Autonomous Visual Mapping and Exploration With a Micro Aerial Vehicle

Lionel Heng*
Computer Vision and Geometry Lab
ETH Zürich
Universitätstrasse 6, 8092 Zürich
hengli@inf.ethz.ch

Dominik Honegger
Computer Vision and Geometry Lab
ETH Zürich
Universitätstrasse 6, 8092 Zürich
dominik.honegger@inf.ethz.ch

Gim Hee Lee
Computer Vision and Geometry Lab
ETH Zürich
Universitätstrasse 6, 8092 Zürich
glee@student.ethz.ch

Lorenz Meier
Computer Vision and Geometry Lab
ETH Zürich
Universitätstrasse 6, 8092 Zürich
lm@inf.ethz.ch

Petri Tanskanen
Computer Vision and Geometry Lab
ETH Zürich
Universitätstrasse 6, 8092 Zürich
tpetri@inf.ethz.ch

Friedrich Fraundorfer
Remote Sensing Technology
Faculty of Civil Engineering and Surveying
Technische Universität München
Arcisstrasse 21, 80333 München
friedrich.fraundorfer@tum.de

Marc Pollefeys
Computer Vision and Geometry Lab
ETH Zürich
Universitätstrasse 6, 8092 Zürich
marc.pollefeys@inf.ethz.ch

Abstract

Cameras are a natural fit for micro aerial vehicles (MAVs) due to their low weight, low power consumption, and two-dimensional field of view. However, computationally-intensive algorithms are required to infer the 3D structure of the environment from 2D image data. This requirement is made more difficult with the MAV's limited payload which only allows for one CPU board. Hence, we have to design efficient algorithms for state estimation, mapping, planning, and exploration. We implement a set of algorithms on two different vision-based MAV systems such that these algorithms enable the MAVs to map and explore unknown environments. By using both self-built and off-the-shelf systems, we show that our algorithms can be used on different platforms. All algorithms necessary for autonomous mapping and exploration run on-board the MAV. Using a front-looking stereo camera as the main sensor, we maintain a tiled octree-based 3D occupancy map. The MAV uses this map for local navigation and frontier-based exploration. In addition, we use a wall-following algorithm as an alternative exploration algorithm in open areas where frontier-based exploration underperforms. During the exploration, data is transmitted to the ground station which runs

*http://people.inf.ethz.ch/hengli

large-scale visual SLAM. We estimate the MAV's state with inertial data from an IMU together with metric velocity measurements from a custom-built optical flow sensor and pose estimates from visual odometry. We verify our approaches with experimental results, which to the best of our knowledge, demonstrate our MAVs to be the first vision-based MAVs to autonomously explore both indoor and outdoor environments.

1 Introduction

MAVs are an ideal choice for autonomous reconnaissance and surveillance because of their small size, high maneuverability, and ability to fly close to the ground in very challenging environments. To perform these tasks effectively, the MAV must be able to perform precise pose estimation, map the environment, navigate from one point to another, and plan exploration strategies. Ideally, the MAV should not depend on external positioning systems such as the Vicon motion capture system and GPS. In addition, all processes necessary for autonomy should run on-board the MAV, as a communications outage, especially in the case of Wi-Fi, can immobilize a MAV which depends on off-board processing.

Due to the popular choice of laser scanners, the use of a camera as the main sensor for MAVs has largely been overlooked. A laser scanner returns accurate 1D range measurements which are easy to process. In contrast, a monocular camera provides image data and numerous 2D features with unknown depth while a stereo camera produces a much higher number of range measurements at a considerable computational cost and with significantly higher measurement uncertainty compared to laser range measurements. Thus, processing relatively complex data from cameras requires significant computational resources, and at the same time, pose estimation and mapping are less robust. However, a camera offers several considerable advantages over a laser range finder for MAVs. The camera is a passive sensing device, is significantly lighter, and consumes less power. Most importantly, the camera as a sensor is a key enabler for a wide range of capabilities that include, but are not limited to, 6-DoF pose estimation, 3D mapping, dynamic object tracking, semantic object labeling, and so on. These advantages of a camera are the major motivation for our work on vision-based mapping and exploration for MAVs; to the best of our knowledge, there is no published work on vision-based exploration with MAVs.

Typically, MAVs that rely on a laser scanner are unable to localize in straight corridors in indoor environments. As a result, such MAVs are unable to do exploration with a metric map. A laser scan taken at any point along a straight corridor would simply show two straight lines; there are no unique structural features that are required for laser scan matching. We show through an indoor corridor experiment that vision-based MAVs have an advantage over laser-based MAVs by demonstrating the ability to explore corridors.

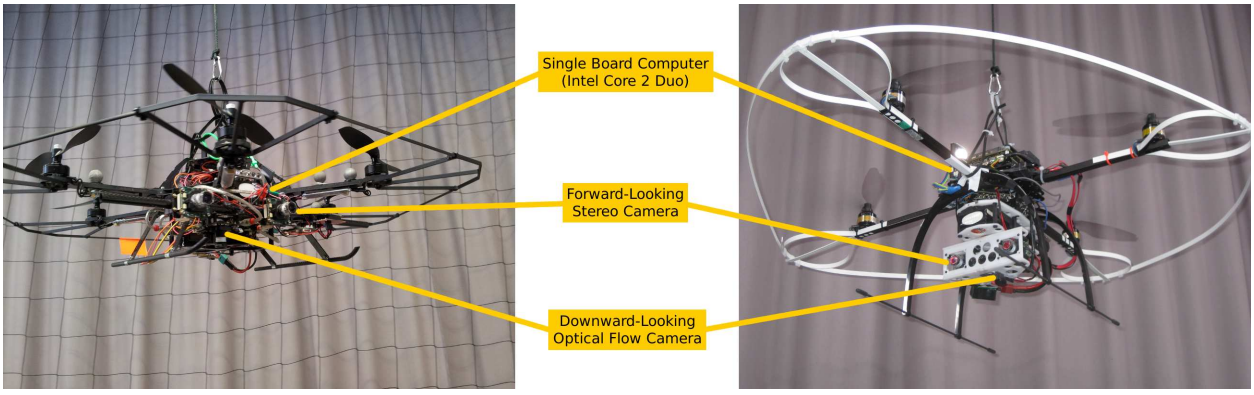


Figure 1: The AscTec Firefly (left) and PIXHAWK Cheetah (right).

The main objective of this paper is to demonstrate the feasibility of autonomous mapping and exploration

for a MAV with cameras as its main sensors. This paper expands on our previous paper (Fraundorfer et al., 2012) by making significant improvements to the mapping algorithm, using an optical flow sensor with a CMOS image sensor instead of a mouse sensor to provide reliable metric velocity estimates in low-light and low-texture environments, and showing more experimental results with an off-the-shelf hexacopter MAV in addition to our custom-built PIXHAWK quadrotor MAV. Improvements to the mapping algorithm include an inverse sensor measurement model appropriate for a stereo camera, pyramidal ray casting, and the use of a tiled octree-based map that allows the mapping to have constant space complexity.

We show our two MAV systems in Figure 1. Both systems are fitted with a single board computer which has a Intel Core 2 Duo processor and 2 GB RAM. In addition, both MAVs have a custom-designed downward-looking optical flow sensor with a built-in CMOS image sensor and an ARM Cortex-M4F microcontroller (Honegger et al., 2013). Each MAV incrementally builds a 3D occupancy map. Using this map, the MAV moves autonomously from one point to another using the VFH+ algorithm (Ulrich and Borenstein, 1998). It plans exploration strategies based on the frontier-based exploration algorithm (Yamauchi, 1997) where frontiers are selected from a horizontal slice of the 3D occupancy map which is constructed on-board. The frontier-based exploration algorithm works very well in cluttered environments where distinctive frontiers can be identified. The algorithm, however, under-performs in open areas where clearly defined frontiers are absent. This is because in open areas, the disparity map computed from stereo is very sparse, and thus, very few measurements are available. Unlike laser measurements, free space can only be inferred from disparity measurements that correspond to obstacles. As a remedy, we implement the Bug algorithm (Choset et al., 2005) for autonomous wall-following which can optionally be selected as the substitute exploration algorithm in cases where the frontier-based exploration under-performs. We manually choose either the frontier-based exploration or the Bug-based exploration depending on the environment which the MAV is expected to explore. Images from the front-looking camera are transmitted together with data from the optical flow sensor and visual odometry to the ground station where large scale SLAM is done off-board with pose graph optimization. We use the g2o (Kümmerle et al., 2011) implementation for SLAM. Each vertex in the graph represents the MAV's pose, and each edge represents the constraints between each MAV's pose obtained from the visual pose estimation. Additional edge constraints are also added from loop-closure detections in the camera images via the vocabulary tree (Nistér and Stewénius, 2006; Fraundorfer et al., 2007).

We summarize the sections of this paper as follows. Section 2 discusses related work. Section 3 shows the hardware and software design of our two MAV systems. The state estimation for the MAV is described in Section 4. In Section 5, we describe our mapping implementation in detail, and likewise in Section 6, we describe our planning and exploration algorithms. The off-board SLAM is discussed in Section 7. Lastly, we show the experimental results in Section 8, and discuss our results in Section 9.

2 Related Work

There is a considerable number of works on the use of a laser scanner to achieve MAV autonomy. The main reason for the choice of a laser scanner over a stereo camera is that range measurements from a laser scanner are much more accurate and reliable compared to those from a stereo camera. Hence, the use of a laser scanner leads to control, pose estimation, mapping, and navigation being more reliable. Laser scan matching is typically employed in relative motion estimation and finding loop closure candidates. A mirror is often used to redirect downwards a few laser beams from a horizontally-mounted laser range finder so that the MAV can estimate the distance to the ground. Likewise, a few laser beams are reflected upwards for estimation of the distance to the ceiling. Current real-time 3D SLAM implementations for computationally constrained platforms do not scale to large environments, and hence, either a 2D SLAM or a pose graph SLAM implementation is used. (Grzonka et al., 2012) runs a 4-DoF pose graph SLAM implementation with all modules except the device drivers run off-board on a ground station. Similarly, (Dryanovski et al., 2013) runs a 2D SLAM implementation (GMapping) with all modules running on-board. (Bachrach et al., 2011) also runs the GMapping implementation and a similar planner, and demonstrates indoor exploration with a 2D map. However, all modules except the state estimation run off-board on a ground station. (Shen et al.,

2012) advances the state-of-art for laser-based MAVs by maintaining a 3D occupancy map and 6-DOF pose graph which is optimized by an iterative EKF, showing 3D indoor exploration and running all processes on-board. In addition, they use a Kinect for mapping, and a camera and vocabulary tree to detect loop closures in the pose graph. Despite the close similarities to our work, we highlight one important difference: we use cameras whereas the described works use a laser scanner as the main sensor. By using cameras, we have to deal with a significantly higher level of measurement noise, and allocate much more computational resources to building a map from image data. The remaining computational resources have to be carefully managed, and suitable planning and exploration algorithms with low computational requirements have to be selected so that autonomous exploration can take place. At the same time, it is not possible to use a stereo camera with a limited field of view to estimate the distance to the floor and ceiling. Limited computational resources do not allow for an additional downward or upward-looking stereo camera. Hence, we develop our custom-built downward-looking optical flow sensor with its own on-board processing to provide ground distance measurements and precise metric velocity measurements.

Artificial-marker-aided vision-based flights are demonstrated in (Eberli et al., 2011; Meier et al., 2012). (Eberli et al., 2011) shows autonomous takeoff, landing, and hovering using one circular artificial marker. This marker is detected in images from a monocular camera, and used to estimate the pose of the MAV. The pose data is fed back to a set-point controller in order to achieve autonomous takeoff, landing, and hovering. (Meier et al., 2012) pushed the technology further by demonstrating autonomous takeoff, landing, hovering, and waypoint following over a much larger space using numerous ARToolKitPlus (Wagner and Schmalstieg, 2007) markers laid on the ground. Their MAV is equipped with a single downward-looking camera which detects these markers, and computes the MAV's pose. We note that the autonomy in both works is largely restricted to the area covered by the artificial markers. (Heng et al., 2011) shows autonomous occupancy grid mapping and global path planning on a MAV. The poses of the MAV are obtained from the Vicon system, thus restricting the autonomy of the MAV to the Vicon space.

MAVs that rely on a single downward-looking camera for pose estimation have demonstrated landing, takeoff, hovering, and mapping in (Blösch et al., 2010; Achtelik et al., 2011). However, they cannot demonstrate other autonomous behaviors such as path planning and exploration because the downward-looking camera is not able to perceive the portion of the environment in front of the MAV. In contrast, we use both a forward-looking stereo camera and a downward-looking optical flow sensor. Consequently, this allows our MAVs to have a better perception of the environment, and thus, the ability to do autonomous path planning and exploration.

(Shen et al., 2013) uses a MAV with two forward-looking cameras for pose estimation: a fisheye camera, and a normal camera. The fisheye camera is used to track features and maintain a local sparse map which comprises currently tracked features. The scale of the map is maintained via stereo correspondences computed with the two cameras. 2D-3D correspondences are used to estimate the pose of the MAV. However, they do not demonstrate mapping, planning, and exploration. (Bills et al., 2011) implements an approach for higher level navigation using a Parrot AR.Drone. Utilizing image classification, the MAV is able to follow corridors, and even make turns, but no notion of a metric map is involved. Similarly, (Mori and Scherer, 2013) uses a Parrot AR.Drone for local obstacle avoidance with a forward-looking monocular camera, but demonstrates neither mapping nor exploration.

(Schmid et al., 2013) demonstrates the most advanced vision-based MAV so far; they show mapping and path planning on a MAV with a forward-looking stereo camera. The computationally-expensive semi-global matching is outsourced to a low-power FPGA board. However, they do not show exploration. Due to the need to transfer images and depth maps to and from the FPGA board respectively, the latency associated with the poses estimated from visual odometry is significantly higher than that in our approach which computes depth maps and visual odometry poses together on the same single board computer. This lower latency associated with the pose estimates from visual odometry together with the extremely high frequency of velocity estimates from optical flow provided at 250 Hz enable the control system on our MAVs to be more responsive. Furthermore, in texture-poor environments, optical flow still works, but visual odometry may fail. Dependence on the optical flow sensor allows our MAVs to fly in texture-poor environments. However,

our use of the optical flow sensor only allows for flights in environments with relatively flat ground.

3 Our MAV Platforms

In this section, we give a brief description of the hardware and software designs of our MAV platforms which are shown in Figure 1. Both MAVs are designed for self-contained autonomous flight with all processes running on-board. Figure 2 shows a schematic of the hardware and software components for each platform. Both platforms weigh approximately 1.5 kg, and have a maximum payload of 600 g.

Both MAVs are equipped with a downward-looking optical flow sensor. This sensor uses a built-in ARM Cortex-M4F microcontroller to compute optical flow estimates, and an ultrasonic range sensor to scale optical flow estimates to metric velocity estimates which are then provided to the state estimator. Both MAVs also have a front-looking stereo rig; the rig contains two Matrix Vision mvBlueFOX cameras with an image resolution of 752×480 pixels. The stereo baselines for the AscTec Firefly and PIXHAWK Cheetah are 80 mm and 120 mm respectively. The stereo process computes rectified images and depth maps from each stereo image pair. Both data streams are used by the visual odometry (see Section 4.2 for more details) while the 3D mapper only uses the depth maps to incrementally build a 3D occupancy map. The exploration module uses the computed 3D map for autonomous behaviors such as local navigation and exploration. The exploration module selects new waypoints and passes these waypoints to the position controller which moves the MAV from its current location to the currently commanded waypoint. The stereo processing, visual odometry, mapping, and exploration modules run on-board the single board computer.

The main differences between the two platforms are the controller and state estimator implementations shown as blocks with red dotted outlines in Figure 2. The controller implementation on the AscTec Firefly is provided by AscTec. We implement and modify the EKF-based sensor fusion framework (Weiss et al., 2012) to fuse data from the IMU, magnetometer, optical flow sensor, and visual odometry. We disabled the self-calibration feature which requires the MAV to be in an excited state most of the time, and therefore, is not accurate when the MAV is flying slowly and at a fixed altitude all the time. We only use the self-calibration feature to obtain a one-time estimate of the inter-sensor transforms by manually moving the MAV with large acceleration and orientation changes. On the PIXHAWK Cheetah, we use the pxIMU hardware which executes state estimation, and position and attitude control on an ARM7 microcontroller in hard real-time. Details on the state estimation and control for the PIXHAWK Cheetah can be found in (Meier et al., 2012). Note that two separate microcontrollers are used for the optical flow and control. The pxIMU hardware is used on both platforms to synchronize and timestamp all on-board sensors including images from the cameras. This hardware synchronization allows us to fuse outputs from different sensors and correctly estimate sensor delays.

The outputs from the stereo process, visual odometry, and state estimation are transmitted to the ground station where pose graph SLAM and loop-closure detection run off-board. We are unable to run this pose graph SLAM on our MAVs due to the lack of computational resources. Furthermore, we do not use the global pose estimates from pose graph SLAM. The reason is that the Wi-Fi connection between the ground station and MAV breaks down intermittently, and each time after the Wi-Fi connection comes back online, the first global pose estimate that the MAV receives significantly differs from the current pose estimate from the state estimator, resulting in a pose jump, and causing the map to be distorted. In the end, exploration is rendered infeasible. However, with recent advances in computing hardware technology, it is only a matter of time before we can run pose graph SLAM on-board, and therefore, we show results from our off-board pose graph SLAM.

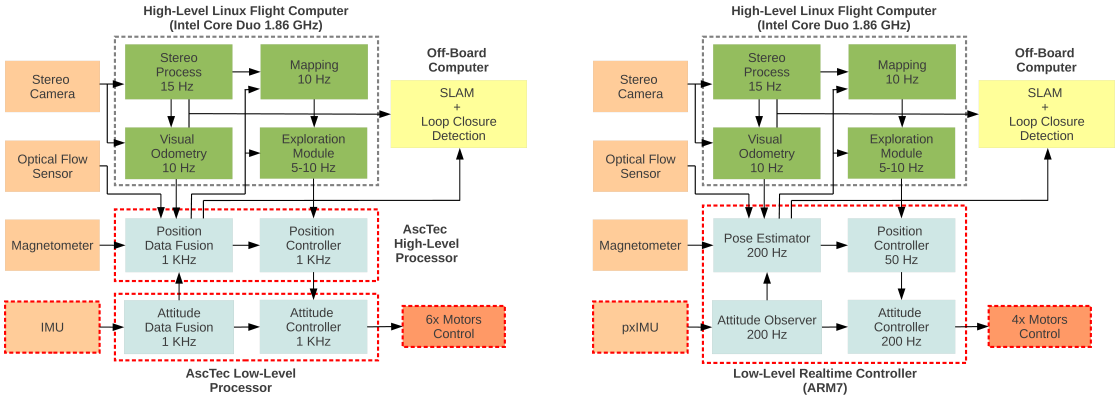


Figure 2: System diagram for the AscTec Firefly (left) and PIXHAWK Cheetah (right). The green blocks mark system components that run on-board the MAV. The blue blocks mark system components which are implemented on microcontrollers. The red dotted outlines enclose the system components that differ between the two MAV systems.

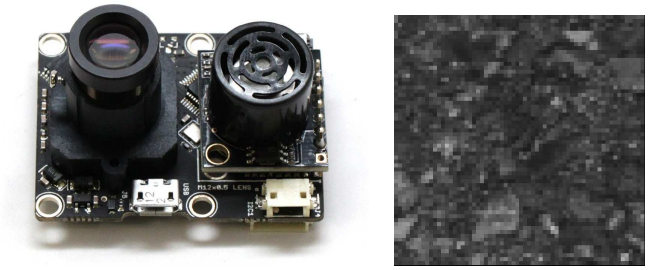


Figure 3: The PX4FLOW optical flow sensor and typical street texture as seen through a 16 mm M12 lens by the flow sensor at an altitude of 0.8 m.

4 State Estimation

In this section, we describe the estimation processes for the metric velocity from optical flow, and pose from visual odometry. We also describe the fusion of the metric velocity measurements from optical flow and pose measurements from visual odometry together with data from the IMU and magnetometer to estimate the full state of the MAV.

4.1 Metric Velocity Estimates from Optical Flow

Figure 3 shows our optical flow sensor together with a typical outdoor image captured by the integrated image sensor. The optical flow sensor estimates the optical flow between two consecutive frames at 250 Hz via block matching based on the sum of absolute differences (SAD). Subsequently, the distance reading from the ultrasonic range sensor is used to scale the optical flow to metric velocity in the sensor frame. For more details on how the optical flow sensor computes the metric velocity estimates, refer to (Honegger et al., 2013). Transformation of the metric velocity estimates from the optical flow sensor frame to the MAV's body frame requires angular rate compensation, which in turn, requires knowledge of the MAV state estimate. Thus, angular rate compensation is performed in the state estimation process.

We run two experiments to demonstrate the accuracy of the velocity measurements from the optical flow sensor. In the first experiment, we repeatedly rotate the optical flow sensor around the vertical axis, and plot in Figure 4 both the rotational component of the velocity estimates from the optical flow sensor and the angular velocity measurements from the gyroscope that is located on the optical flow sensor and aligned with the image sensor. The RMSE of the rotational velocity estimates from the optical flow sensor with respect to the gyroscope measurements is 0.192 rad/s. In the second experiment, the MAV moves twice along a square-shaped trajectory which starts and ends at the same point. To obtain the position of the MAV, we do a simple integration of the linear component of the velocity estimates from the optical flow sensor over time; the linear component is obtained by applying angular rate compensation to the velocity estimates from the optical flow sensor. We plot the estimated position of the MAV in figure 5. We observe a position error of 52 cm between the start and end points over a trajectory of 32 m; the resulting loop closure error is 1.63%. This position error is mainly caused by small errors in the velocity estimates from the optical flow sensor that accumulate over time, and the limited accuracy of the magnetometer and accelerometer data used in the attitude estimation by the IMU and which angular rate compensation depends on. Both experiments show the accuracy of the velocity measurements from the optical flow sensor.

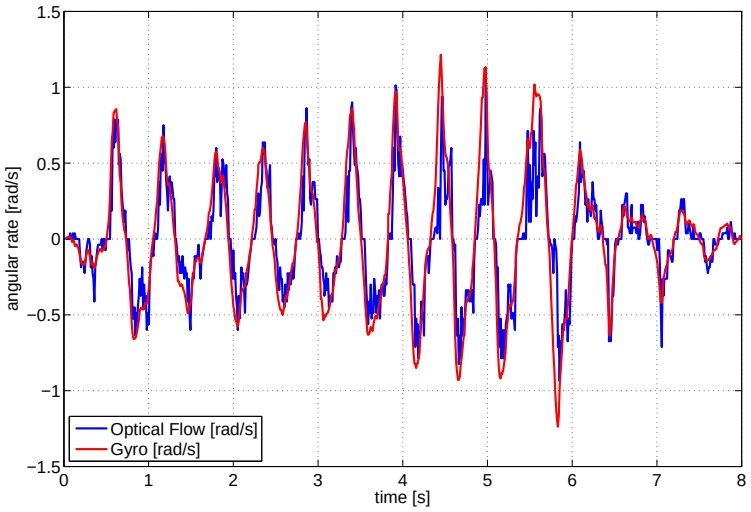


Figure 4: A comparison is shown between angular velocity measurements from the optical flow sensor and gyroscope during a rotation around the vertical axis. Both measurements are shown to coincide closely.

One limitation of the optical flow sensor is that velocity measurements are only available as long as the MAV is at most 5 m above the ground. This is because the maximum range of the ultrasonic distance sensor is 5 m. Another limitation is the maximum velocity of the MAV. The maximum velocity v_{max} that can be measured with the optical flow sensor using the SAD algorithm with a search window of size w pixels and at x Hz can be calculated:

$$v_{max} = \frac{whx}{f} \tag{1}$$

where f is the focal length, and h is the height of the optical flow sensor above the ground. In our experiments, the MAV typically flies 1 m above the ground, and we use the following values: $w = 4$, $x = 250$, and $f = 666$. Hence, the maximum velocity of the MAV that the optical flow sensor works up to is 1.5 m/s. The use of the optical flow sensor introduces the assumption that the ground over which the MAV flies is locally flat, and thus, the MAV is restricted to flights over relatively flat ground.

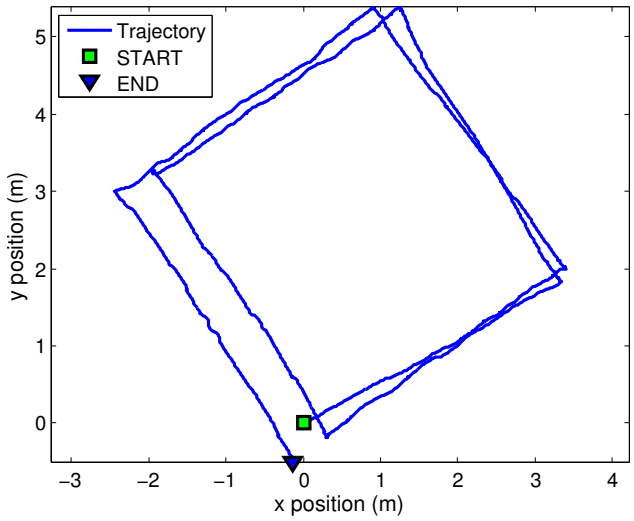


Figure 5: Linear velocity measurements from the optical flow sensor are integrated over time as the MAV moves twice along a square-shaped trajectory which starts and ends at the same point. The green square marks the estimated position of the MAV at the beginning of the trajectory, and the blue triangle marks the estimated position of the MAV at the end of the trajectory.

4.2 Pose Estimates from Visual Odometry

We use the front-looking stereo camera to compute the visual odometry that estimates the absolute 6-DOF pose of the MAV. We do not compute the pose covariance here. As mentioned earlier, the visual odometry runs on the single board computer.

We wish to minimize visual odometry drift so we use keyframe-based visual odometry as described in (Scaramuzza and Fraundorfer, 2011). To limit drift, we maintain a reference frame and compute the poses of the subsequent frames by localization with respect to the reference frame. From the left cameras of both the reference and current frames, we first extract FAST corners (Rosten et al., 2010), compute their BRIEF descriptors (Calonder et al., 2012), and find matches between the two frames. Here, the reference frame includes the rectified image from the left camera, and depth map from the stereo process. The current frame only consists of the image from the left camera. The 3D points are computed from the depth map of the reference frame. Next, we get the 2D-3D correspondences between the current frame and the reference frame from 2D-2D matches of the image features. Lastly, we compute the current pose with RANSAC PnP (Fischler and Bolles, 1981) followed by non-linear refinement with a robust cost function (Hartley and Zisserman, 2004). We do not use the depth map for the current frame because (Nistér et al., 2004) shows that optimization of the reprojection error in 2D-3D matching provides more accuracy than optimization of the 3D-3D error in 3D-3D matching. Furthermore, we do not use the image from the right camera for both the reference and current frames as the depth map is already available for the reference frame, and we avoid extra computations incurred from matching features between the left and right images and triangulating the matched features.

We select a new reference frame based on the following conditions:

- The number of geometric inliers of the feature correspondences has to be above a threshold value.
- Either the Euclidean or angular distance measured between the existing reference frame and the current frame exceeds a threshold value.

In cases where there are no sufficient matches, we do not compute the pose. If this persists over a certain

period of time, we replace the reference frame with the current frame. The advantage of using reference frames as compared to consecutive frame-to-frame matching is that it is less susceptible to drift. This is because we compute the current pose by localizing the current frame directly on the reference frame instead of concatenating frame-to-frame relative pose estimates which increases the chance for errors to accumulate. In particular, our key-frame visual odometry helps to stabilize the MAV during hovering by eliminating any possible local drift.

4.3 Sensor Fusion

Sensor fusion is required to fuse measurement data from multiple sources so that we obtain a coherent state estimate. We use different sensor fusion approaches for our MAV platforms. For the Asctec Firefly, we use the EKF-based sensor fusion framework from (Weiss et al., 2012). This framework is distributed in the sense that the state prediction runs on the AscTec Firefly's autopilot, while the state update runs on the single board computer. The state update treats data from the magnetometer, optical flow sensor, and visual odometry as three separate measurements which are used to update the state. In contrast, the state estimation approach on the PIXHAWK Cheetah (Meier et al., 2012) differs in two aspects: both the state prediction and estimation run on the pxIMU hardware, and due to the computational limitations of the pxIMU hardware, we use a simple Kalman filter. However, we show in our experimental results in Section 8 that the state estimates from these two platforms are comparable in accuracy. More details on the state estimation implementations for the AscTec Firefly and PIXHAWK Cheetah can be obtained from (Weiss et al., 2012) and (Meier et al., 2012) respectively.

It is well-known that the ultrasonic distance sensor suffers from spikes in distance measurements due to specular reflectance and acoustic noise. We plot the distance measurements from the ultrasonic distance sensor located on the downward-looking optical flow sensor, and the z-component of the state estimates in Figure 6. There is a constant offset between the two plots as the optical flow sensor is located below the IMU whose reference frame coincides with the body reference frame. The figure shows that the state estimator gracefully handles the spikes in the distance measurements from the ultrasonic distance sensor. As a result, to maximize the robustness of the state estimator, we use the estimated z-component of the estimated state instead of the ultrasonic distance measurement to scale the optical flow measurements to metric velocity measurements.

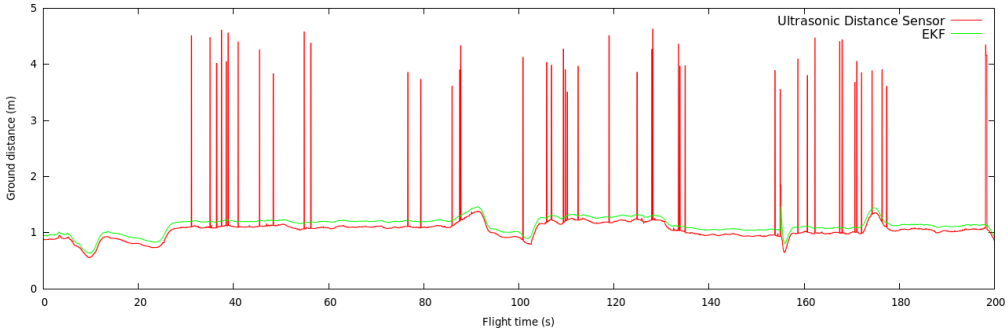


Figure 6: The ultrasonic distance measurements are plotted with the z-estimates from the state estimator over time. The state estimator successfully rejects outlier measurements from the ultrasonic distance sensor.

5 Mapping

An accurate 3D occupancy map is critical for successful planning and exploration. At the same time, the map has to be updated at a sufficiently high frequency so that the planning module can react in time to unforeseen obstacles. Our occupancy map implementation is based on an octree with dynamic tile caching which requires a constant amount of memory regardless of the size of the area being mapped. With our

occupancy map implementation, MAVs with limited memory resources can explore large areas as long as sufficient hard disk space is available to store unused tiles. We use an inverse sensor measurement model based on a stereo camera to update the 3D occupancy map. We make the map update possible at a high frequency and in real-time via the use of a modified version of an efficient ray traversal technique (Revelles et al., 2000) and a pyramidal line drawing technique (Andert, 2009) which reduces the number of cells in the 3D occupancy map to be updated.

5.1 Data Structure

We choose the data structure for the 3D occupancy map to be an octree; any environment in which the MAV operates is always sparse in the sense that the size of unknown areas is much larger than the size of explored areas, and in this case, an octree data structure makes much more efficient use of memory in contrast to a regular 3D occupancy grid as the octree data structure only stores occupancy cells corresponding to areas that have been explored. However, an octree-based 3D occupancy map grows over time as the area explored by the MAV increases. At some point, the map runs out of memory, causing a catastrophic failure. To solve this issue, we implement a tiled octree-based 3D occupancy map with dynamic tile caching; the occupancy map is a rolling tiled map that is continually centered on the MAV. The tiles that currently make up the current map are stored in a octree data structure in memory, while all other tiles are stored on disk.

In contrast, in the widely-used OctoMap implementation (Hornung et al., 2013) which we used in our previous papers, the size of the 3D occupancy map increases over time with no bounds on the maximum size of the map. Furthermore, OctoMap does not allow us to efficiently implement an inverse sensor measurement model for a stereo camera and pyramidal line drawing (Andert, 2009). (Revelles et al., 2000) shows that the bottom-up 3D-DDA ray traversal technique used by OctoMap is less efficient than the top-down ray traversal technique we use in our 3D occupancy map.

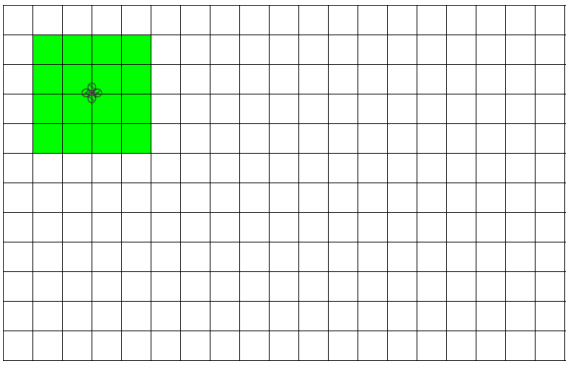
In our implementation, the 3D occupancy map has equal dimensions and has resolution r. We use a map resolution of 0.25 m for both indoor and outdoor environments. The width of the map is 4 tiles, each of which corresponds to a octree with height:

$$h = \left\lceil \log_2 \frac{R}{r} \right\rceil + 1 \tag{2}$$

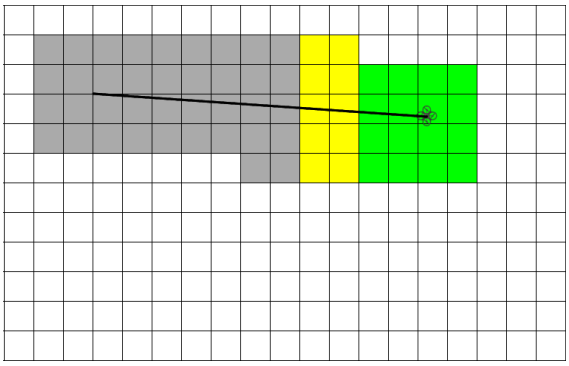
where R is the maximum range of the stereo camera. In this way, we ensure that regardless of the location of the MAV, any occupancy cell to be updated does not lie outside the map. Each cell stores a log-odds value which reflects the occupancy probability. Intuitively, we model the occupancy map as an octree whose height is $h + 2$ as an octree is memory-efficient and is able to model 3D environments with high levels of detail. The occupancy map is a rolling map that moves with the MAV, and which only contains tiles within the MAV's vicinity. All other tiles that have been initialized and are no longer within the map boundary are saved to disk, and reloaded when the MAV returns to the same area in future. To minimize the latency effects associated with disk I/O operations, a separate thread caches all existing tiles that are outside the map, but are within the boundary of a hypothetical octree of height $h + 3$ with the same center as the occupancy map. With this data structure design, the memory required by the map has an upper bound which is in the order of tens of megabytes in our case. For purposes of clarity, we use Figure 7 to explain how our map implementation would work in 2D space.

5.2 Inverse Sensor Measurement Model

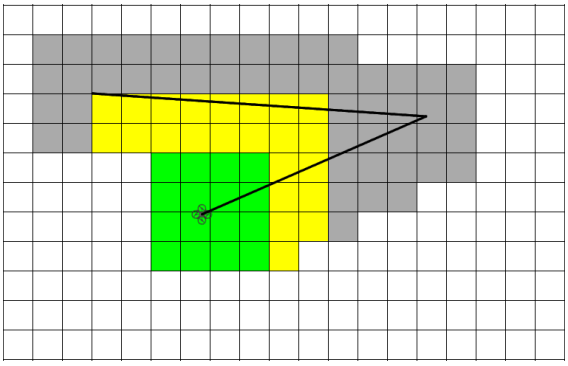
An inverse sensor measurement model maps the measurement from a sensor to an occupancy map; this occupancy map specifies at every point in space the probability that the 3D point inferred from the sensor measurement exists at that point. In our case, for purposes of computational efficiency and tractability, this occupancy map is a 1D map that spans a ray with its endpoint located at the sensor, and which intersects the 3D point associated with the sensor measurement.



(a) At $t = 0$, during the map initialization, 16 tiles marked as green squares are created to form the current map. Each tile is an octree with height h, and the current map is an octree with height $h + 2$.



(b) At $t = t_1$, the MAV has flown to a new position, and as the MAV flies, the map is recentered. The yellow squares represent tiles that have been created as part of the map before, and are now cached as they are considered to lie outside but near the current map boundary. The grey squares represent tiles that have been created as part of the map before, and are now stored on disk, as they are far away from the current map boundary.



(c) At $t = t_2$, the MAV has again flown to a new position. Note that only at most 64 tiles are stored in memory at any time; 16 tiles form the map, and at most 48 tiles are cached and ready to be loaded once the map recenters.

Figure 7: We illustrate how our tiled octree-based occupancy map with dynamic tile caching works in 2D space.

For a stereo camera, given a disparity measurement d at pixel coordinate (i,j) in the disparity map, the coordinates of the corresponding 3D point relative to the camera coordinate system are:

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix} = \begin{bmatrix} \frac{i-c_x}{f} \\ \frac{j-c_y}{f} \\ \frac{bf}{d} \end{bmatrix} \tag{3}$$

where b, (c_x,c_y), and f are the baseline, coordinates of the principal point, and focal length of the camera respectively.

Differentiating the 3rd row of equation 3, we get:

$$\Delta z_{cam} = \frac{bf}{d^2} \Delta d \tag{4}$$

Δz_{cam} denotes the uncertainty of the range measurement corresponding to Δd, the uncertainty of the disparity measurement d. We use the OpenCV stereo block matching implementation that computes sub-pixel disparity measurements with a resolution of $\frac{1}{16}$ pixels. However, we choose a conservative value of $\Delta d = 0.5$ here.

Equation 4 tells us that the smaller the disparity measurement, the larger the measured distance of the associated 3D point, and the larger the uncertainty of the range measurement. Correspondingly, the occupancy probability of a cell located at that 3D point is lower. Hence, we design the inverse sensor measurement model for a stereo camera such that the occupancy probability is a function of the distance from the camera, and this function is based on the Gaussian function with the following properties:

Property I. The peak is centered on the range measurement and its height is a decreasing exponential function of the range measurement resolution.

Property II. The width is controlled by the range measurement resolution.

Property III. On the left side of the peak, the minimum of the function equals the occupancy probability of free space, p_{free}, and on the right side of the peak, the minimum of the function equals the occupancy probability of unknown space, $p_{unknown}$.

We parameterize the model as a piecewise-defined function based on two Gaussian functions due to property III:

$$p(r, r_p) = \begin{cases} p_{free} + (a + p_{unknown} - p_{free}) e^{-\frac{1}{2}\left(\frac{r-r_p}{\Delta r_p}\right)^2} & \text{if } 0 < r \leq r_p \\ p_{unknown} + a e^{-\frac{1}{2}\left(\frac{r-r_p}{c}\right)^2} & \text{if } r > r_p \end{cases} \tag{5}$$

Given a distance measurement r_p and distance r, $p(r)$ is the occupancy probability at that distance. $r_p = \sqrt{x_p^2 + y_p^2 + z_p^2}$ is the distance to the measured point (x_p, y_p, z_p). $\Delta r_p = \frac{r_p^2}{bf} \Delta d$ is the uncertainty of the measured distance with $\Delta d = 0.5$ as explained earlier. a and c are two parameters which control the height and width of the function respectively. The inclusion of the left term of each subfunction satisfies property III. The use of the Gaussian function in the right term of each subfunction and whose width is dependent on Δr_p satisfies property II.

To satisfy property I, the value of a is computed as:

$$a = k(1 - p_{unknown}) e^{-\Delta r_p} \tag{6}$$

where $k \leq 1$ is a weight indicating how significant the range measurement is.

We choose the value of c such that both subfunctions are approximately symmetric around the peak. We make use of the fact that c is related to w, the full width at x percent of maximum of the peak of the second subfunction via the equation:

$$w = 2\sqrt{2\ln(100/x)}c \tag{7}$$

where $0 \leq x \leq 100$. Setting the width of the second subfunction to be the same as that of the first subfunction at the point of half maximum of the peak of the second Gaussian function, we compute c:

$$c = \sqrt{\frac{\ln(1 + \frac{a}{a+1-2p_{min}})}{\ln 2}}\Delta r_p \tag{8}$$

For the model in (Andert, 2009), the occupancy probability distribution is asymmetrical around the peak; this asymmetry gets more pronounced as the distance measurement increases. This asymmetry property is not ideal since two points located near the peak and at either side of the peak should have the same occupancy probability. Furthermore, as $r_p \to 0$, $p(r, r_p) \to \infty$ which violates the constraint that a occupancy probability value must be between 0 and 1. In contrast, for our model, the occupancy probability distribution is approximately symmetrical around the peak, and is bounded between $p_{free} \geq 0$ and 1.

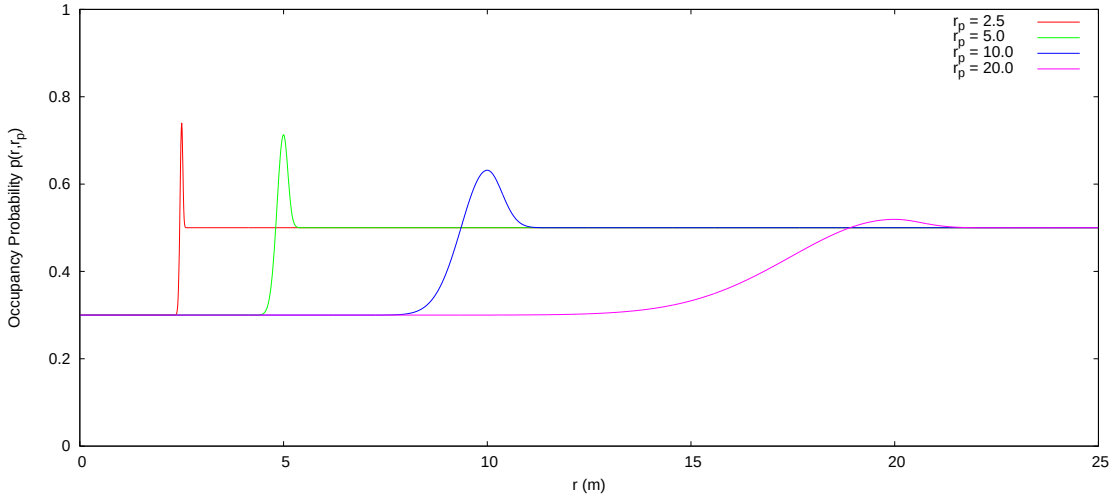


Figure 8: Inverse sensor measurement model for a stereo camera with $b = 0.2$, $f = 650$, $k = 1$, $p_{free} = 0.3$, and $p_{unknown} = 0.5$.

Figure 8 shows the output from the inverse sensor measurement model for varying values of r_p. For computational efficiency, we use a look-up table for the inverse sensor measurement model.

5.3 Ray Traversal

(Revelles et al., 2000) details an algorithm for octree traversal of a ray with an origin, unit direction vector, and infinite length. However, we do not want to unnecessarily update occupancy cells that lie behind the observed 3D points. Hence, we modify the algorithm such that we traverse a line segment with two endpoints in the map which is an octree with height $h + 2$.

The ray traversal algorithm also computes all points at which the ray intersects the cell boundaries. Using this information, we compute the measurement probability for each cell by choosing the maximum of the probability distribution between the point at which the ray enters the cell, and the point at which the ray leaves the cell.

5.4 Pyramidal Line Drawing

Since the point cloud from a stereo frame potentially contains up to ∼350000 points, it is inefficient to directly use each point in the point cloud to update a 3D occupancy map. Instead, we use the pyramidal line drawing technique (Andert, 2009) to significantly reduce the number of occupancy grid cells that have to be updated.

If the 3D points associated with neighboring pixels in the depth map are closer to the camera, it is more likely that these points could fall in the same occupancy cell, and in this case, it is not efficient to update the same occupancy cell using all these measured 3D points. Pyramidal line drawing solves this problem. First, we create a pyramid of depth maps in which the original depth map is stored at the minimal level and is successively downsampled as we go up the pyramid. Denoting the original depth image as $d_0(x, y) = d(x, y)$, we generate successive downsampled depth images:

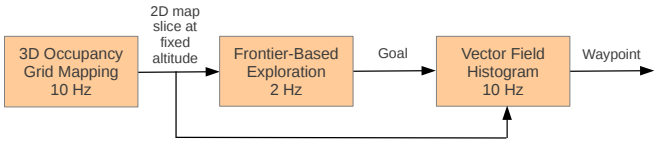
$$d_{i+1}(x, y) = \max\{d_i(2x, 2y), d_i(2x + 1, 2y), d_i(2x, 2y + 1), d_i(2x + 1, 2y + 1)\} \tag{9}$$

We start at the maximal pyramid level. Given a pixel value $d_i(x, y)$, if $\Delta p = \frac{d_i(x,y)}{\frac{f}{2^i}}$ is less than the map resolution, we traverse the line from the camera to the associated 3D point. Otherwise, we try to trace the lines for the pixels $d_{i-1}(2x, 2y)$, $d_{i-1}(2x + 1, 2y)$, $d_{i-1}(2x, 2y + 1)$, and $d_{i-1}(2x + 1, 2y + 1)$. We stop the recursion at the minimal pyramid level where lines are always traversed.

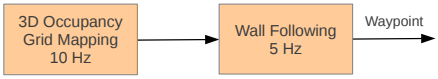
In our experiments, we observe that the map update with the pyramidal line drawing technique typically takes a single-digit percentage of the time required for the map update with the technique disabled.

6 Planning and Exploration

The MAV makes use of the current state estimate and the 3D map to do path planning and exploration. We use two different exploration strategies depending on the type of environment that the MAV operates in: frontier-based exploration and wall-following exploration.



(a) Frontier-based exploration module for environments densely populated with objects.



(b) Wall-following exploration module for open areas and environments sparsely populated with objects.

Figure 9: Two different exploration modules for specific environments.

Figure 9(a) shows the sub-components of the frontier-based exploration module for environments which are densely populated with objects. In these environments, frontiers are clearly defined. Autonomous goal selection is done by the frontier-based exploration which makes the selection based on a 2D slice of the current occupancy map at a fixed altitude. The MAV follows a sequence of waypoints chosen by the VFH+ algorithm. Figure 9(b) shows the sub-components of the wall-following exploration module for environments

which are sparsely populated with objects, for example, mostly open areas. In these environments, frontiers are absent, and thus, frontier-based exploration does not work. In this case, the Bug algorithm for wall-following replaces the frontier-based exploration and VFH+ algorithm, and generates waypoints at a fixed altitude. The selection of the exploration algorithm is done manually in our current implementation.

6.1 Exploration

6.1.1 Frontier-Based Exploration

We build on top of our state estimation (Section 4) and mapping (Section 5) algorithms the capability to autonomously explore and map an unknown environment. This essentially turns our mapping algorithm into active mapping (Makarenko et al., 2002) where waypoints are selected autonomously based on the exploration strategy. We use the frontier-based exploration (Yamauchi, 1997) algorithm as our exploration strategy, and we set a fixed altitude for the MAV to fly at during the exploration process.

Exploration begins at an arbitrary location in an unknown environment. The frontier behind the MAV is designated as the home frontier, and the MAV goes to as many frontiers as possible until it reaches the home frontier and lands at that frontier. Initially, the MAV extracts a 2D slice of the 3D occupancy grid map based on its fixed altitude. Each grid cell in the map is classified as either occupied, free, or unknown based on its occupancy probability. A grid cell is labeled as a frontier cell if the cell is classified as free and has at least 1 neighbor classified as unknown. These frontier cells are then clustered into frontiers via connected-component labeling. Both frontiers whose cell count is below a threshold, and inaccessible frontiers are removed from the frontier set. A flood-fill algorithm is initialized at the MAV's current position, and frontiers which contain cells not labeled by the flood-fill algorithm are determined as inaccessible. The remaining frontiers are the regions where the MAV should fly to in order to maximize additional information about the environment. The centroid (C_x^j, C_y^j) of the j^{th} frontier in the map is computed as:

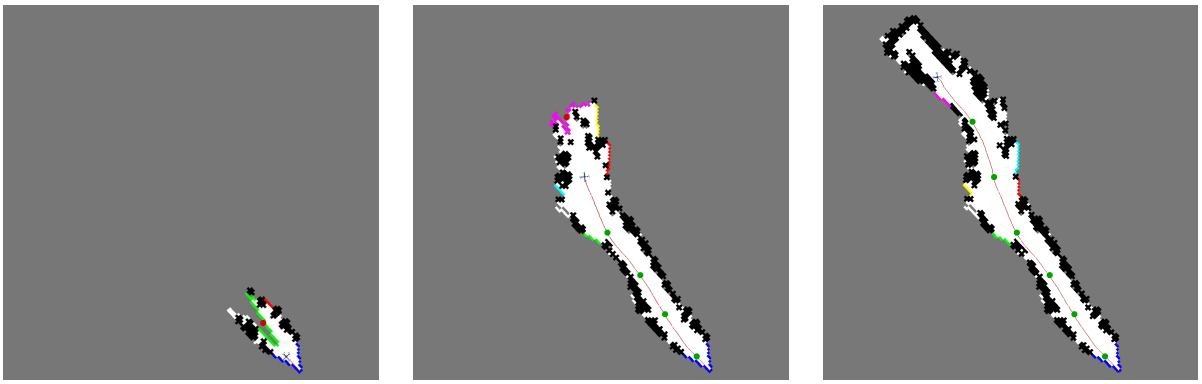
$$\begin{aligned} C_x^j &= \frac{1}{n}\sum_{i=0}^{n} x_i^j \\ C_y^j &= \frac{1}{n}\sum_{i=0}^{n} y_i^j \end{aligned} \tag{10}$$

The MAV selects the centroid that is closest to its current location as its next desired waypoint. The desired heading κ is set to be the mean of the heading of the frontier cells with respect to the centroid. The MAV then flies to the desired waypoint using VFH+, and along the way, the occupancy map expands based on successive stereo images. Once the MAV either reaches the desired frontier or cannot reach that frontier within a certain period of time, the MAV selects another frontier to go to. Exploration continues until the home frontier is the only frontier left in the map, and the MAV lands at the home frontier.

Figure 10 shows the occupancy maps generated during an exploration run. Figure 10a shows the MAV at the starting position. It builds a local occupancy map in which two frontiers (represented by green and blue cells) are detected. The MAV selects the nearest frontier (represented by a red dot) and flies towards that frontier using the VFH+ algorithm. Once the MAV reaches its destination, it retrieves the latest occupancy map, and chooses the next frontier to go to as shown in Figure 10b. The frontier selection and mapping processes are repeated until it is manually terminated in Figure 10c.

6.1.2 Wall-Following Exploration

We adopt a wall-following exploration strategy using the Bug algorithm (Choset et al., 2005) in open areas where the frontier-based exploration under-performs. The MAV starts at an arbitrary location near a wall. A 3D occupancy grid map described in Section 5 is built incrementally as the MAV follows the wall. An



(a) Exploration at t_1 (b) Exploration at t_2 (c) Exploration at t_3

Figure 10: Various stages of indoor exploration. Each frontier is given an unique color. White, gray, and black areas represent free, unknown, and occupied space respectively. As the MAV explores the environment, an increasing number of cells in the frontier map are marked as either occupied or free space.

example of the 3D occupancy map is shown in Figure 11. We extract the wall plane from the 3D points from the stereo camera by doing iterative plane fittings with RANSAC. The plane fitting process first begins with a plane hypothesis from a randomly chosen set of 3 points and which is supported by a minimum number of inlier points. A plane hypothesis is taken as the wall plane if its normal is approximately orthogonal to the gravity vector as measured by the IMU. The 3D points belonging to a plane which does not fulfil this criteria are discarded, and the selection process continues iteratively with the remaining 3D points until we find a suitable wall plane. Figure 12 shows an example of a wall detected (highlighted in blue) by the plane detection algorithm. The MAV has to select a desired waypoint once the wall is detected. This is done by first computing the point p_1 on the wall which is closest to the current position of the MAV. We then compute the point p_2 which is at a distance d away from p_1 along the wall plane at the same altitude as the MAV, and the next waypoint p_3 is chosen as the point in the perpendicular direction from the plane with a distance equal to the distance between p_1 and the MAV's current position. The choice of d depends on the admissible distance from one waypoint to another. The chosen waypoint is directly sent to the position controller of the MAV. Exploration with wall-following terminates at the command of the operator.

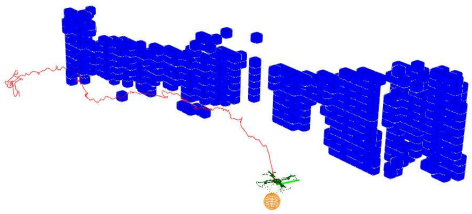


Figure 11: A 3D occupancy map in the form of blue-colored voxels is built incrementally as the MAV is following a wall. The trajectory of the MAV is shown in red. An orange sphere marks the current waypoint.

6.2 Local Planning

The VFH+ algorithm is used to plan the path to the desired frontier if the frontier-based exploration algorithm is selected. The VFH+ planner uses the current pose estimate and occupancy map. At each iteration, the planner constructs a polar histogram centered at the MAV's current position, and expands each cell classified as occupied in the occupancy map by the sum of the MAV's width and safety clearance distance. The score of each histogram sector is based on the distance between the MAV's position and the obstacle cells falling in the histogram sector. The scores of all histogram sectors are thresholded to yield

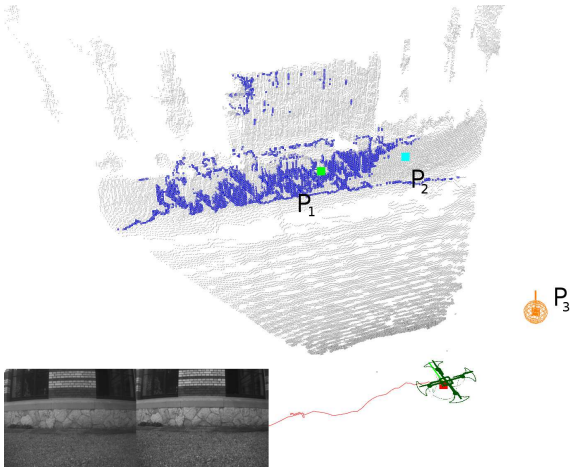


Figure 12: The Bug algorithm detects the wall in an open area, and selects a waypoint at a certain distance from the wall which the MAV flies to. The current images from the stereo camera are shown at the bottom left corner.

candidate vector directions. Each vector direction has a score equal to the weighted sum of three angular differences: between that vector direction and the MAV's current yaw, between that vector direction and the direction towards the desired goal, and between that vector direction and the previously chosen direction. This scoring method facilitates a smooth trajectory. The vector direction with the lowest score is chosen as the direction for the MAV to travel in. A set-point at a preset distance from the MAV along the vector direction is sent to the position controller.

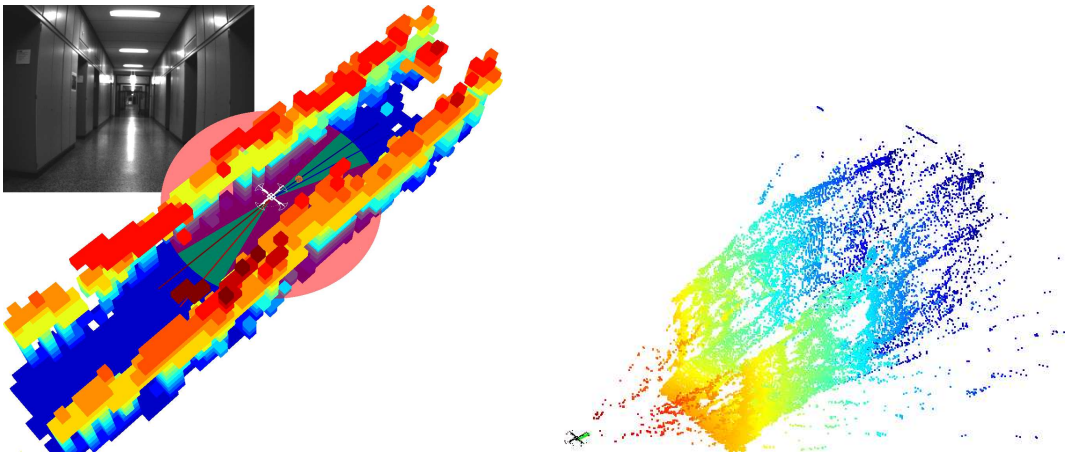
In Figure 13, the frontier-based exploration module has selected the desired waypoint to be at the upper right of the image along the corridor. This figure shows on the left side the cells in the occupancy map classified as occupied and colored from blue to red in order of increasing height. The thresholded polar histogram is visualized as a circle around the MAV. The green sectors represent the set of feasible vector directions, while the red sectors represent the set of infeasible vector directions due to the directions' proximity to nearby obstacles. The lines radiating out from the MAV represent the candidate vector directions. The lines are colored according to their score: blue indicates a low score while red indicates a high score. The set-point represented as an orange sphere indicates the set-point along the desired vector direction.

7 Off-Board Visual SLAM

Images from the stereo camera, metric velocity estimates from the optical flow sensor, and pose estimates from visual odometry are transmitted to the ground station which runs off-board visual pose graph SLAM. This SLAM module estimates the global poses of the MAV using loop detection.

7.1 Visual SLAM

We use the g2o (Kümmerle et al., 2011) framework for our off-board visual pose graph SLAM. Figure 14 shows an example of the hypergraph representation of our visual pose graph SLAM. Each node in the graph represents the absolute pose X_t of the MAV at time t. The edges of the graph represents the constraints between the poses. These constraints are measurements from the visual odometry $z^{vo}_{t,t+1}$, optical flow $z^{of}_{t,t+1}$ and loop-closure constraint $z^{loop}_{t,t+5}$. The visual odometry constraint $z^{vo}_{t,t+1}$ is the transformation between two successive poses X_t and X_{t+1} estimated by visual odometry. The optical flow constraint $z^{of}_{t,t+1}$ is the transformation between two successive poses X_t and X_{t+1} estimated by the optical flow sensor. The



(a) Visualization of obstacle map and VFH+ planning along a corridor. The left stereo image on the top left shows the corridor. The obstacle cells in the map are colored from blue to red in order of increasing height. The map resolution is 0.25 m.

(b) Each point in the point cloud is colored according to its distance from the camera. This point cloud is computed from the current stereo images at the time instance that corresponds to the left subfigure.

Figure 13: Data visualization with a MAV traversing a corridor.

loop-closure constraint $z^{loop}_{t,t+5}$ is the transformation between matching views X_t and X_{t+5} that have been identified by the loop-closure detection step. Additionally, the measurement from the ultrasonic distance sensor z^{alt}_t is used as a prior estimate for the pose graph SLAM. The maximum-likelihood estimates of the MAV positions are computed by minimizing the Euclidean distances between the transformations. The non-linear optimization is done by sparse Cholesky decomposition using the g2o framework.

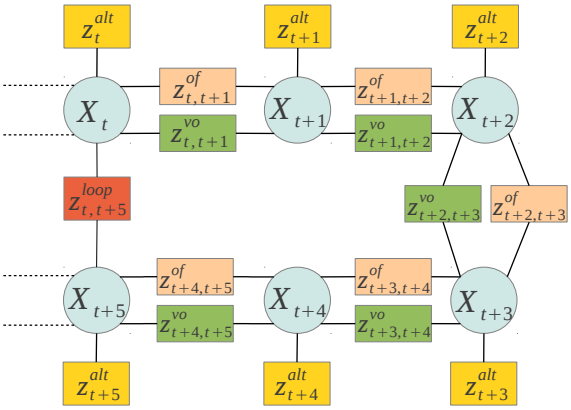


Figure 14: g2o hypergraph representation of our pose graph SLAM problem. X_t represents the pose of the MAV. $z^{vo}_{t,t+1}$, $z^{of}_{t,t+1}$ and $z^{loop}_{t,t+5}$ represent pose-pose constraints that correspond to measurements from visual odometry, optical flow, and loop closures respectively. z^{alt}_t represents the measurement from the ultrasonic distance sensor.

7.2 Loop Detection

We remove drifts in the visual odometry and enforce global consistency on the pose estimations with loop detections and loop closures. In our system, loop detection is done with the vocabulary tree (Nistér and Stewénius, 2006; Fraundorfer et al., 2007) approach. We extract the SURF features and descriptors (Bay

et al., 2008) for every image and quantize them into visual words using a vocabulary tree which was pre-trained on a general image data set. The visual words and image indices are stored in a database which is organized as an inverted file for efficient insertion and retrieval. This inverted file has an index, which given a word index, returns the indices of the images in which that word appears. The global poses of the MAV are also added as meta-data into the database. We rank all existing images in the database according to visual similarity with every new incoming image. The n best candidates go through a geometric verification where we match the SURF features, compute the relative pose between the image frames with RANSAC PnP and check for the feature inliers. A candidate image passes the geometric verification if the inlier count is above a preset threshold. The transformation between the current image and the candidate image that passes the geometric verification test is computed and added to the pose graph as constraint. Node X_t from Figure 14 is an example of a candidate image that passes the geometric verification with the current image $X_{t,t+5}$. $z^{loop}_{t,t+5}$ is the loop constraint between the two nodes. We compute the transformation for loop constraint with the RANSAC PnP that was mentioned earlier in Section 4.2.

8 Experimental Results

We show results from our experiments with the frontier-based exploration and visual pose graph SLAM. Videos of experiments on wall-following and frontier-based exploration can be found at `http://www.inf.ethz.ch/personal/hengli/mav_exploration`.

8.1 State Estimation and Mapping

We demonstrate the state estimation and 3D mapping in an outdoor experiment in which the AscTec Firefly MAV autonomously flies around a rectangular-shaped building using manually-set waypoints. In this environment, tall buildings are located on both sides of the MAV's flight path, and hence, no GPS fix can be obtained. The distance flown by the MAV is 135 m. Figure 15 shows the 3D obstacle map built by the AscTec Firefly; we overlay the obstacle map on aerial imagery for ground truth comparison. For each occupancy cell, we threshold the occupancy probability to check whether that cell is an obstacle. A red line plots the MAV's position estimated by the state estimator during the flight. Drift is clearly visible mainly due to inaccurate magnetometer readings in the area marked by a red circle in Figure 15.

8.2 Frontier-Based Exploration

We test the automonous frontier-based exploration of our MAVs in both indoor and outdoor environments, and present results from the autonomous exploration experiments.

8.2.1 Indoor Experiment

The PIXHAWK Cheetah automonously explores a straight corridor which has an open space in the middle. In this experiment, the indoor environment has low lighting conditions and has little texture, and hence, visual odometry does not work reliably. Here, only the optical flow sensor is used for localization, and the PIXHAWK Cheetah flies from the left end of the corridor to the right end over a distance of 61 m. This flight takes 1 minute and 42 seconds. Figure 16a shows the obstacle map built by the PIXHAWK Cheetah. We overlay this map on the floor plan of the area for ground truth comparison. Figure 16b shows the corresponding frontier map. Both figures show that the pose estimate noticeably drifts downwards. At the end of the flight, the difference between the estimated position and the actual position of the PIXHAWK Cheetah is 4.9 m based on the floor map. In the presence of visual odometry failures, the MAV would not have been able to traverse the corridor without the optical flow sensor.

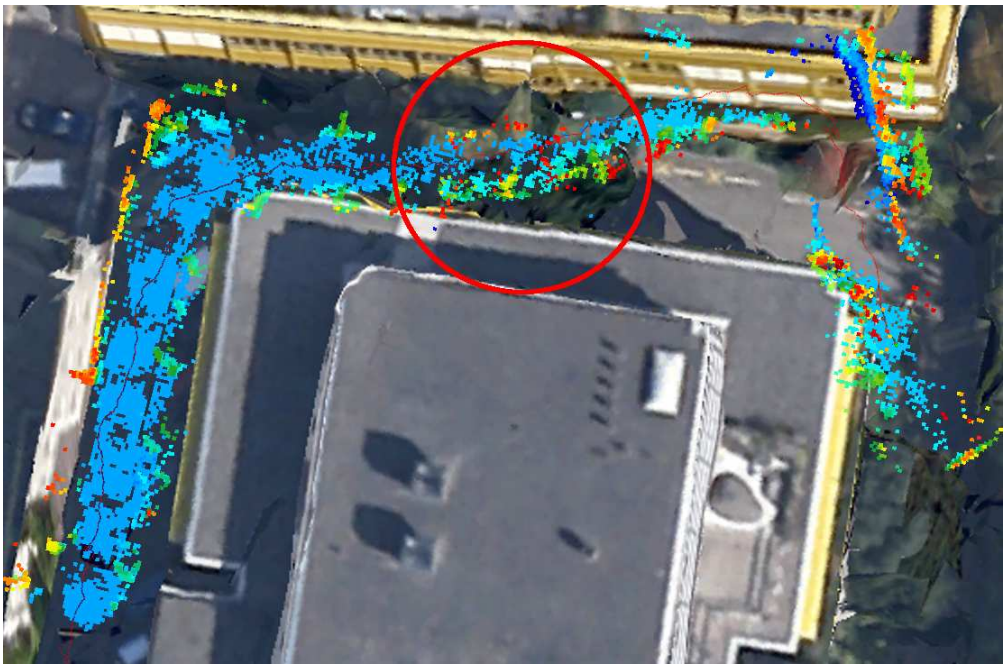


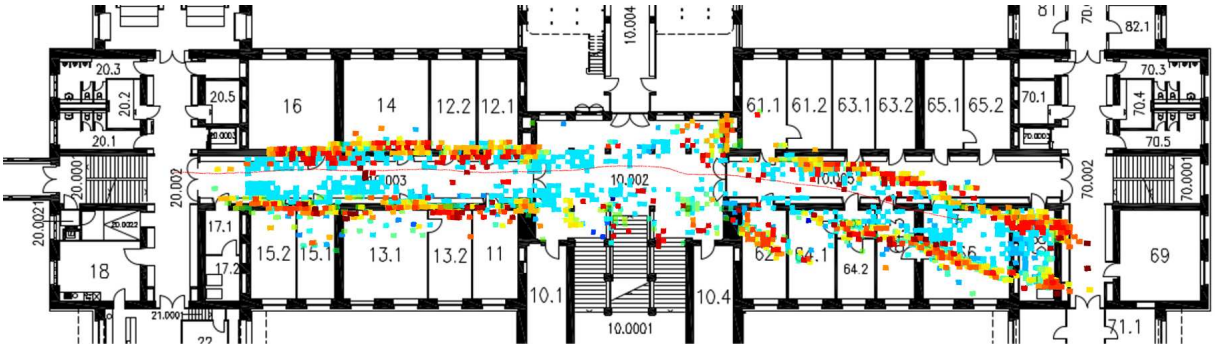
Figure 15: Top view of the 3D obstacle map overlaid on aerial imagery provided by Google Maps. The obstacle cells are colored from blue to red in order of increasing height. A red circle marks the area where inaccurate magnetometer readings occur, and significant drift occurs as a result.

8.2.2 Outdoor Experiments

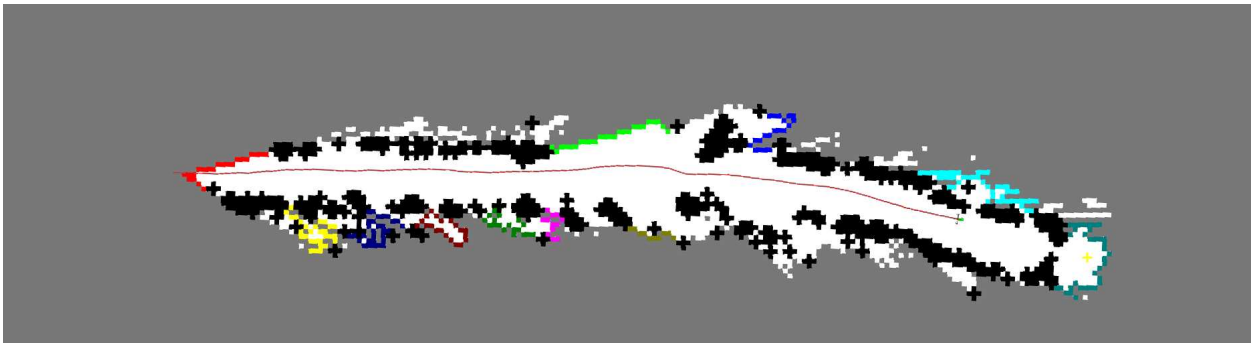
In the first outdoor experiment, the AscTec Firefly flew between tall buildings in which GPS accuracy would have been limited. Obstacles are placed such that the space explored by the AscTec Firefly has a closed boundary. Figure 17 shows the 3D obstacle map and the MAV's path during the exploration. The walls of the buildings are clearly visible in the figure. The ground is clearly visible on the right side of the figure as that area is shaded, and the ground has sufficient texture for stereo block matching. In contrast, the ground is missing on the left side; the corresponding area is not shaded and exposed to strong sunlight, and thus, the ground appears washed-out and lacks texture in the stereo images. Figure 18 shows the corresponding frontier map based on a 2D slice of the 3D occupancy map. Phantom obstacles as a result of specular reflections from windows are clearly visible as isolated small black dots. We can see that in Figure 19, the built 3D obstacle map aligns well with the aerial imagery, and demonstrates that the state estimation performs well. The flight distance is 94 m, and the flight time is 2 minutes and 29 seconds. There is considerably less drift in the pose estimates compared to the indoor experiment as the outdoor environment has more texture. At the end of the flight, the difference between the estimated position and the actual position of the AscTec Firefly is measured to be 2.0 m based on the aerial imagery which is provided by swisstopo[1] and has ground resolution 10 cm.

In the second outdoor experiment, the AscTec Firefly flew along a corridor-like structure. We show the 3D obstacle map in Figure 20, and the map overlaid on aerial imagery in Figure 21. In this experiment, the ground is clearly visible together with the building walls, as lighting conditions are close to ideal. Phantom obstacles seen as isolated red blocks can be visible in the map; these obstacles are attributed to incorrect stereo measurements arising from specular reflections. The flight distance is 83 m, and the flight time is 3 minutes and 58 seconds. At the end of the flight, the difference between the estimated position and the actual position of the AscTec Firefly is 3.5 m based on the aerial imagery.

[1] http://www.swisstopo.admin.ch/



(a) Top view of the 3D obstacle map overlaid on a floor plan.



(b) Top view of the frontier map.

Figure 16: An indoor experiment in a corridor-like environment.

8.3 Visual SLAM

We test the scalability of our pose graph SLAM with a large outdoor dataset that consists of more than 5000 stereo image pairs and spans a total trajectory of approximately 1 km. The dataset contains two large and two small nested loops, and it also contains passing cars, bicyclists, and pedestrians. As the dataset was collected along urban streets, we manually carried the MAV around instead of flying it to prevent accidental injuries to the pedestrians. We took a log of the dataset with the PIXHAWK Cheetah and processed it off-board. Figure 22a shows the raw pose graph of the trajectory, which is estimated from visual odometry and optical flow, and overlaid on the satellite image. The green lines are loop closure candidates detected by the vocabulary tree. It is clear from the figure that the raw pose graph is not accurate due to drifts in the visual odometry and optical flow. Figure 22b shows the pose graph after optimization and overlaid on the satellite image. Since the loop closure constraints are taken into account, the optimized pose graph is significantly more accurate compared to the raw pose graph. Figure 23 shows the point cloud that was reconstructed based on stereo images and the optimized pose graph. In addition, Figure 24 shows the reconstructed point cloud from the first outdoor exploration experiment which was discussed in Section 8.2.2.

9 Discussions

Our experiments show that vision-enabled MAVs are feasible for mapping and exploration in both indoor and outdoor environments without GPS. We developed two different MAV systems both capable of autonomous vision-based exploration. During the course of our experiments, we learned valuable lessons which we can apply towards the next iteration of vision-enabled MAVs.

The success of our vision-enabled MAV systems can be attributed to two factors:

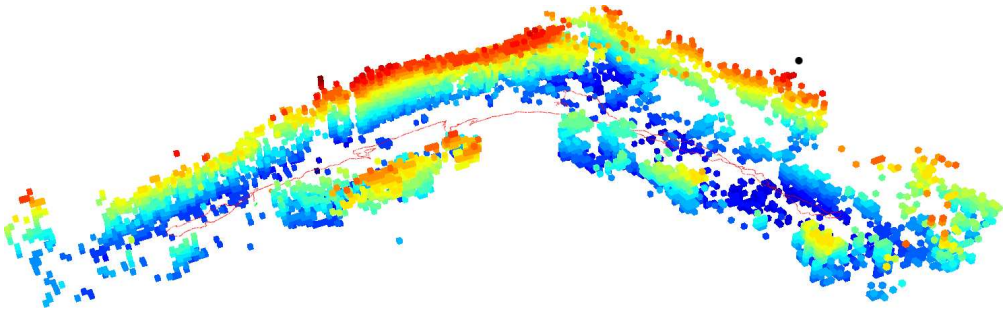


Figure 17: Side view of the 3D obstacle map built by the AscTec Firefly.

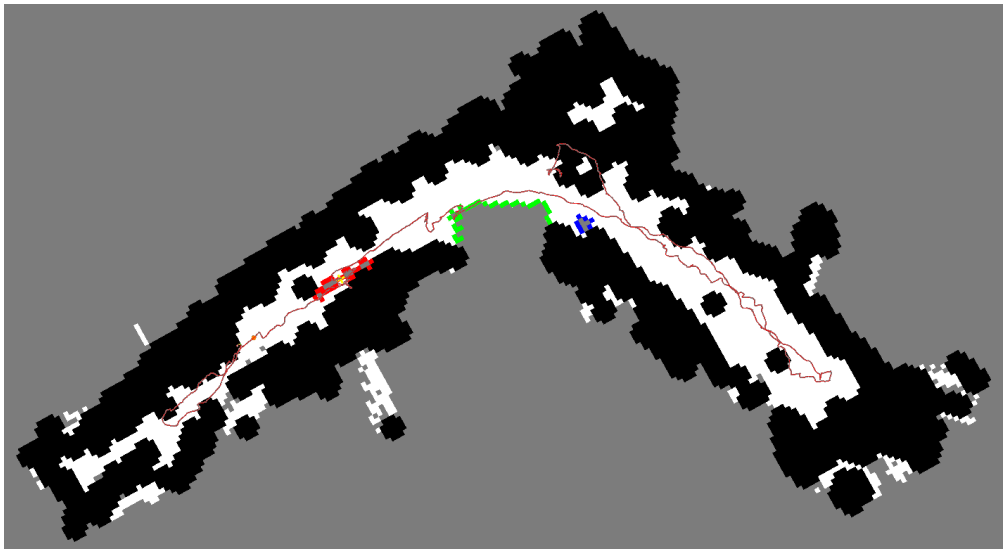


Figure 18: Top view of the 2D slice of the map tagged with frontiers.

- **State estimation in low-texture environments**: Our optical flow sensor works reliably in low-texture environments in which visual odometry usually fails. Given the large drifts associated with MEMS IMUs normally found on MAVs, accurate metric velocity estimates from our optical flow sensor are critical to keeping the MAV in the air, and more importantly, maintaining a stable flight with minimal drift.

- **Integration of processes with low computational requirements**: Stereo processing is a inherently computationally demanding process which typically takes up one of the two CPU cores available on the single board computer. We make efficient use of the other CPU core by using processes that require little computational resources but are necessary for MAV autonomy. As a result, we are able to show autonomous vision-based MAV exploration with a single board computer equipped with a dual-core CPU.

We also discovered issues that constrain the performance of the MAV and which we have to solve before vision-enabled MAVs are able to handle all kinds of environments. At the same time, we propose solutions to these issues.

- **Specular objects**: There are sometimes incorrect but consistent depth measurements arising from specular surfaces, in particular, windows. Hence, phantom obstacles appear in the 3D map in environments where windows are present. Work (Zhou and Kambhamettu, 2006) has been done

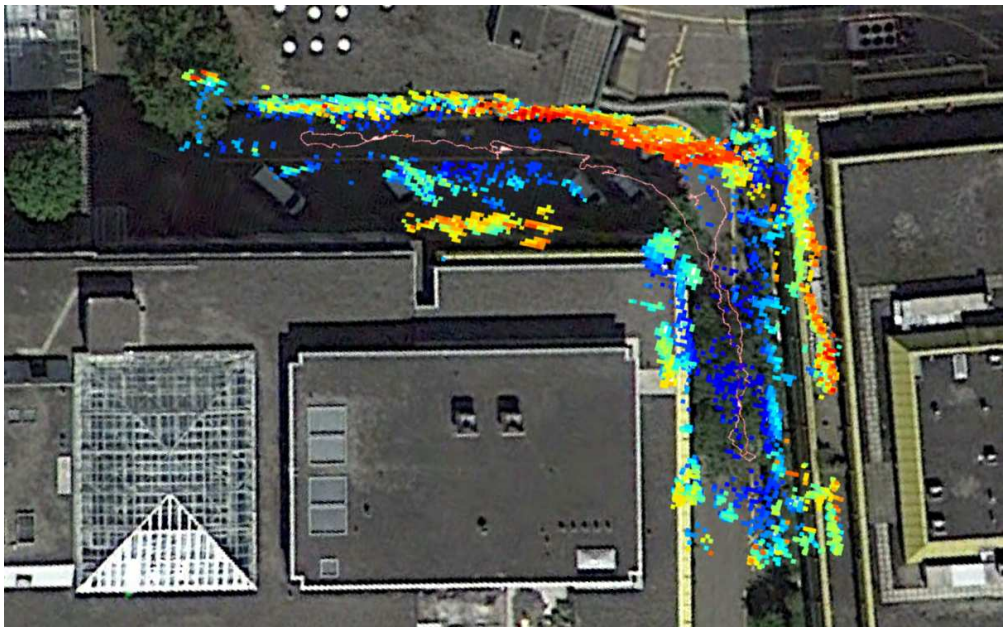


Figure 19: Top view of the 3D obstacle map overlaid on aerial imagery provided by Google Maps.

to reliably obtain stereo depth maps in the presence of specular reflection, but to the best of our knowledge, there is no real-time implementation for doing so. Semi-global matching may mitigate the problem but cannot be deployed on a computationally constrained MAV. A FPGA that implements semi-global matching as in (Schmid et al., 2013) is a possible solution.

- **Absence of global pose estimation**: We do not perform global pose estimation on-board as there are insufficient computational resources for doing so. We also do not use the global pose estimates from the off-board visual SLAM as the Wi-Fi connection between the ground station and the MAV breaks down periodically in typical scenarios. Another reason is that after a connection outage, if the global pose estimate received from off-board visual SLAM significantly differs from the current pose estimate on the MAV, a pose jump occurs, and the 3D map becomes distorted as a result, causing the exploration to fail. Similarly, when there is a loop closure, the entire pose graph has to be optimized, and as a result, to avoid the 3D map from becoming distorted, the entire 3D map has to be rebuilt. However, there exists no computationally tractable CPU-based solution for rebuilding 3D maps in real-time in the case of a loop closure. (Whelan et al., 2013) is the only known solution for real-time 3D map deformation with loop closures but requires use of a GPU which is not feasible for use on a MAV due to the GPU's considerable weight and high power consumption.

- **Loosely-coupled pose estimation**: The optical flow, visual odometry, and inertial modules run separately. Our loosely-coupled pose estimation restricts the dynamics of the MAV, and limits the MAV to slow movements. However, this design allows greater flexibility in the system design, as modules can be developed separately, and visual SLAM on its own provides precise pose estimates in general.

- **Flight limitations**: The use of the ultrasonic range sensor by the optical flow sensor limits the altitude of the MAV and the horizontal speed of the MAV to 5 m and 1.5 m/s respectively, and introduces the assumption that the ground over which the MAV flies is flat. The use of the optical flow sensor is critical for flight stabilization, and we cannot use our stereo visual odometry to entirely replace the optical flow sensor as the stereo visual odometry does not work in cases where there are an insufficient number of features. One example of such a situation is when the MAV is in close proximity to and is facing a wall with little texture. A real-time visual SLAM implementation with multiple cameras arranged in a configuration that provides an all-surround view can overcome the

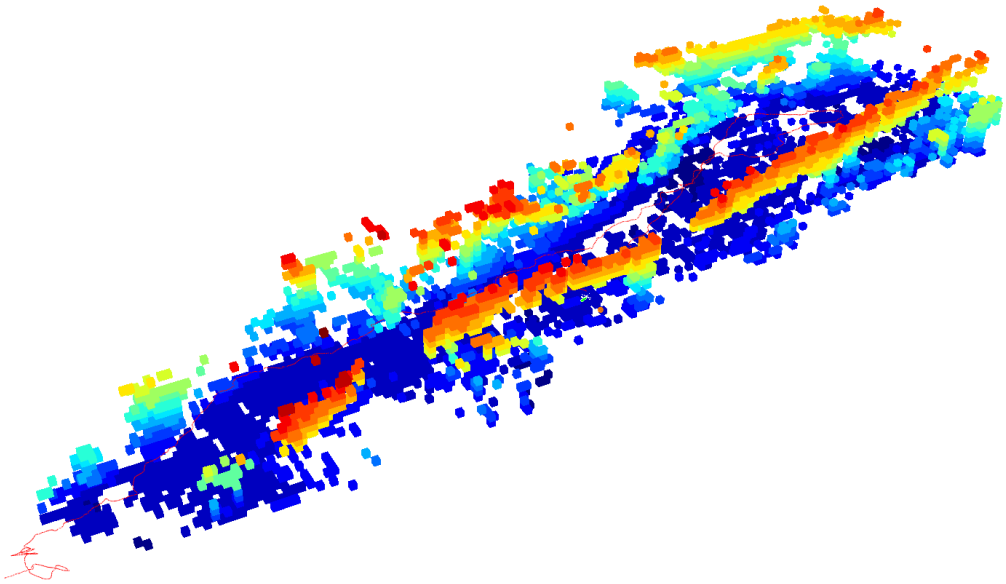


Figure 20: Side view of the 3D obstacle map built by the AscTec Firefly.

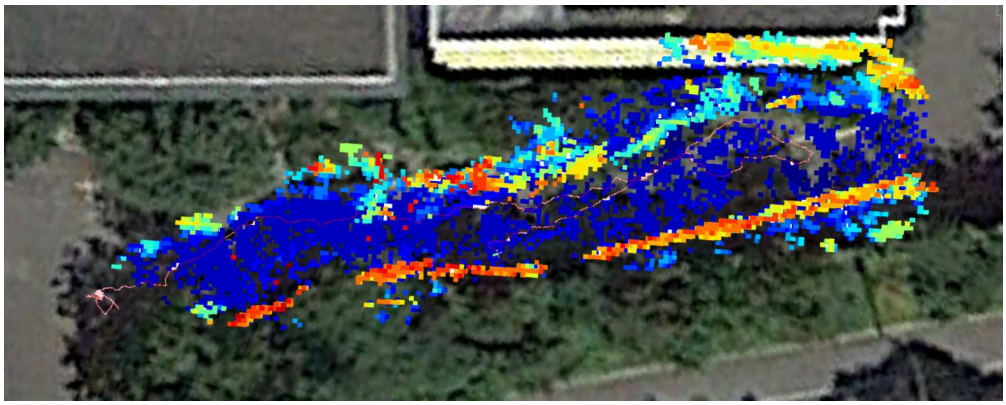


Figure 21: Top view of the 3D obstacle map overlaid on aerial imagery provided by Google Maps. The voxels are colored from blue to red in order of increasing height.

issues associated with the limited field of view of a single stereo camera. Furthermore, real-time visual SLAM with an all-surround view can potentially make the optical flow sensor redundant, and thus, remove the flight limitations associated with the optical flow sensor. The all-surround view allows the MAV to fly in close proximity to objects, and still be able to reliably estimate its position. We can use the generalized camera concept to treat many cameras as one. For example, (Lee et al., 2013) uses a 4-camera rig that provides a 360-degree field of view, and applies the generalized camera model and the assumption of an Ackermann motion model to estimate the motion of a car. It is possible to make use of this generalized camera model with known attitude angles from an IMU to estimate the motion of the MAV in real-time.

- **Absence of global path planning and full 3D exploration**: The absence of these modules limit how far vision-based MAVs can advance. Stereo processing takes up approximately one CPU core, and after accounting for all other processes that utilize the remaining CPU core, there are little residual computational resources. Hence, global path planning and full 3D exploration are not possible. However, Ascending Technologies has recently released a Intel quad-core CPU board for use on the AscTec Firefly, and we plan to make use of this board together with current research on real-time visual SLAM, FPGA stereo, and dense mapping with loop closures to advance the

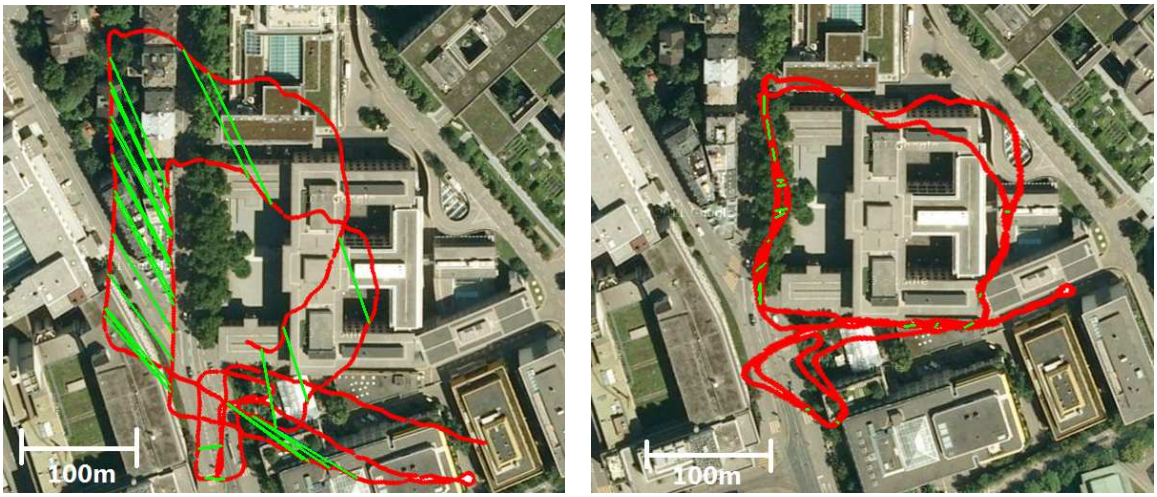


Figure 22: Large-scale pose graph before and after optimization. We overlay the pose graph on aerial imagery provided by Google Maps. The estimated poses of the MAV are marked with a red line. The green lines represent the detected loop closures.

state-of-the-art for vision-based MAVs.

- **Waypoint-based navigation**: The MAV follows waypoints set by either wall-following or the VFH+ planner. This waypoint following can lead to erratic movements of the MAV as waypoints change constantly, causing the MAV to continually re-orient itself. Furthermore, local planning and wall-following leads to suboptimal behavior especially in cul-de-sac situations and when negotiating corners. Although we have developed global path planning based on the state lattice in (Heng et al., 2011), global path planning cannot currently be implemented on our MAVs as it exclusively requires one CPU core which is not available given the significant number of modules running on the on-board computer.

10 Conclusions

The advantages of using cameras as the main sensor modality for MAVs are the primary motivation for this work. We have showed the feasibility of doing pose estimation, autonomous mapping, and exploration on-board with our two MAV systems equipped with a stereo camera and optical flow sensor. In comparison with other existing approaches that use vision, we are able to do pose estimation, autonomous mapping, and exploration on-board in environments with little texture because of our exclusive combination of a front-looking stereo camera and a downward-looking optical flow sensor. In addition, we showed large-scale pose graph SLAM that was done off-board. Experimental results that verified our approaches are shown. It should be emphasized that despite the extensive experiments that we have done for vision-based autonomous mapping and exploration of the MAV, we still face many other challenges associated with vision-based autonomous MAVs such as detecting sufficient image features in challenging lighting conditions, and reliable stereo in the presence of specular reflections. All these challenges remain as future works.

Multimedia

The accompanying video file shows a MAV performing the following tasks: hovering with the optical flow sensor, doing wall-following exploration in an outdoor setting, and doing frontier-based exploration in both outdoor and indoor environments.

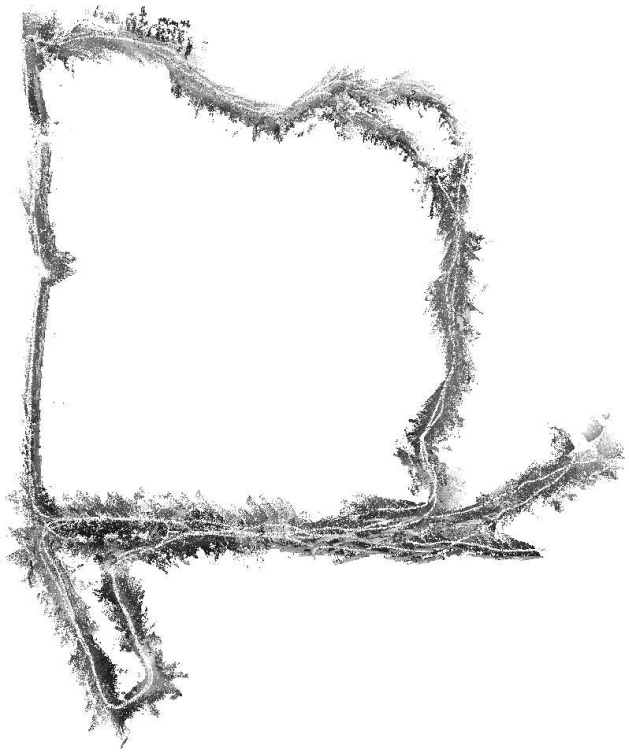


Figure 23: The reconstructed point cloud based on stereo images and the optimized pose graph from a large outdoor dataset.

Acknowledgments

Lionel Heng is funded by the DSO National Laboratories Postgraduate Scholarship. We thank the reviewers for their invaluable feedback.

References

Achtelik, M., Achtelik, M., Weiss, S., and Siegwart, R. (2011). Onboard imu and monocular vision based control for mavs in unknown in- and outdoor environments. In *International Conference on Robotics and Automation (ICRA)*, pages 3056--3063.

Andert, F. (2009). Drawing stereo disparity images into occupancy grids: Measurement model and fast implementation. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5191--5197.

Bachrach, A., Prentice, S., He, R., and Roy, N. (2011). Range - robust autonomous navigation in gps-denied environments. *Journal of Field Robotics*, 28(5):644--666.

Bay, H., Ess, A., Tuytelaars, T., and Gool, L. V. (2008). Surf: Speeded up robust features. *Computer Vision and Image Understanding*, 110(3):346--359.

Bills, C., Chen, J., and Saxena, A. (2011). Autonomous mav flight in indoor environments using single image perspective cues. In *International Conference on Robotics and Automation (ICRA)*, pages 5776--5783.

Blösch, M., Weiss, S., Scaramuzza, D., and Siegwart, R. (2010). Vision based mav navigation in unknown and unstructured environments. In *International Conference on Robotics and Automation (ICRA)*, pages 21--28.

(a) (b)

Figure 24: (a) Top view and (b) side view of the reconstructed point cloud based on stereo images and the optimized pose graph from the outdoor exploration experiment that corresponds to figures 16-18.

Calonder, M., Lepetit, V., Ozuysal, M., Trzcinski, T., Strecha, C., and Fua, P. (2012). BRIEF: Computing a Local Binary Descriptor Very Fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1281--1298.

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.

Dryanovski, I., Valenti, R. G., and Xiao, J. (2013). An open-source navigation system for micro aerial vehicles. *Autonomous Robots*, 34(3):177--188.

Eberli, D., Scaramuzza, D., Weiss, S., and Siegwart, R. (2011). Vision based position control for mavs using one single circular landmark. *Journal of Intelligent and Robotic Systems*, 61(1-4):495--512.

Fischler, M. and Bolles, R. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. volume 24, pages 381--395.

Fraundorfer, F., Engels, C., and Nistér, D. (2007). Topological mapping, localization and navigation using image collections. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 3872--3877.

Fraundorfer, F., Heng, L., Honegger, D., Lee, G. H., Meier, L., Tanskanen, P., and Pollefeys, M. (2012). Vision-based autonomous mapping and exploration using a quadrotor mav. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 4557--4564.

Grzonka, S., Grisetti, G., and Burgard, W. (2012). A fully autonomous indoor quadrotor. *IEEE Transactions on Robotics*, 28(1):90--100.

Hartley, R. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

Heng, L., Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M. (2011). Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. In *International Conference on Robotics and Automation (ICRA)*, pages 2472--2477.

Honegger, D., Meier, L., Tanskanen, P., and Pollefeys, M. (2013). An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *International Conference on Robotics and Automation (ICRA)*, pages 1736--1741.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189--206.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g2o: A general framework for graph optimization. In *International Conference on Robotics and Automation (ICRA)*, pages 3607--3613.

Lee, G. H., Fraundorfer, F., and Pollefeys, M. (2013). Motion estimation for self-driving cars with a generalized camera. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2746--2753.

Makarenko, A. A., Williams, S. B., Bourgault, F., and Durrant-Whyte, H. F. (2002). An experiment in integrated exploration. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 534--539.

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.

Mori, T. and Scherer, S. (2013). First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles. In *International Conference on Robotics and Automation (ICRA)*, pages 1750--1757.

Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, page 652659.

Nistér, D. and Stewénius, H. (2006). Scalable recognition with a vocabulary tree. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2161--2168.

Revelles, J., Urea, C., and Lastra, M. (2000). An efficient parametric algorithm for octree traversal. In *Journal of WSCG*, pages 212--219.

Rosten, E., Porter, R., and Drummond, T. (2010). Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105--119.

Scaramuzza, D. and Fraundorfer, F. (2011). Visual odometry: Part i - the first 30 years and fundamentals. *IEEE Robotics and Automation Magazine*, 18(4):80--92.

Schmid, K., Tomic, T., Ruess, F., Hirschmüller, H., and Suppa, M. (2013). Stereo vision based indoor/outdoor navigation for flying robots. In *International Conference on Intelligent Robots and Systems (IROS)*.

Shen, S., Michael, N., and Kumar, V. (2012). Stochastic differential equation-based exploration algorithm for autonomous indoor 3d exploration with a micro-aerial vehicle. *International Journal of Robotic Research*, 31(12):1431--1444.

Shen, S., Michael, N., and Kumar, V. (2013). Vision-based state estimation for autonomous rotorcraft mavs in complex environments. In *International Conference on Robotics and Automation (ICRA)*, pages 1758--1764.

Ulrich, I. and Borenstein, J. (1998). Vfh+: Reliable obstacle avoidance for fast mobile robots. In *International Conference on Robotics and Automation (ICRA)*, pages 1572--1577.

Wagner, D. and Schmalstieg, D. (2007). Artoolkitplus for pose tracking on mobile devices. In *Computer Vision Winter Workshop*.

Weiss, S., Achtelik, M., Lynen, S., Chli, M., and Siegwart, R. (2012). Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *International Conference on Robotics and Automation (ICRA)*, pages 957--964.

Whelan, T., Kaess, M., Leonard, J., and McDonald, J. (2013). Deformation-based loop closure for large scale dense rgb-d slam. In *International Conference on Intelligent Robots and Systems (IROS)*.

Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. In *International Symposium on Computational Intelligence in Robotics and Automation*, pages 146--151.

Zhou, W. and Kambhamettu, C. (2006). Binocular stereo dense matching in the presence of specular reflections. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2363--2370.