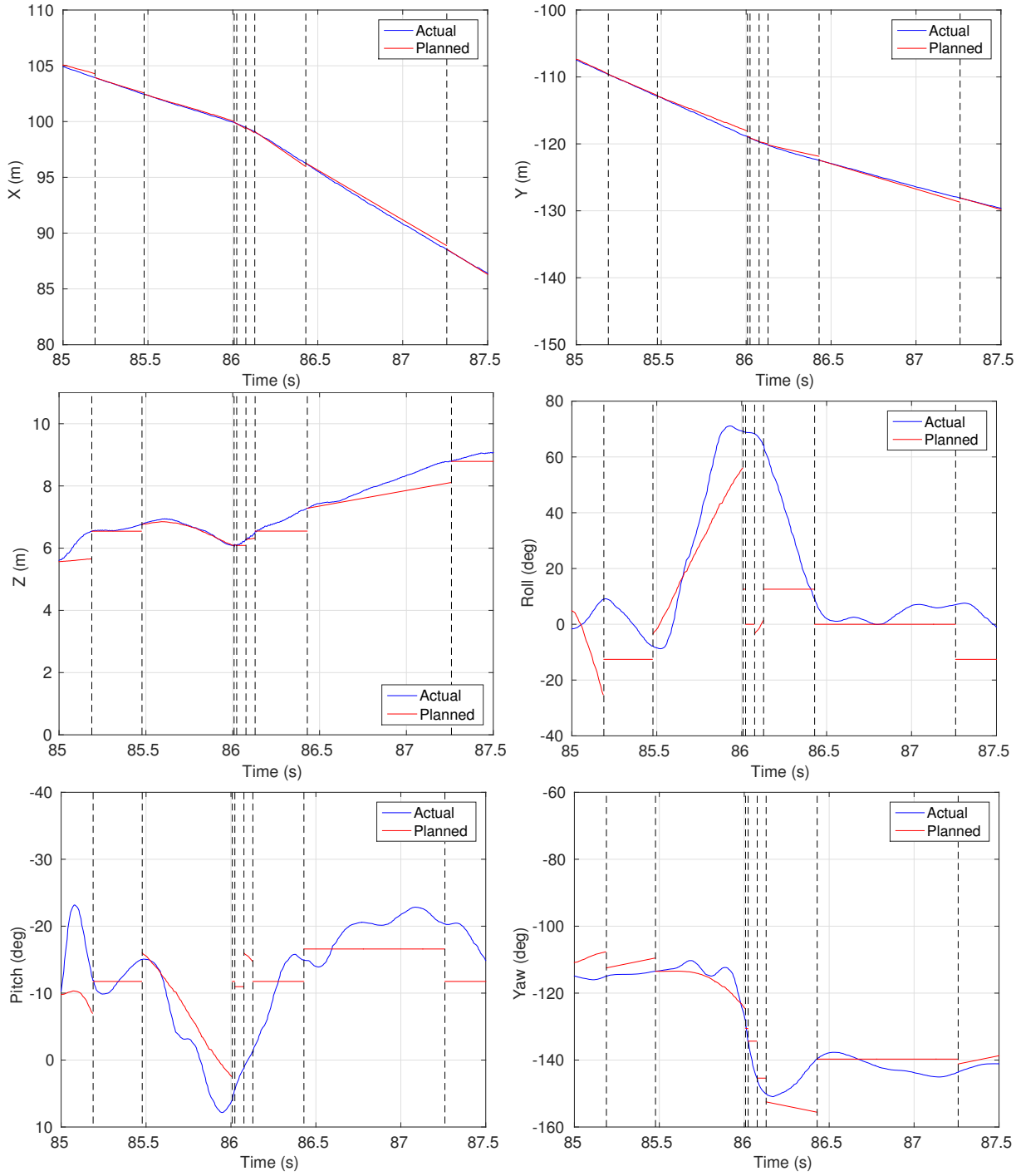


Figure 20: Planned and estimated tracking for an obstacle avoidance maneuver. Vertical dashed lines indicate a trajectory change. The first obstacle detection happens at $t = 85.15$ seconds. Notice the significant turn at $t = 86$ seconds where the aircraft dramatically alters its yaw angle to avoid an obstacle.

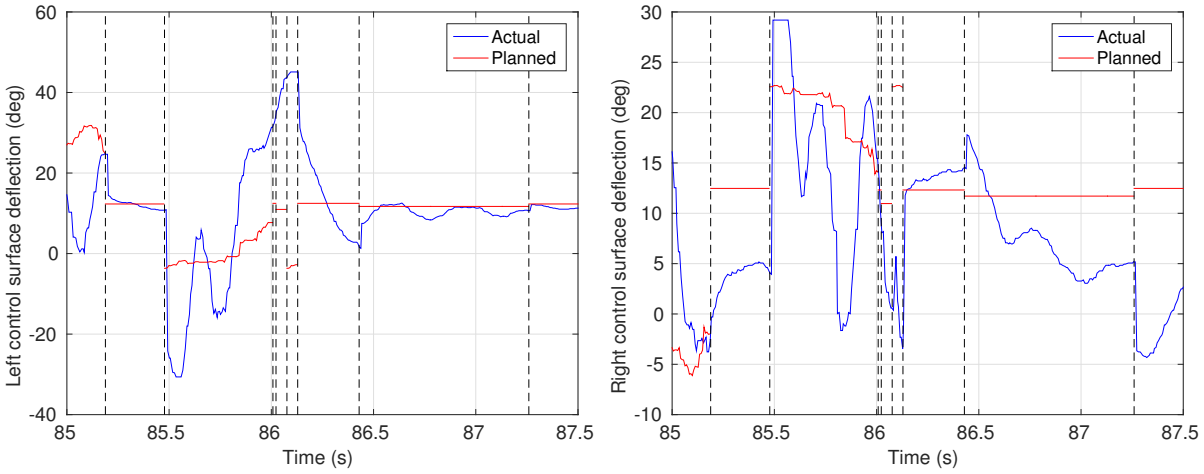


Figure 21: Planned and actual control inputs for an obstacle avoidance maneuver. The first obstacle detection happens at $t = 85.15$ seconds. We omit throttle here since it is commanded at, and runs at, approximately 100% throughout the entire maneuver.

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 3: Breakdown of system failures.

7.3 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 3).

7.3.1 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 22 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 23 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

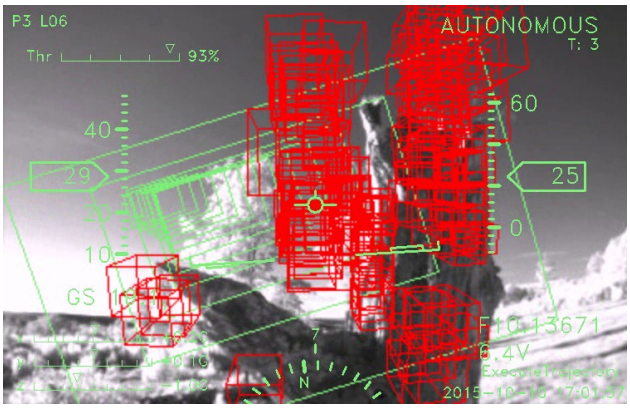


Figure 22: An example of a control failure. The vision system has clearly marked the dead tree ahead as an obstacle (red voxels), but the control system has failed to take appropriate corrective action in time. In this particular flight, the aircraft clipped its right wing but stabilized itself and continued flying.

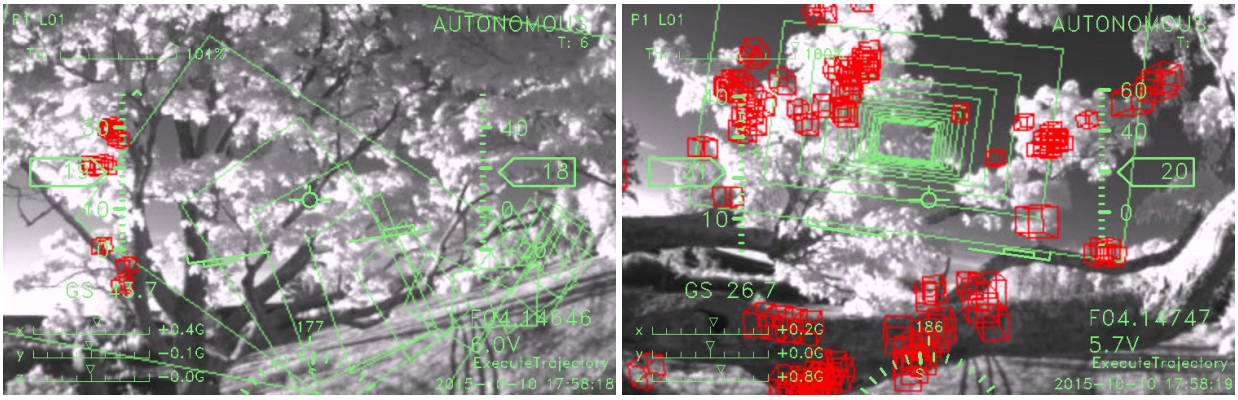


Figure 23: Failure case where the trajectory library was insufficient to avoid the obstacle. The left image shows the first frame where an obstacle is detected. The right image shows the aircraft about halfway through the tree's canopy and 0.2 seconds from impact.

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 obstacle, and then fails to fly the planned path. In this case, the problem lies with a limitation in our particular trajectory library. Our 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 began executing, the aircraft rolled to the *right*. This happened because the controller attempted to bring the starting state of the aircraft (rolled left) to the trajectory's starting state (level). As the trajectory began rolling left, the aircraft started to stop its roll to the right, but it was 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 the 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 24 shows the various state variables and control actions during this flight.

Overall, a trajectory library with only one starting state worked surprisingly well. Adding more trajectories is not particularly difficult, nor is choosing them online, but our experimental season ended before we could deploy them. Regardless, our results suggest that the number of starting states required for good performance may be quite small. The dangerous 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. Adding additional starting states will not burden the online system since it can immediately choose a set of trajectories with nearby starting states, adding no extra expensive collision checking or other processing.

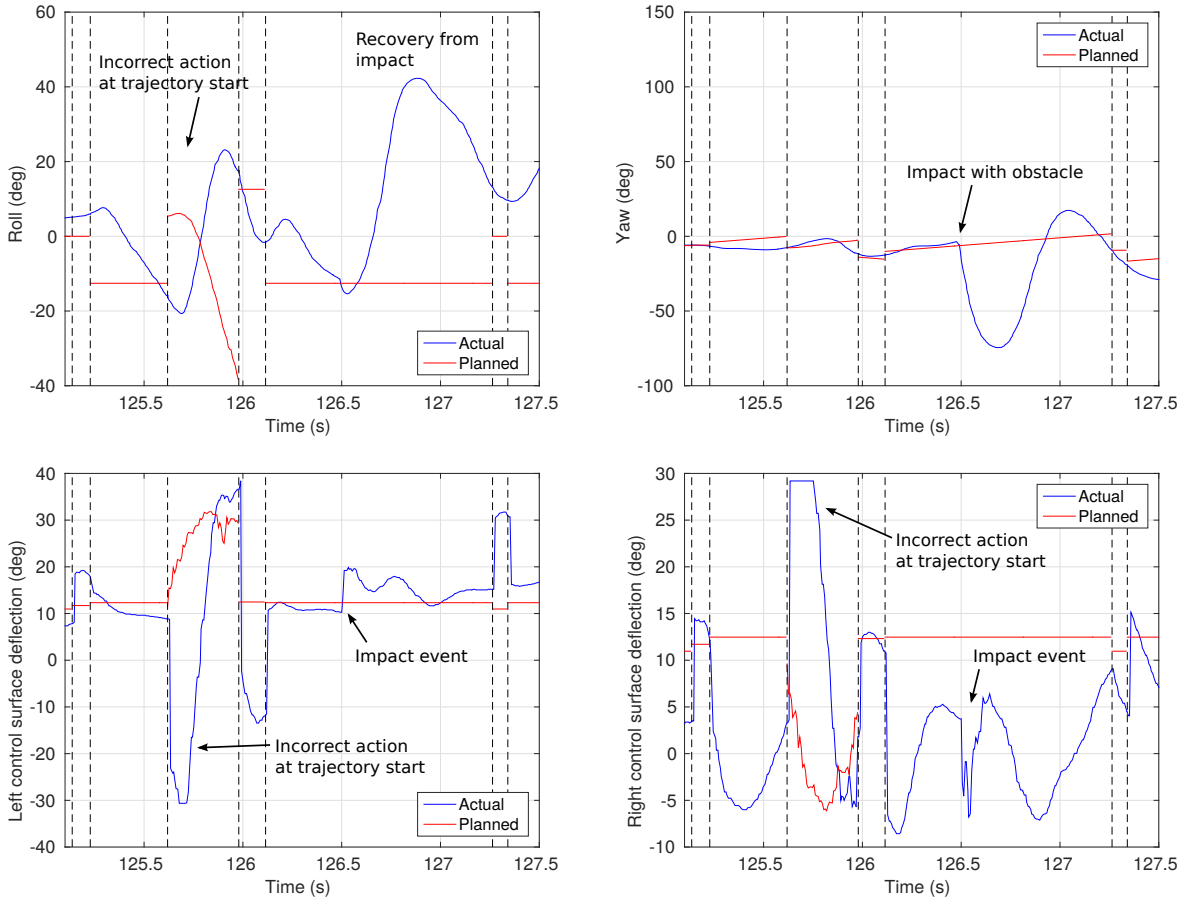


Figure 24: Roll, yaw, and control actions during a "trajectory initial condition" failure. In the top left (roll) the large discrepancy at the beginning of the trajectory is apparent, along with the incorrect control action taken (bottom plots). At approximately $t = 126.5$ the aircraft's right wingtip collides with an obstacle, producing a substantial yaw (about 74°), but the controller is able to recover and continue flying.

7.3.2 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.

8 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. With only using stereo matches at one depth,

there are limitations on the types of obstacle environments that can be handled, as analyzed in Section 2.7 of the author's doctoral thesis (Barry, 2016). 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 performance. Our aircraft collided with obstacles because of this limitation, which again, would not require much (if 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 (Moore and Tedrake, 2012) and expanded in (Majumdar et al., 2014) that would give us guarantees about when it was safe to switch trajectories, preventing some of the failures described in Section 7.3.1. Majumdar has even shown these techniques working in flight (Majumdar and Tedrake, 2017). 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 paper 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.

Videos / open source code

All code is open source and available:

- https://github.com/andybarry/flight
- https://github.com/andybarry/simflight
- http://drake.mit.edu
- https://github.com/ipab-slmc/pronto-distro
- https://github.com/andybarry/ardupilot/tree/arduread

Videos:

- https://www.youtube.com/watch?v=_qah8oIzCwk
- https://www.youtube.com/watch?v=iksfHQkkq88

Appendix A: Index to Multimedia Extensions

Extension	Media Type	Description
1	Video	This technical video includes videos from flight data, and provides an overview of pushbroom stereo and the presented integrated system.

Acknowledgements

The authors are grateful to John Carter for his help during field experiments and to Bill Freeman and Nick Roy for the fruitful discussions and comments. This work was supposed by the Office of

Naval Research (MURI grant #N00014-09-1-1051).

References

Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2006). An application of reinforcement learning to aerobatic helicopter flight. In *Proceedings of the Neural Information Processing Systems (NIPS '07)*, volume 19.

Barber, D. B., Griffiths, S. R., McLain, T. W., and Beard, R. W. (2007). Autonomous landing of miniature aerial vehicles. *Journal of Aerospace Computing, Information, and Communication*, 4(5):770–784.

Barry, A. J. (2016). *High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo*. PhD thesis, Massachusetts Institute of Technology.

Barry, A. J., Jenks, T., Majumdar, A., Lin, H.-T., Ros, I. G., Biewener, A., and Tedrake, R. (2014). Flying between obstacles with an autonomous knife-edge maneuver. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA), Video Track*.

Barry, A. J. and Tedrake, R. (2015). Pushbroom stereo for high-speed navigation in cluttered environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3046–3052. IEEE.

Beyeler, A., Zufferey, J.-C., and Floreano, D. (2009a). optiPilot: control of take-off and landing using optic flow. In *Proceedings of the 2009 European Micro Air Vehicle conference and competition (EMAV '09)*.

Beyeler, A., Zufferey, J.-C., and Floreano, D. (2009b). Vision-based control of near-obstacle flight. *Autonomous robots*, 27(3):201–219.

Bouffard, P., Aswani, A., and Tomlin, C. (2012). 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)*, pages 279–284. IEEE.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Bry, A., Bachrach, A., and Roy, N. (2012). 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)*, pages 1–8. IEEE.

Byrne, J., Cosgrove, M., and Mehra, R. (2006). Stereo based obstacle detection for an unmanned air vehicle. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2830–2835. IEEE.

Collins, R. T. (1996). A space-sweep approach to true multi-image matching. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*, pages 358–363. IEEE.

Cory, R. and Tedrake, R. (2008). Experiments in fixed-wing UAV perching. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, pages 1–12. AIAA.

DARPA (2014). Broad agency announcement fast lightweight autonomy (fla). *DARPA-BAA-15-16*.

Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). Monoslam: Real-time single

camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067.

De Wagter, C., Tijmons, S., Remes, B., and de Croon, G. (2014). 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.

Dey, D., Shankar, K. S., Zeng , S., Mehta, R., Agcayazi, M. T., Eriksen, C., Daftry, S., Hebert , M., and Bagnell, J. A. D. (2015). Vision and learning for deliberative monocular cluttered flight. In *Field and Service Robotics (FSR)*.

Frazzoli, E., Dahleh, M. A., and Feron, E. (2000). Robust hybrid control for autonomous vehicle motion planning. In *Proceedings of the 39th IEEE Conference on Decision and Control, 2000*, volume 1, pages 821–826. IEEE.

Gallup, D., Frahm, J.-M., Mordohai, P., Yang, Q., and Pollefeys, M. (2007). Real-time plane-sweeping stereo with multiple sweeping directions. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE.

Goldberg, S. B. and Matthies, L. (2011). 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)*, pages 169–176. IEEE.

Gupta, R. and Hartley, R. I. (1997). Linear pushbroom cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):963–975.

Harris, C. G. and Pike, J. (1988). 3d positional integration from image sequences. *Image and Vision Computing*, 6(2):87–90.

Hehn, M. and D'Andrea, R. (2011). A flying inverted pendulum. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 763–770. IEEE.

Hirschmuller, H. (2005). 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*, volume 2, pages 807–814. IEEE.

Honegger, D., Greisen, P., Meier, L., Tanskanen, P., and Pollefeys, M. (2012). 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)*, pages 5177–5182. IEEE.

Honegger, D., Oleynikova, H., and Pollefeys, M. (2014). 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*, Chicago, Illinois, USA. IEEE/RSJ.

Hrabar, S., Sukhatme, G., Corke, P., Usher, K., and Roberts, J. (2005). 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*, pages 3309–3316. IEEE.

Kim, J.-H. and Sukkarieh, S. (2003). Airborne simultaneous localisation and map building. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 406–411. IEEE.

Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 225–234. IEEE.

Kushleyev, A., Kumar, V., and Mellinger, D. (2012). Towards a swarm of agile micro quadrotors. In *Robotics: Science and Systems*.

Lindsey, Q., Mellinger, D., and Kumar, V. (2012). Construction with quadrotor teams. *Autonomous Robots*, 33(3):323–336.

Liu, S., Watterson, M., Tang, S., and Kumar, V. (2016). High speed navigation for quadrotors with limited onboard sensing. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1484–1491. IEEE.

Lupashin, S., Schollig, A., Sherback, M., and D'Andrea, R. (2010). A simple learning strategy for high-speed quadrocopter multi-flips. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, pages 1642–1648. IEEE.

Majumdar, A., Ahmadi, A. A., and Tedrake, R. (2014). Control and verification of high-dimensional systems with DSOS and SDSOS programming. In *Proceedings of the 53rd Conference on Decision and Control (CDC)*.

Majumdar, A. and Tedrake, R. (2012). Robust online motion planning with regions of finite time invariance. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, page 16, Cambridge, MA.

Majumdar, A. and Tedrake, R. (2017). Funnel libraries for real-time robust feedback motion planning. *International Journal of Robotics Research (IJRR), (To Appear)*.

Meier, L., Tanskanen, P., Heng, L., Lee, G. H., Fraundorfer, F., and Pollefeys, M. (2012). Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1-2):21–39.

Mellinger, D., Michael, N., and Kumar, V. (2010a). Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Proceedings of the 12th International Symposium on Experimental Robotics (ISER 2010)*.

Mellinger, D., Shomin, M., Michael, N., and Kumar, V. (2010b). Cooperative grasping and transport using multiple quadrotors. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*.

Michels, J., Saxena, A., and Ng, A. Y. (2005). High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 593–600. ACM.

Moore, J. and Tedrake, R. (2012). Control synthesis and verification for a perching UAV using LQR-trees. In *Proceedings of the IEEE Conference on Decision and Control*, page 8, Maui, Hawaii.

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

Napier, A., Corke, P., and Newman, P. (2013). 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)*, pages 3679–3684. IEEE.

Oleynikova, H., Honegger, D., and Pollefeys, M. (2015). Reactive avoidance using embedded stereo vision for mav flight. In *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 50–56. IEEE.

Richter, C., Bry, A., and Roy, N. (2013). Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *International Symposium of Robotics Research (ISRR)*, Singapore.

Ritz, R., Muller, M. W., Hehn, M., and D'Andrea, R. (2012). Cooperative quadrocopter ball throwing and catching. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4972–4978. IEEE.

Ross, S., Melik-Barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A., and Hebert, M. (2013). Learning monocular reactive uav control in cluttered natural environments. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1765–1772. IEEE.

Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

Shen, S., Mulgaonkar, Y., Michael, N., and Kumar, V. (2013). Vision-based state estimation and trajectory control towards aggressive flight with a quadrotor. In *Robotics: Science and Systems*, Berlin, Germany.

Sim, R., Elinas, P., Griffin, M., and Little, J. J. (2005). Vision-based slam using the rao-blackwellised particle filter. In *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, volume 14, pages 9–16.

Sobolic, F. M. (2009). Agile flight control techniques for a fixed-wing aircraft. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA.

Tedrake, R. (2009). LQR-Trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems (RSS)*, page 8.

Yang, R. and Pollefeys, M. (2003). Multi-resolution real-time stereo on commodity graphics hardware. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1. IEEE.

Zufferey, J.-C., Beyeler, A., and Floreano, D. (2008). Near-obstacle flight with small uavs. In *Proceedings of the International Symposium on Unmanned Aerial Vehicles*, Orlando, FL.