High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo

Andrew J. Barry
Computer Science and Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge, MA, USA
Email: abarry@csail.mit.edu

Russ Tedrake
Computer Science and Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge, MA, USA
Email: russt@csail.mit.edu

March 10, 2016

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 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.

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.

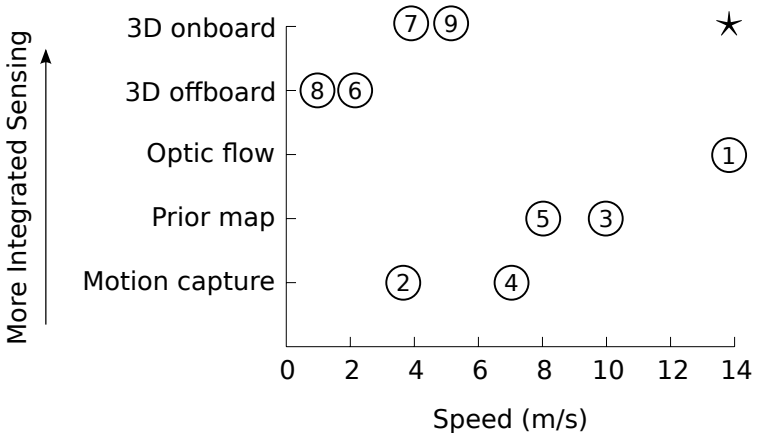
In this paper, we present a vision and control system that is capable of high-speed flight (10-14 m/s or 22-31 MPH) in natural environments. The system is completely self-contained on a small 34 inch wingspan, 664 gram aircraft, using only onboard sensing and onboard computation while in flight. To the 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.

1.1 Organization

In Section 2 we discuss related work followed by Sections 3–5 which introduce and evaluate our stereo vision algorithm. Sections 6–8 present the autonomous system and results. Finally, we offer concluding remarks in Section 9. We note that Section 3 was previously presented in [6] and the combined, autonomous system is presented in the author's doctoral thesis [4].

2 Related Work

2.1 Obstacle Avoidance with Micro Aerial Vehicles



1:	Beyeler, 2009	[9]
2:	Mellinger, 2010	[36]
3:	Bry, 2012	[12]
4:	Barry, 2012	[7]
5:	Richter, 2013	[43]
6:	Ross, 2013	[45]
7:	Shen, 2013	[47]
8:	Dey, 2015	[17]
9:	Oleynikova, 2015	[42]
⋆:	This work	–

Figure 1: Plot of related work in small UAV obstacle avoidance examining speed and sensor integration. The ⋆ 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, remains an open problem. 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 maps some of the related work in the field based on sensor integration and speed.

2.1.1 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

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

autonomous helicopter maneuvers [1] was an early demonstration of these techniques. Mellinger and Kumar demonstrate quadrotor flight through small openings [36] and in formation [37, 29]. Quadrotors have further been shown to perform agressive flips [31] and interact with objects including catching and juggling a ball [10, 40], balancing a pendulum [22], cooperatively throwing a ball [44], and building structures [30].

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 [49] and Cory demonstrated autonomous perching on a glider, landing on a wire with impressive 5cm accuracy [14]. We have previously demonstrated flight through a gap smaller than the aircraft's wingspan [5].

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 [12, 43]. 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[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.

2.2.1 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 [13]. 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 [26]. Yang and Pollefeys, among others, implemented stereo on GPUs for significant processing gains [54]. 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 [35]. 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 [19]. 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 [16]. Honegger et al. show that stereo and optical flow matching is possible

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

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) [25, 24] with their latest work demonstrating onboard obstacle avoidance at 5 m/s [42].

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 [38]. 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 [17].

2.2.3 Optic Flow

Optical flow techniques work well, controlling stable flight, takeoff, landing [3, 8], and obstacle avoidance [9, 55]. 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.

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 [15, 21, 48] on MAVs [27]. There are a few fundamental problems with VSLAM on MAVs, notably: (1) computational power and (2) robustness of the navigation solution.

Recently, researchers have begun using parallel tracking and mapping (PTAM) [28] 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] senseFly Ltd., `http://sensefly.com`

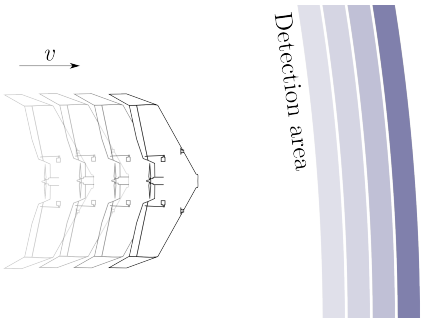


Figure 2: By detecting at a single depth (dark blue) and integrating the aircraft's odometry and past detections (lighter blue), we can quickly build a full map of obstacles in front of our vehicle.

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[4] 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.

3.2 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). 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.

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 [41]). 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 [20]. These cameras are often found on satellites and can be used for stereo vision [23].

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

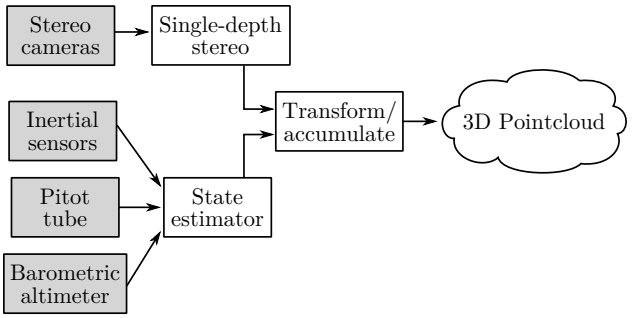


Figure 3: Pushbroom stereo overview. Note that the system does not require GPS.

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.

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.

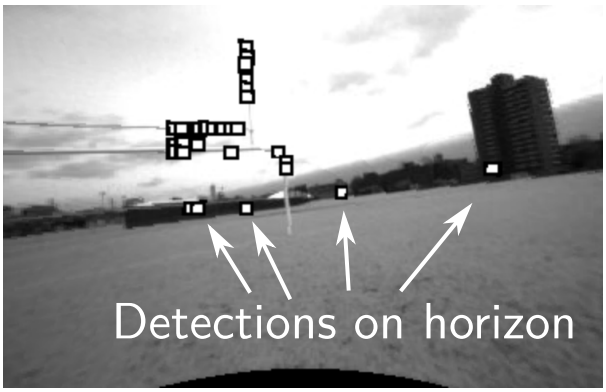
After rejecting blocks for lack of edges, we score the remaining blocks based on SAD match divided by the summation of edge-values in the pixel-block:

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

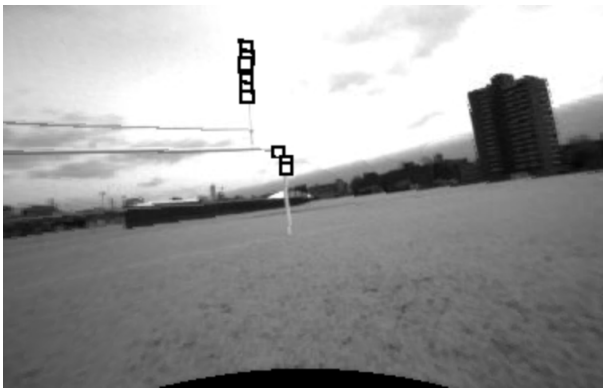
where $p(i)$ denotes a pixel value in the 5x5 block and L is the Laplacian. We then threshold on the score, S, to determine if there is a match.

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 uninter-



(a) Without horizontal invariance filter.



(b) With horizontal invariance filter.

Figure 4: 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.

rupted 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 this filter online. Figure 4 demonstrates this filter running on flight data.

4 Pushbroom Stereo Results

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 onboard stereo system with the state-estimator disabled[5]. We then, offline, used OpenCV's [11] 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 each single-disparity stereo coordinate to the nearest point produced by the depth-cropped StereoBM (Figure 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 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

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

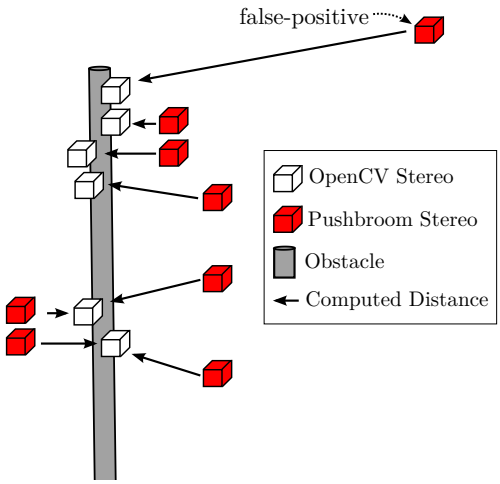


Figure 5: Sketch of our evaluation strategy for single-disparity stereo. We detect false-positives by computing the distance from single-disparity stereo's output (red) to the nearest point from OpenCV's `StereoBM` (white). False positives stand out with large distances (labeled box).

produces points within 0.5 meters of StereoBM 60.9% and within 1.0 meters 71.2% of the time (Figure 6). For context, the aircraft's wingspan is 0.86 meters and it covers 0.5 meters in 0.03 to 0.07 seconds.

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 7 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 6.2) 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 7.

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 9a).

Figures 8 and 9b 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 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 9 shows, pushbroom stereo detects most of the points on obstacles that StereoBM sees, miss-

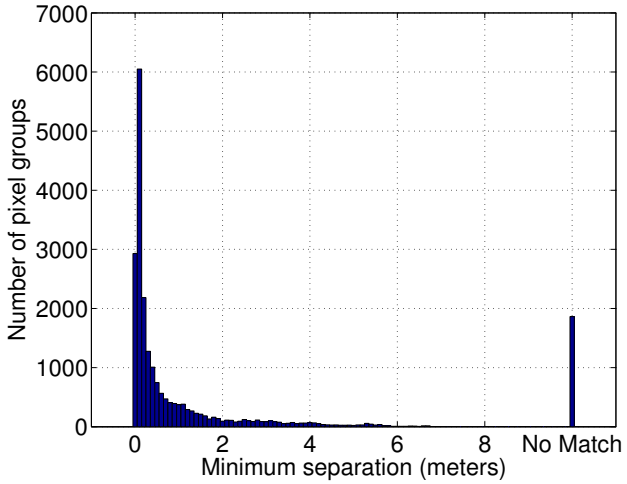


Figure 6: Results of the false-positive benchmark described in Figure 5 on 23,000+ frames. *No Match* indicates single-disparity points where there was no matching StereoBM point on the frame. We find that 8.2% of detected pixels fall into this category.

ing 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 the detection distance, lens focal length, and camera calibration between Sections 3 and 7.

5 Analysis of Obstacle Avoidance Limits

5.1 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. 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 10):

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

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

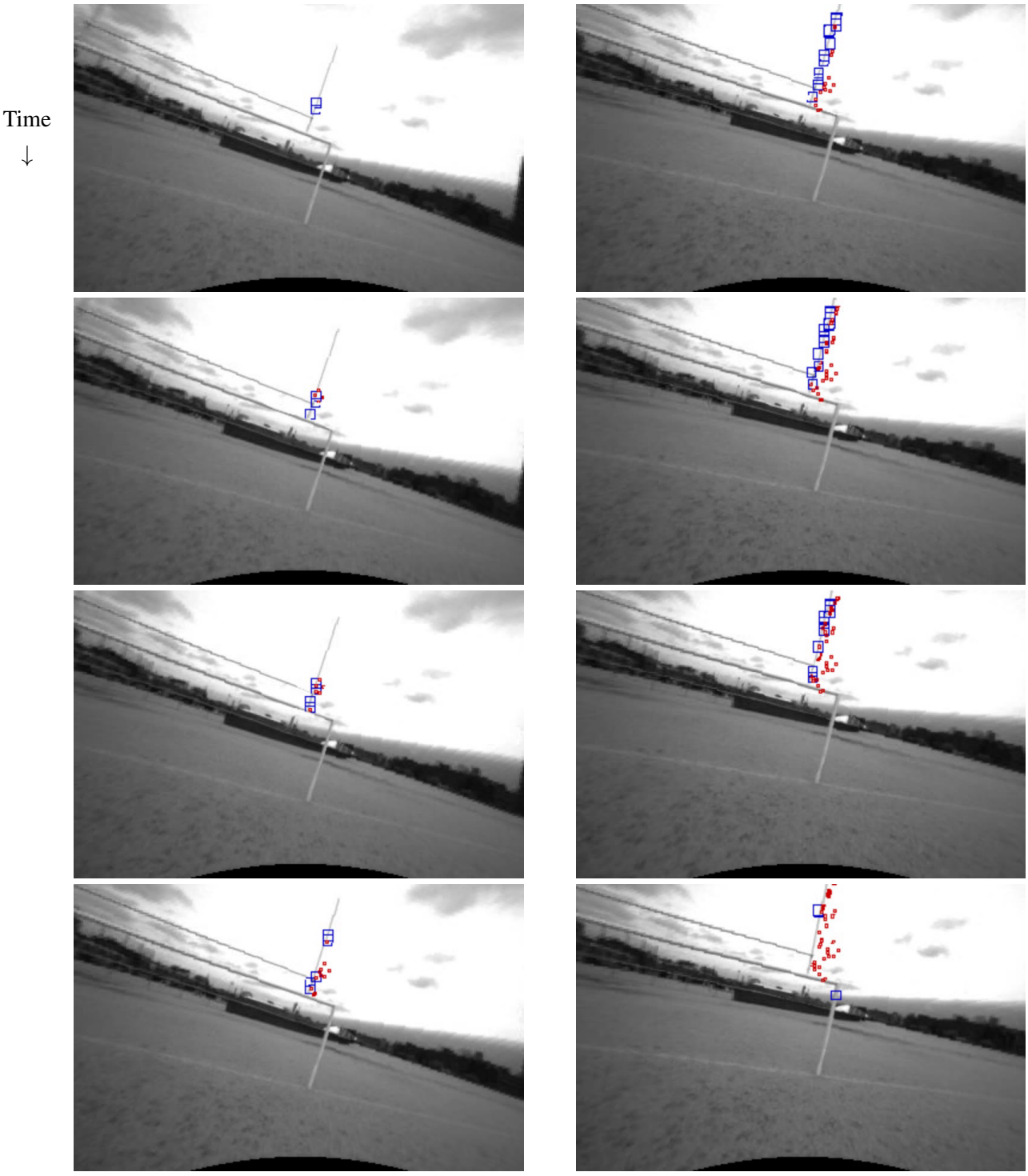


Figure 7: Sequence of stills from an obstacle detection. Each image is 0.02 seconds (20ms) after the previous. The entire set captures 0.16 seconds. Here, the fieldgoal is detected in the first frames (blue boxes). Afterwards, the position of those detections is estimated via the state estimator and reprojected back onto the image (red dots).

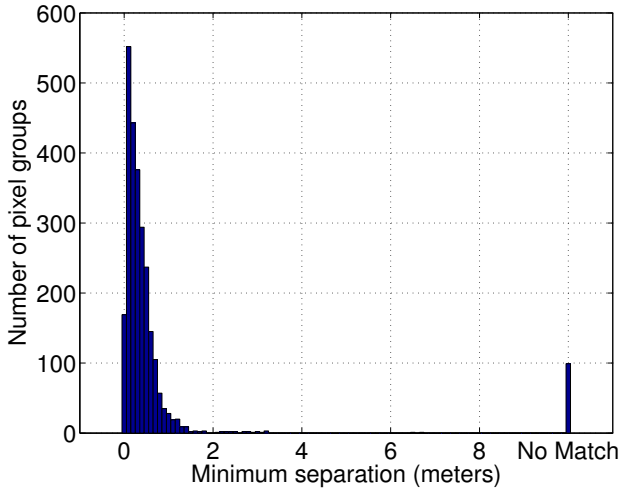
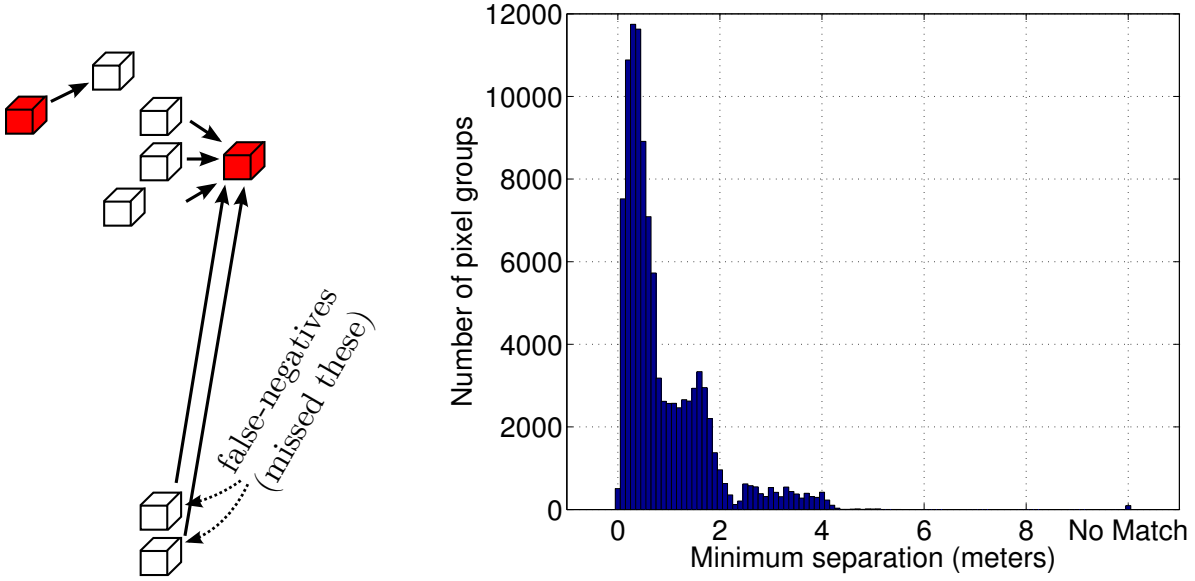


Figure 8: Results of the comparison as described in Figure 5 (a). Our system produces few outliers (74.8% and 92.6% within 0.5 and 1.0 meters respectively), even as we integrate our state estimate, and the obstacle positions, forward. *No Match* indicates points that pushbroom stereo detected but there were no block-matching stereo detections on that frame.



(a) Comparison technique.

(b) False-negative comparison.

Figure 9: 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.

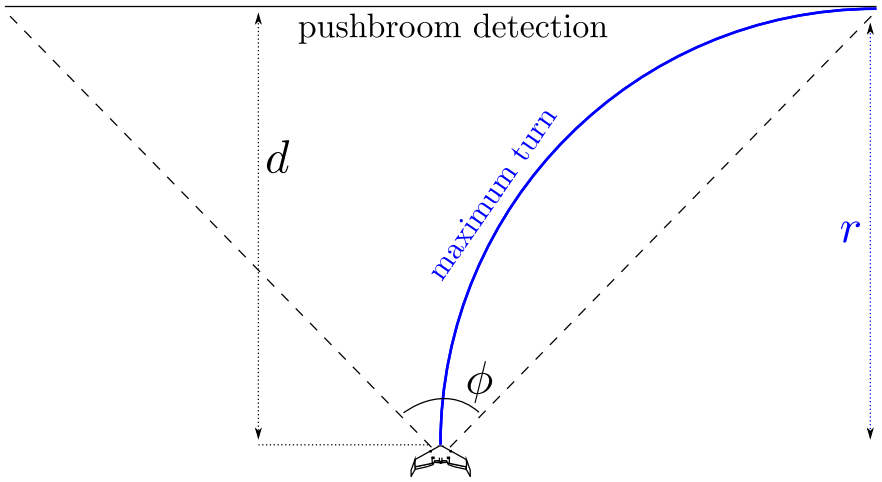


Figure 10: Sketch of the 2D obstacle-avoidance case showing the pushbroom detection distance (d), the field of view (ϕ), and the turning radius (r). In this drawing, $d = r$ and $\phi = 90^\circ$.

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 $2r$.

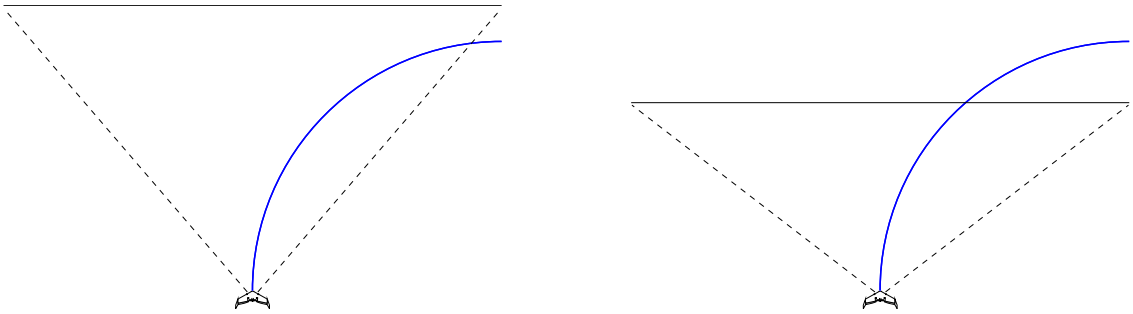
If $\phi > 90^\circ$, the system is capable of seeing areas it cannot reach, so no limits are improved or reduced. Clearly, the aircraft could reach those regions with multiple future maneuvers, but we discount that case since pushbroom stereo removes past obstacles from memory relatively quickly. If, 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 11, 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 12 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.

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



(a) Case II ($d > r$). (b) Case III ($d < r$).

Figure 11: Sketches of Cases II and III.

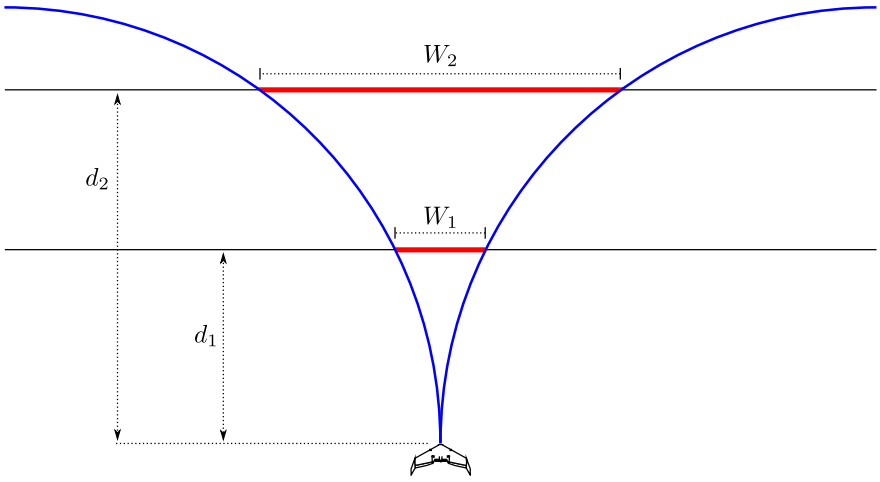


Figure 12: The maximum width of an obstacle (excluding the aircraft's geometry) when $d < r$ is $W = 2\left(r - \sqrt{r^2 - d^2}\right)$.

can combine these results to find an expression for W:

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

Our system described below has $d = 10m$ and $\phi = 92^\circ$. For our aircraft using the trajectory library, $r_{nominal} = 70.8m$ 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 $70.8m$.

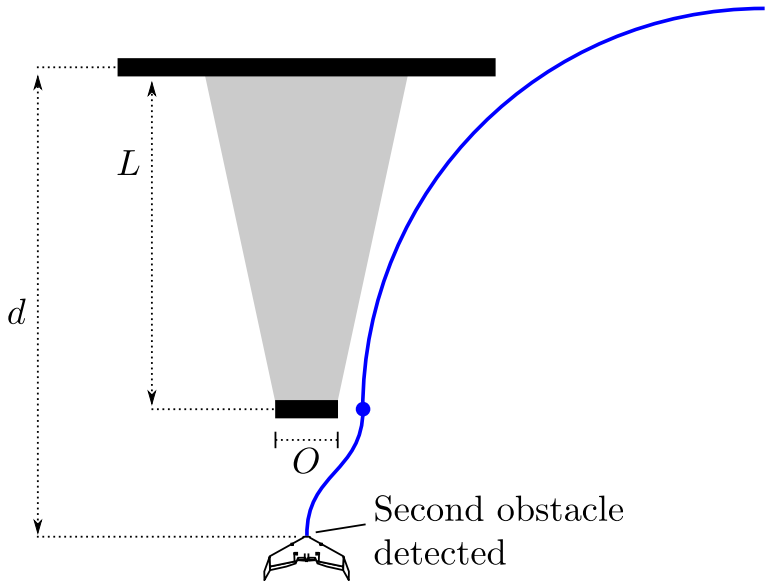


Figure 13: Sketch of the obstacles with occlusions case. The gray region shows the occluded area.

5.2 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 13). 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:

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

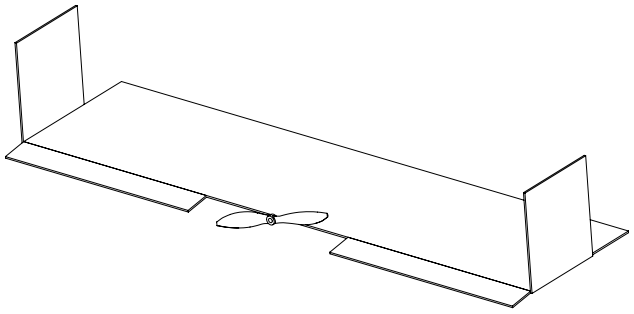


Figure 14: Sketch of the flat plate model with a single flat plate for the wing, two flat plate elevons, two flat plate winglets, and a thurst model.

6 Autonomous Control

To perform autonomous obstacle avoidance, we use a model-based control system consisting of trajectories selected from a library in realtime 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 section details the aircraft model, trajectory library generation, control, and integration with the vision system.

6.1 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 [14]. We model the aircraft with three fixed flat plates representing the wing and winglets and two smaller moving flat plates for the elevons (Figure 14). We model the propeller as a thrust-generating element, but ignore its aerodynamic drag, blade flapping, and other higher order effects.

6.2 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.

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 [12][6]. 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 except for x and y and can use pushbroom stereo.

6.3 Trajectory Libraries

A common 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. A robust implementation of this idea relies on three main things: trajectory generation, choice of distance function, and good model performance [50].

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 [18]. 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 [2].

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 [34]. In recent work, Majumdar has shown a full working system that uses these guarantees to aggressively avoid obstacles [33].

Our system relies on trajectories computed offline and then selects a trajectory to execute online, similar to [18]. 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.

6.3.1 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:

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

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

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

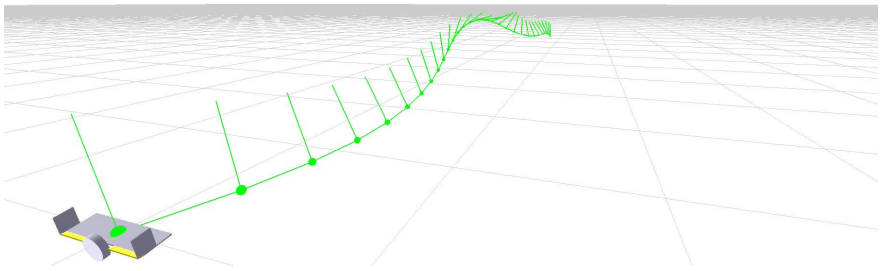


Figure 15: Knife-edge trajectory optimized and simulated in Drake.

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.

6.3.2 Trajectory Optimization

To build trajectories with non-zero, time-varying accelerations directly from the aircraft model, we use a direct-collocation method like [51] implemented in Drake [52]. 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 15 shows a knife-edge trajectory, with a roll contraint of 90° at $t = t_F/2$, being optimized.

6.3.3 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.

6.4 Feedback Control

6.4.1 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 use a standard, time-invariant LQR controller:

First, we define error coordinates:

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

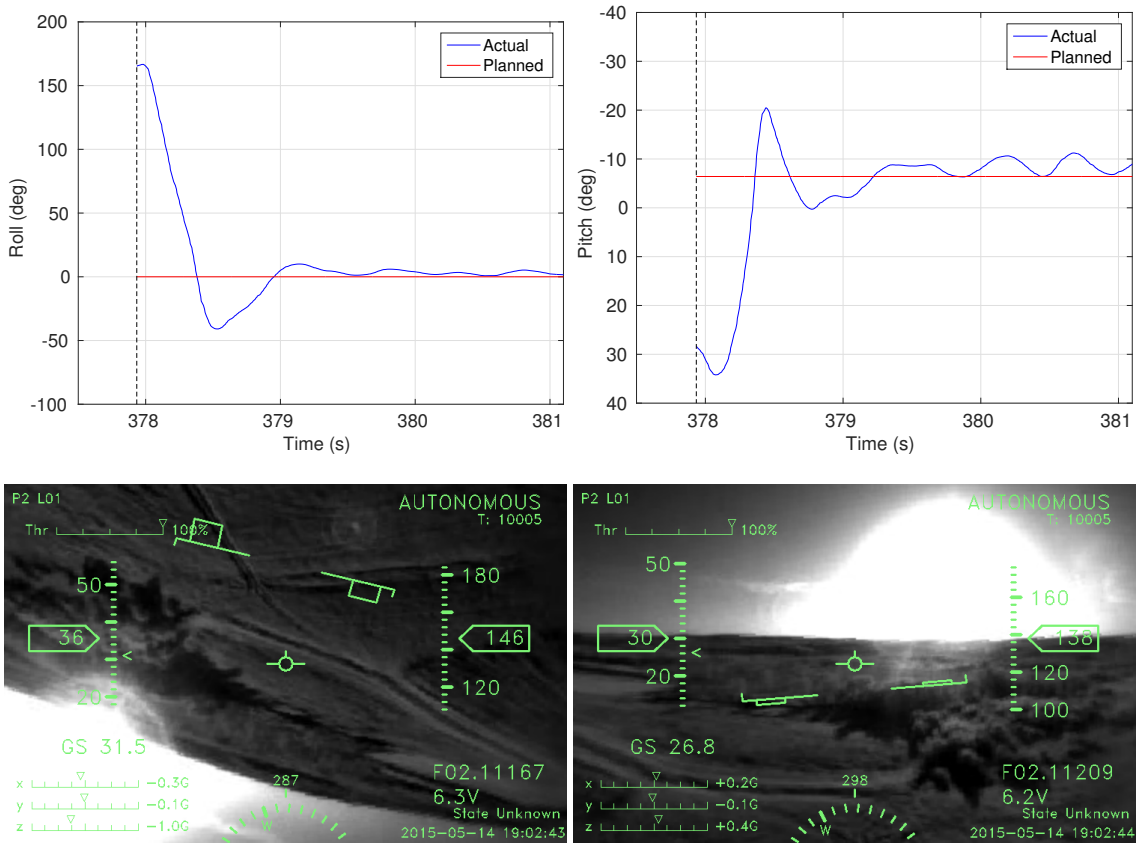


Figure 16: Top: roll and pitch traces for autonomous recovery from inverted flight. The controller is enabled at the black vertical line. Bottom: Onboard view at the beginning and end of the recovery.

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

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 16 shows the system recovering from inverted flight.

6.4.2 Time-Varying Control

To provide feedback control for time-varying trajectories, we use a time-varying linear quadratic regulator (TVLQR) as in [53]. 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) to substan-

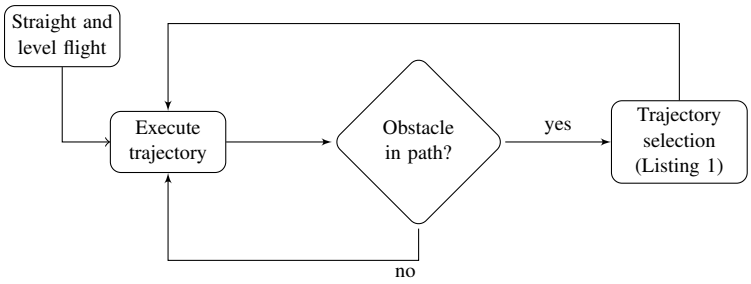


Figure 17: Diagram of the online planning system. An obstacle is deemed to be in the path if the minimum distance between the trajectory and the point cloud is below a threshold.

tially reduce the number of trajectories required in the library. Thus, when a trajectory is selected online, we record those four initial states and transform the trajectory along those coordinates.

6.5 Online Planning

6.5.1 Online Planning Algorithm

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 17 outlines the system and Listing 1 shows the trajectory selection algorithm.

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.

6.5.2 Point Cloud Management

To enable fast nearest-neighbor searches through a potentially large point cloud, we use an octree structure implemented in PCL [46]. 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.

6.5.3 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).

7 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.

7.1 Aircraft Platform

Our hardware (Figure 18) is based on a Team Black Sheep Caipirinha[7], modified to carry a substantial sensing, vision, and computation platform. We use two ODROID-U3[8] 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[9,10] 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.

The stereo system is based on two Point Grey Firefly MV cameras[11] 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[12]. The cameras are not hardware synchronized, but instead rely on

[7] Team Black Sheep, `http://team-blacksheep.com/products/prod:tbscaipirinha`

[8] Hardkernel Co. Ltd, `http://hardkernel.com`

[9] 3D Robotics, Inc. `http://3drobotics.com/`

[10] Firmware available: `https://github.com/andybarry/ardupilot/tree/arduread`

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

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

Figure 18: Aircraft hardware on launcher.

USB to stay in sync. We note that it is not possible to achieve both hardware synchronization and 120 fps 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, 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.

7.2 Experimental Setup

In addition to the manual flight 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[13]. Figure 19 shows a diagram of the experiment's flight phases.

7.2.1 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 20). We avoid a variety of trees, including a number of fully grown trees and one large dead tree.

7.3 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 that a larger library could have improved performance (see Section 9).

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

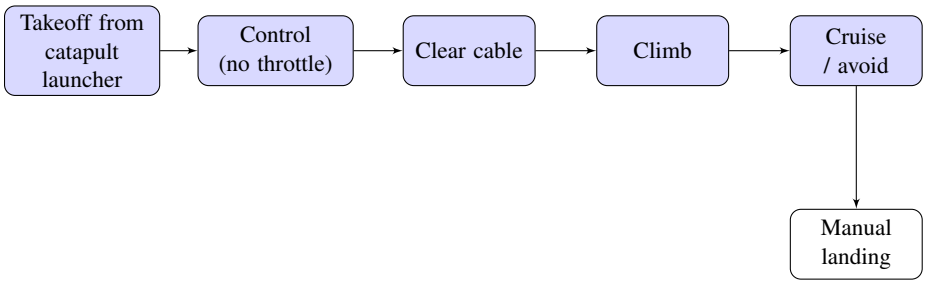


Figure 19: Experimental phases. Autonomous modes are labeled in blue.

Figure 20: Example of an artificial obstacle (left) and natural obstacles (right).

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 (see Section 6.4.1). TV stands for "time-varying" (see Section 6.4.2).

8 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 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 8.3 details the types of failures and suggests future work for their mitigation.

8.1 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.

On 16 successful flights (Table 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 21 demonstrates the aircraft's onboard view during six of these avoidance maneuvers and Figure 22 shows a chase-camera view.

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, 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.

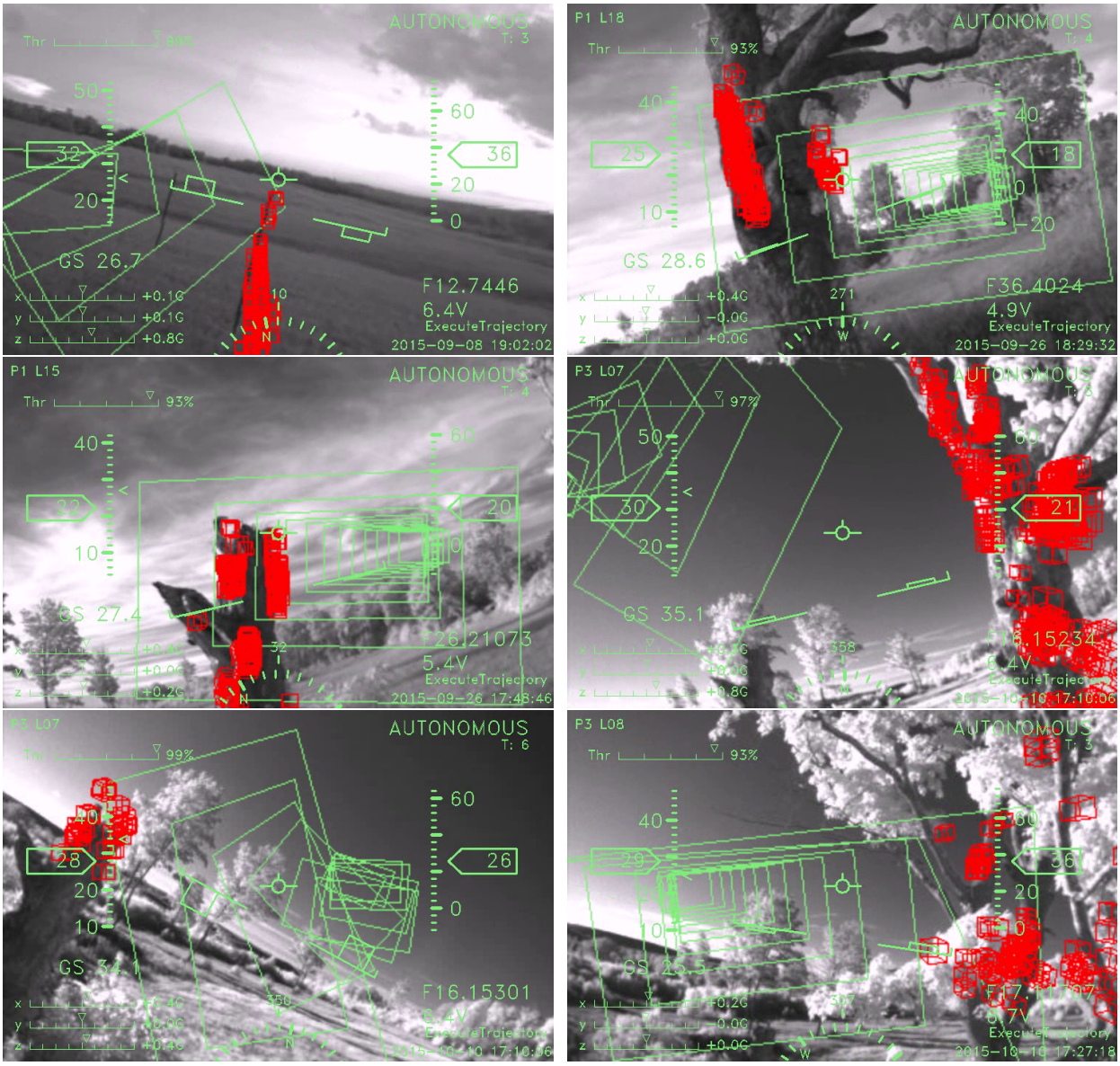


Figure 21: Onboard view of 6 different avoidance maneuvers. The red voxels show detected obstacles and the green squares show the planned trajectory as it is being executed. The top left view shows an artificial obstacle and the remaining 5 are trees.

Figure 22: Chase camera view of the autonomous aircraft avoiding a tree.

8.2 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. 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 23 shows this sequence of maneuvers from the onboard camera's view, Figure 24 shows an offboard view, Figure 25 presents the trajectories and obstacles in a 3D visualizer, and Figures 26 and 27 show the flight's planned trajectories and estimated tracking for position rotation, and control.

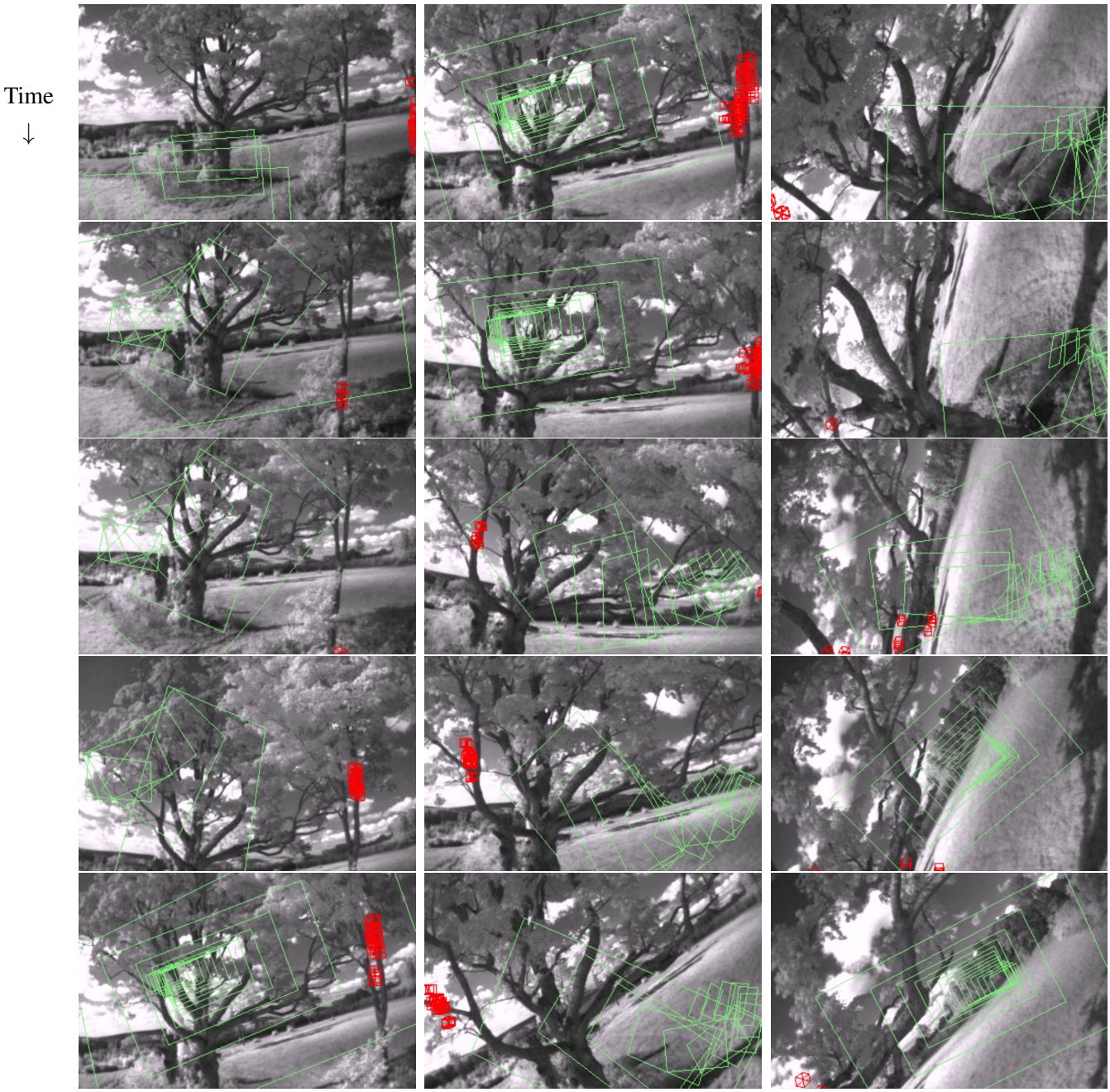


Figure 23: Autonomous obstacle avoidance from the onboard view. Time progresses down in columns and red voxels mark detected obstacles. The current trajectory is plotted as green boxes. Each frame is 0.0833 seconds (83.3ms) apart (1/10 actual framerate). The entire sequence in this figure lasts for 1.166 seconds and is sampled from a set of 140 frames.

Figure 24: Aircraft avoiding obstacles as viewed from a camera mounted on the tree seen in the left of the images in Figure 23. Each snapshot is 0.083 seconds apart and the total sequence spans 1.4 seconds.

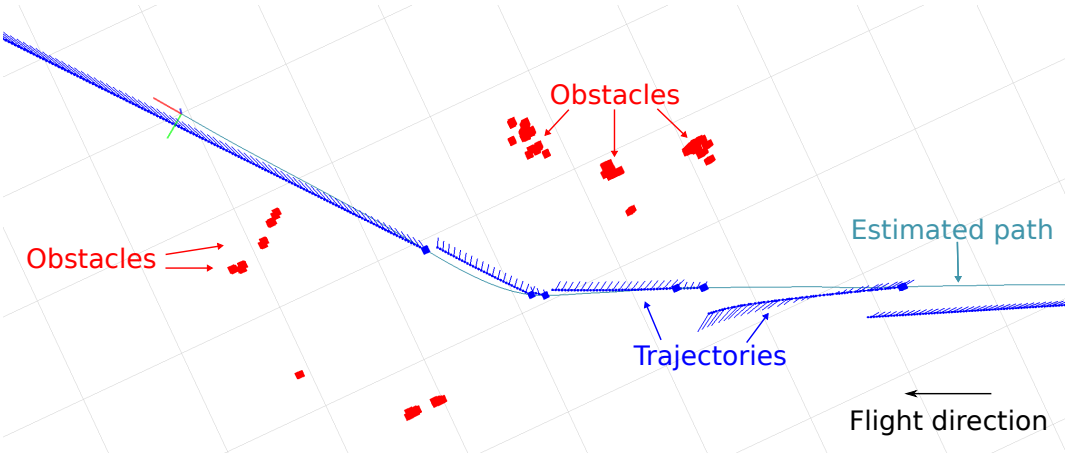


Figure 25: 3D visualization (top view) of the flight in Figure 23. Red boxes represent obstacles in the pointcloud, blue lines represent executed trajectories (including roll). The lighter solid line is the estimated path of the aircraft. The triad near the left represents the state of the aircraft near the end of the maneuver.

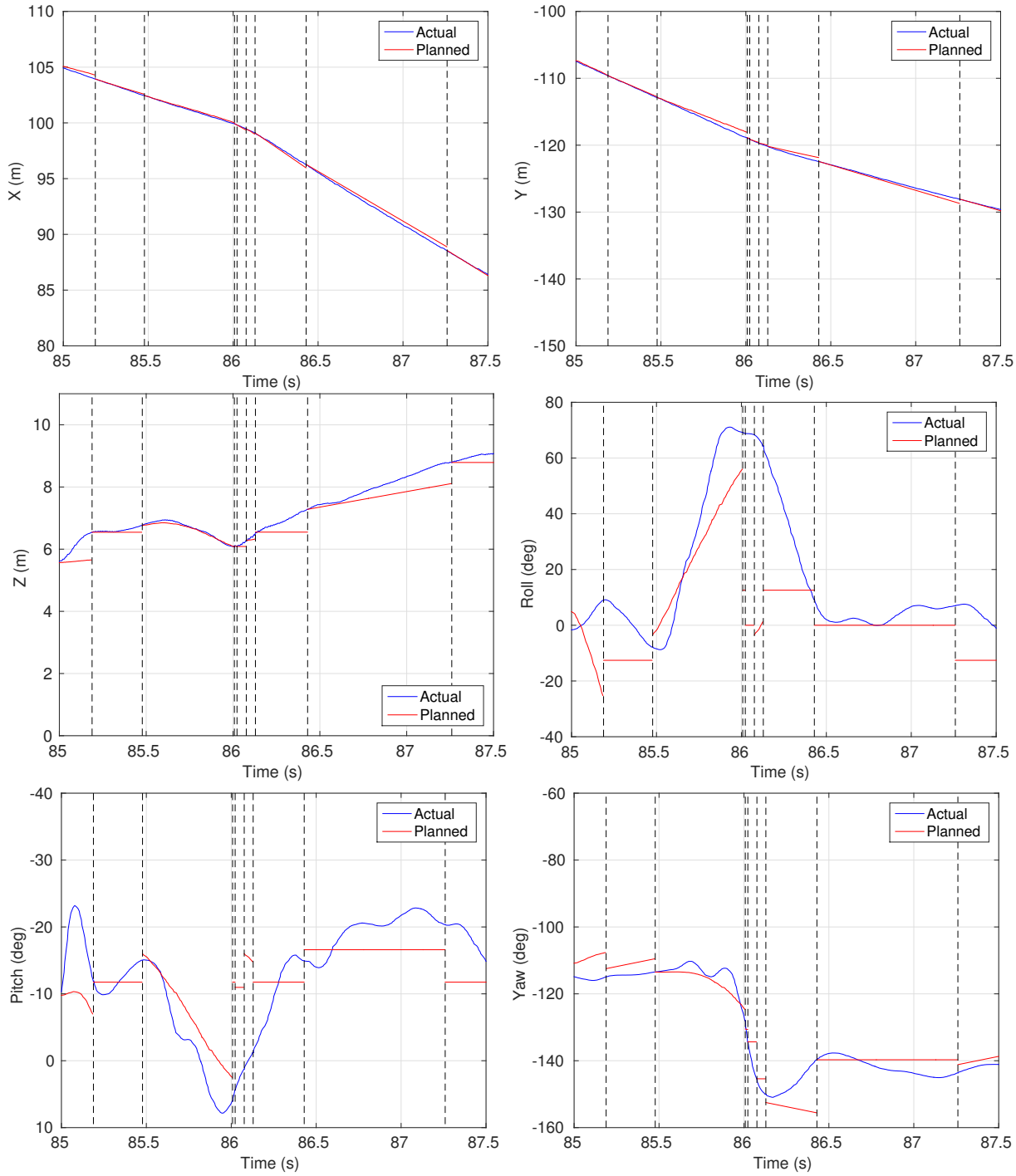


Figure 26: Planned and estimated tracking for an obstacle avoidance maneuver. Vertical dashed lines indicate a trajectory change. The first obstacle detection happens at $t = 85.15$ seconds. Notice the significant turn at $t = 86$ seconds where the aircraft dramatically alters its yaw angle to avoid an obstacle.

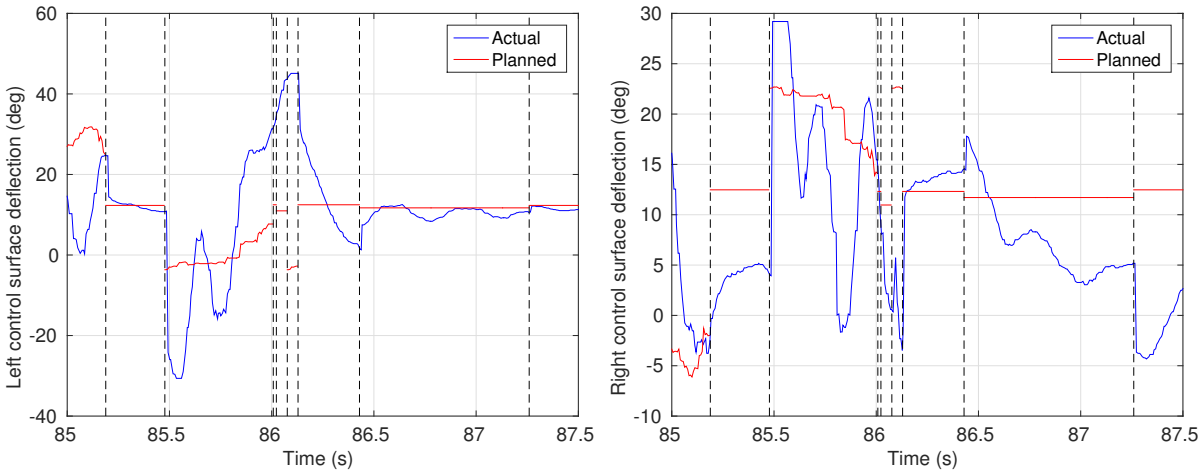


Figure 27: Planned and actual control inputs for an obstacle avoidance maneuver. The first obstacle detection happens at $t = 85.15$ seconds. We omit throttle here since it is commanded at, and runs at, approximately 100% throughout the entire maneuver.

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.

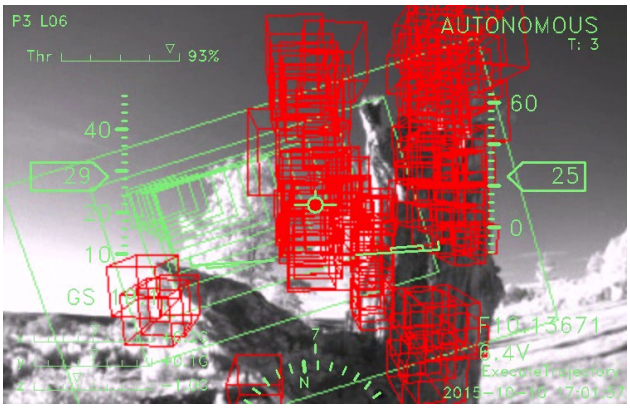


Figure 28: An example of a control failure. The vision system has clearly marked the dead tree ahead as an obstacle (red voxels), but the control system has failed to take appropriate corrective action in time. In this particular flight, the aircraft clipped its right wing but stabilized itself and continued flying.

8.3 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).

8.3.1 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 28 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 29 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

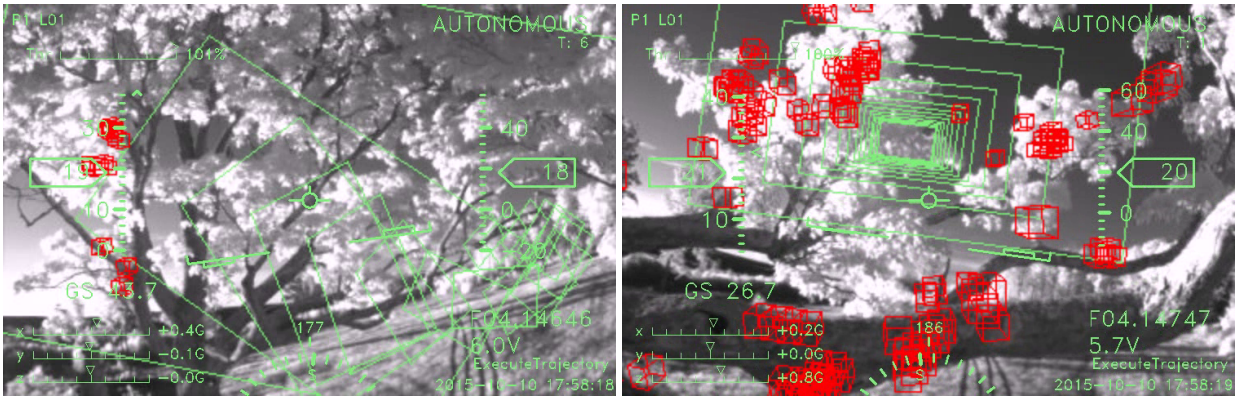


Figure 29: Failure case where the trajectory library was insufficient to avoid the obstacle. The left image shows the first frame where an obstacle is detected. The right image shows the aircraft about halfway through the tree's canopy and 0.2 seconds from impact.

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 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 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 30 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.

8.3.2 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,

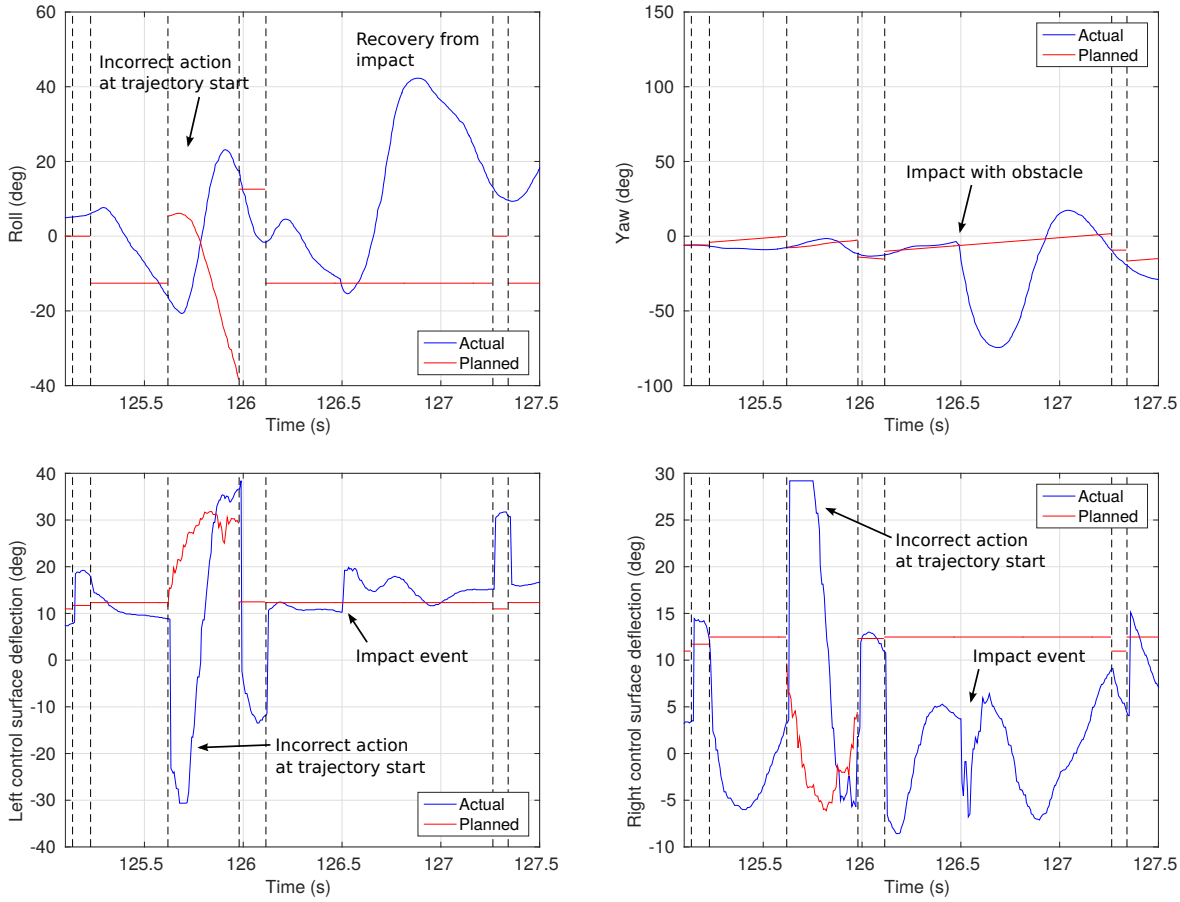


Figure 30: Roll, yaw, and control actions during a "trajectory initial condition" failure. In the top left (roll) the large discrepancy at the beginning of the trajectory is apparent, along with the incorrect control action taken (bottom plots). At approximately $t = 126.5$ the aircraft's right wingtip collides with an obstacle, producing a substantial yaw (about 74°), but the controller is able to recover and continue flying.

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.

9 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 performance. 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 [39] and expanded in [32] that would give us guarantees about when it was safe to switch trajectories, preventing some of the failures described in Section 8.3.1. Majumdar has even shown these techniques working in flight [33]. 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`

Acknowledgements

The authors are grateful to Pete Florence and John Carter for their 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

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

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

[4] Andrew J. Barry. "High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo". PhD thesis. Massachusetts Institute of Technology, 2016.

[5] 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.

[6] Andrew J Barry and Russ Tedrake. "Pushbroom stereo for high-speed navigation in cluttered environments". In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 3046–3052.

[7] 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.

[8] 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.

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

[10] 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.

[11] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

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

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

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

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.

[49] 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.

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

[51] 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.

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

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

[54] 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.

[55] 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.