Robust Autonomous Flight in Constrained and Visually Degraded Shipboard Environments

Zheng Fang
State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang, Liaoning 110819, China
Shichao Yang, Sezal Jain, and Geetesh Dubey
Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
e-mail: shichaoy@andrew.cmu.edu, sezal@andrew.cmu.edu, gdubey@andrew.cmu.edu
Stephan Roth
Sensible Machines Inc., Pittsburgh, Pennsylvania 15212
e-mail: sroth@sensiblemachines.com
Silvio Maeta and Stephen Nuske
Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
e-mail: smaeta@andrew.cmu.edu, nuske@andrew.cmu.edu
Yu Zhang
Zhejiang University, Hangzhou, Zhejiang 310027, China
e-mail: zhangyu80@zju.edu.cn
Sebastian Scherer
Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213
e-mail: basti@cmu.edu

Received 20 November 2015; accepted 28 June 2016

This paper addresses the problem of autonomous navigation of a micro aerial vehicle (MAV) for inspection and damage assessment inside a constrained shipboard environment, which might be perilous or inaccessible for humans, especially in emergency scenarios. The environment is GPS-denied and visually degraded, containing narrow passageways, doorways, and small objects protruding from the wall. This causes existing two-dimensional LIDAR, vision, or mechanical bumper-based autonomous navigation solutions to fail. To realize autonomous navigation in such challenging environments, we first propose a robust state estimation method that fuses estimates from a real-time odometry estimation algorithm and a particle filtering localization algorithm with other sensor information in a two-layer fusion framework. Then, an online motion-planning algorithm that combines trajectory optimization with a receding horizon control framework is proposed for fast obstacle avoidance. All the computations are done in real time on the onboard computer. We validate the system by running experiments under different environmental conditions in both laboratory and practical shipboard environments. The field experiment results of over 10 runs show that our vehicle can robustly navigate 20-m-long and only 1-m-wide corridors and go through a very narrow doorway (66-cm width, only 4-cm clearance on each side) autonomously even when it is completely dark or full of light smoke. These experiments show that despite the challenges associated with flying robustly in challenging shipboard environments, it is possible to use a MAV to autonomously fly into a confined shipboard environment to rapidly gather situational information to guide firefighting and rescue efforts. © 2016 Wiley Periodicals, Inc.

1. INTRODUCTION

Over the past few years, micro aerial vehicles (MAVs) have gained wide popularity in both military and civil domains. For example, MAVs have made a huge impact in the areas of surveillance and reconnaissance. In this paper, we aim to develop a MAV that is capable of autonomously navigating through a ship to aid in fire control, damage assessment, and inspection, all of which might be dangerous or inaccessible for humans. The idea is that the drone will be released into a ship's potentially dark, smoke-filled hallways in the event of an emergency and will proceed to fly autonomously through the vessel, looking for fires, measuring heat, and locating any personnel who may be found along the way. After reaching its destination, the MAV will

Direct correspondence to: Zheng Fang: e-mail: fangzheng @mail.neu.edu.cn

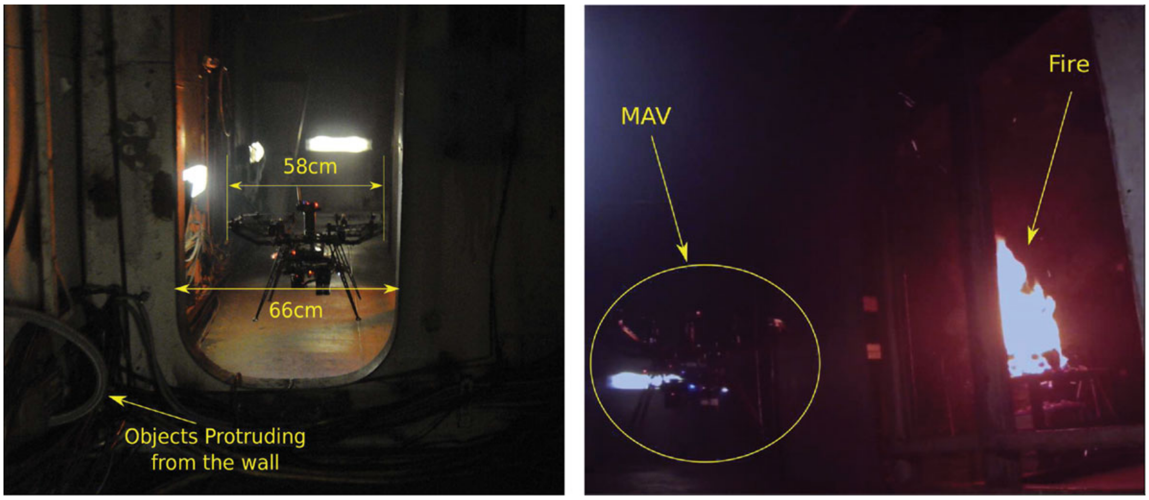


Figure 1. Autonomous MAV for fire detection inside a ship: The left picture shows the MAV's autonomous flight through doorways. The right picture shows a testing scenario involving fire.

return and provide the information it has collected to the operator. The operator interface displays the location structure and video information, and the operator assesses any structural damage and knows the locations of fires and personnel inside the ship. An illustrative picture is shown in Figure 1. The key technology for realizing this goal is autonomous navigation in such a *constrained, visually degraded, and GPS-denied* environment.

Recently, there have been several MAV systems designed for vessel environments (Bonnin-Pascual, Garcia-Fidalgo, & Ortiz, 2012; Eich, Bonnin-Pascaul, Gracia-Fidalgo, & Ortiz, 2014; Garcia-Fidalgo, Ortiz, Bonnin-Pascual, & Company, 2015; Ortiz, Bonnin-Pascual, & Garcia-Fidalgo, 2013). Among these, most are designed for visual inspection, which occurs autonomously or semi-autonomously inside of the spacious and bright vessels or ships to collect images for later analysis. However, to the best of our knowledge, there have been very few, if any, autonomous MAV navigation systems demonstrated in *constrained, visually degraded, and GPS-denied* shipboard environments for fire detection or rescue purpose. For successful operation in such environments, we need to address several challenging problems compared to the existing MAVs designed for vessels (Eich et al., 2014; Garcia-Fidalgo et al., 2015) or indoor environments (Droeschel et al., 2015; Grzonka, Grisetti, & Burgard, 2012; Nieuwenhuisen, Droeschel, Beul, & Behnke, 2015; Shen, Michael, & Kumar, 2011). First, the MAV should be small enough to travel in narrow corridors (1 m width) with narrower doorways (66 cm width) inside of the vessel. Therefore, only lightweight sensors, which provide limited measurement range and noisy data, can be used due to the payload limitations. Second, the onboard computational resources are very limited, while every module should run in real-time, posing great challenges for state estimation and motion planning. Third, since the environment may be dark and smoke-filled, we are unable to use state-of-the-art visual navigation methods. Our mean is that even you put some LED lights on the MAV, sometimes you still could not get useful images under hazy conditions, for example in dense smoke. Therefore, I think the original sentence is correct: Though putting LED lights on the MAV can provide better illumination, it might not output a useful RGB image under hazy conditions. In addition, clear corridors with few geometric features, or corridors with many small objects on the wall, pose a great difficulty for accurate pose estimation and obstacle avoidance. In addition, air turbulence from the MAV in confined spaces makes it difficult to achieve precise control, thus the MAV is not as stable as in a spacious environment. This makes state estimation and motion planning more difficult.

In this paper, we present a MAV system that can autonomously navigate inside this type of challenging (constrained, visually degraded, and GPS-denied) shipboard environment using only onboard sensors and processors. To address the above challenges, we extend our previous work (Fang & Scherer, 2015; Fang et al., 2015) by using a multisensor fusion framework to obtain robust, accurate, and high-frequency state estimates to enable our MAV to go through narrow doorways autonomously. We first propose a fast odometry estimation method that can directly recover the relative pose from a series of depth images, which can work very well in challenging visually degraded environments. Then, we fuse the odometry with inertial measurement unit (IMU) information, and we feed the fused visual-inertial odometry into a particle filter to realize real-time six-degree-of-freedom (6DoF) localization in a given three-dimensional (3D) map. After that, the pose estimates from visual-inertial odometry and localization are fused with other sensor information to obtain fast and robust state estimates for real-time control. Furthermore, we present an online motion-planning algorithm using a modified CHOMP trajectory optimization method under a receding horizon control framework to obtain a minimum-snap collision-free trajectory for navigation in case of damage or dynamic obstacles inside the shipboard. As a further contribution, our implementations of this autonomous navigation system are available as open-source robot-operating system (ROS) packages.[1]

We demonstrate the effectiveness of our system through both laboratory and field experiments. The field experiment was performed in a constrained shipboard environment containing a 20-m-long, 1-m-wide corridor and a 66-cm-wide doorway. The width of the vehicle is 58 cm, leaving only 4 cm clearance on both sides. We conducted more than 10 runs under various environmental conditions, from normal to completely dark and smoke-filled environments, to demonstrate the autonomous navigation capabilities of our MAV.

The remainder of this article is organized as follows. Section 2 surveys related work. In Section 3, we present an overview of our system. Section 4 explains the details of the proposed odometry estimation, the localization algorithms running on the navigation controller, and the state estimation algorithm running on the flight controller. In Section 5, we introduce our motion-planning algorithms, which consist of a global path-planning module, an online obstacle-mapping module, and a local motion-planner module. Section 7 describes the laboratory and field experiments, evaluating the accuracy, robustness, and real-time performance of our proposed methods and demonstrating the performance of our system under various environmental conditions. In Section 8, we discuss lessons learned from this project. We summarize and conclude in Section 9.

2. RELATED WORK

In recent years, a number of autonomous navigation solutions have been proposed for MAVs. Those solutions mainly differ in the sensors used for perception in the autonomous navigation problem, the amount of processing performed onboard/offboard, and the assumptions made about the environment.

2D LIDARs have been used extensively and successfully for autonomous navigation of MAVs for their accuracy and low latency (Dryanovski et al., 2013; Grzonka et al., 2012; Shen et al., 2011). However, those systems are usually only suitable for structured or 2.5D environments. Compared to 2D LIDARs, 3D LIDARs, which provide more information on the surrounding environments, are widely used on ground mobile robots for autonomous navigation (Kr, Colas, Furgale, & Siegwart, 2014; Nüchter, Lingemann, Hertzberg, & Surmann, 2007). However, 3D LIDARs are often much more expensive and heavier than 2D LIDARs. Up to now, such 3D laser scanners have rarely been used on lightweight MAVs due to limited payload and computation resources. In the past few years, there have also been some researchers who tried to use a customized 3D LIDAR (rotating 2D LIDAR) for autonomous navigation of MAVs (Droeschel et al., 2015; Nieuwenhuisen et al., 2015; Scherer et al., 2012). However, these "lightweight" rotating 2D LIDARs (usually more than 400 g) are still too heavy for small aerial vehicles aiming to work in very confined environments. Recently, there have also been many vision-based navigation systems because cameras can provide rich information and have low weight. For example, a stereo camera is used in Fraundorfer et al. (2012) and Schauwecker & Zell (2014), and a monocular camera with IMU is used in Wu, Johnson, Kaess, Dellaert, & Chowdhary (2013), Scaramuzza et al. (2014), and Weiss et al. (2011), but vision is sensitive to illumination changes, and the camera could not function in dark or smoky environments. More recently, RGB-D cameras have become very popular for autonomous navigation of indoor MAVs (Fang & Scherer, 2014; Flores, Zhou, Lozano, & Castillo, 2014; Huang & Bachrach, 2011; Valenti, Dryanovski, Jaramillo, & Str, 2014) because they can provide both color and depth images. For example, in Huang & Bachrach (2011), a RGB-D visual odometry method is proposed for real-time pose estimation of a MAV, and a 3D map is created offline. In Valenti et al. (2014), a fast visual odometry method is used for pose estimation, and a 3D visual SLAM is used for constructing a 3D octomap in real time.

Unfortunately, the existing autonomous navigation solutions cannot work in our case because our application environment is a confined, complex, visually degraded 3D environment that may be very dark or filled with smoke. For example, for state estimation, vision-based methods (Huang & Bachrach, 2011; Weiss et al., 2011) could not work in our case because our environment is potentially dark or smoke-filled. In addition, for obstacle avoidance, 2D LIDAR-based methods are also unqualified for this complex environment because they only perceive planar information while there are many small objects (e.g., slim cables and pipes) protruding from the wall in our environment. Although 3D LIDARs combined with other sensors have been successfully used for obstacle detection and autonomous navigation of MAVs in Droeschel et al. (2015) and Nieuwenhuisen et al. (2015), unfortunately those 3D LIDARs are still too heavy for our MAV. In their work, they usually use Octocopter (size: 85 cm×85 cm×35 cm) as their experiment platform, which is much bigger than ours (size: 58 cm×58 cm×32 cm), and it is also too big for our application environments (the doorway is only 66 cm wide). In addition, their systems require stereo visual odometry for scan assembly, which also cannot work well in our visually degraded environments. In addition, the motion-planning methods of many existing systems either compute paths offline (Dryanovski et al.,

[1]https://bitbucket.org/castacks/

2013; Valenti et al., 2014) or rely heavily on prior maps for obstacle detection (Grzonka et al., 2012), while in our case we want the vehicle to be able to avoid unknown obstacles. Some papers generate angles (Fraundorfer et al., 2012) or waypoints (Shen et al., 2011) online, but those commands are not smooth enough and are not suitable for high precision control in confined shipboard environments.

In this paper, we present a robust autonomous navigation system that can work in challenging practical shipboard environments. In our system, we mainly use depth images for *state estimation* and *motion planning*, which can work in completely dark or even light smoke-filled environments. In addition, all the components of the system run on an onboard ARM-based embedded computer.

In terms of *state estimation* of MAVs, it is mainly composed of two subproblems, namely *odometry estimation* (relative pose estimation) and *localization* (absolute pose estimation). For *odometry estimation*, many methods have been proposed with stereo cameras, monocular cameras, RGB-D cameras, and 2D laser scanners. In Achtelik et al. (2009), an odometry estimation method based on a stereo camera and a 2D laser scanner is proposed for indoor MAVs. This method uses a sparse visual feature matching and scan matching algorithm to compute odometry. However, this method runs on the ground station computer (not the onboard computer) in their paper. A monocular visual odometry method that can run very fast on an embedded computer is proposed in Forster et al. (2014). However, monocular visual odometry can only estimate the odometry up to an unknown scale. To solve the unknown scale problem, IMU information is often used (Weiss et al., 2011, 2012) to estimate the absolute metric scale. In recent years, many RGB-D visual odometry methods have also been proposed. For example, Huang & Bachrach (2011) propose the *Fovis* RGB-D odometry estimation method for MAVs. In Kerl et al. (2013), a dense RGB-D visual odometry that minimizes the photometric error between two RGB images is proposed. Pomerleau, Colas, Siegwart, & Magnenat (2013) develop an ICP-based odometry estimation method that only uses depth information. However, this method can only run at about 10 Hz on our embedded computer (Odroid XU3), and the CPU usage is very high. For *localization*, there are several ways to locate a robot on a given map. The first kinds of methods are 2D methods (Angeletti & Valente, 2008; Grzonka, Grisetti, & Burgard, 2009). However, those methods usually only work in structured or 2.5D environments. Some people also use a floor plan for the localization by using a RGB-D camera (Biswas & Veloso, 2012). This method is efficient and fast, but limited to environments with many line features. The second kinds of methods are 3D methods. A common idea is to create a global point cloud map and then use ICP (Besl & McKay, 1992) or NDT (Stoyanov et al., 2012) -based methods to match the current point cloud to the global map. However, those methods are usually very slow. Some researchers also try to use 3D planes as the global map to locate the robot (Cupec, Nyarko, Filko, Kitanov, & Petrović, 2013; Fallon, Johannsson, & Leonard, 2012). For example, Fallon et al. (2012) propose the Kinect Monte Carlo Localization (KMCL) method. However, this method only estimates x, y, and yaw using the particle filter, and it requires a powerful GPU to run in real time. Oishi Oishi et al. (2013) use a particle filter to track the robot's pose in a known 3D NDT map. However, this method is still too slow to run on a MAV. Another method is an Octomap-based method (Maier, Hornung, & Bennewitz, 2012). But in their paper, they have a relatively accurate and robust odometry from the encoders, and everything is running on a remote desktop. Bry et al. (2012) also propose a real-time localization algorithm based on an Octomap for a fix-wing MAV. However, they use a highly accurate 2D laser scanner for sensing, which is too heavy to be used on our small quadrotor MAV.

For *motion planning*, there are typically two categories: optimization methods and discrete or randomized search methods. Optimization methods try to minimize some objectives, such as minimal traversal time or control input energy subject to the differential motion constraints and collision avoidance. Differential dynamic programming (Mayne, 1966) is a typical optimization approach. More recently, covariant gradient optimization (CHOMP) (Ratliff et al., 2009) achieved a faster performance by using a covariant gradient technique. It can operate in discrete occupancy-grid maps, while conventional numerical optimization requires the analytical representation of obstacles. For a special nonlinear system that is differentially flat (Murray, 2009; Murray et al., 1995), such as a quadrotor, optimization can be performed efficiently in the *flat output space* instead of the original configuration space. In this space, the states and control inputs of the system can be mapped analytically to the flat output variables and their derivatives. Therefore, during optimization, we do not need to explicitly consider the motion constraints. If we represent the flat output variables as a sequence of differential trajectories such as polynomials or B-splines, optimization over the spline parameters could be efficiently solved by analytic or numerical methods such as quadratic programming (Mellinger & Kumar, 2011). Bry *et al.* applied this method to an aggressive flight of aerial vehicles (Bry, Richter, Bachrach, & Roy, 2015). As the second category, search-based approaches usually discretize the trajectories as feasible action spaces, and then search a sequence of actions using a graph representation such as A*. This idea has been adapted to different styles (Likhachev et al., 2008) and widely used in autonomous vehicles (Ferguson et al., 2008). The optimality of these grid search algorithms is guaranteed up to the grid resolution, but the complexity grows exponentially with the dimensionality of state space. Therefore, it is unsuitable for high-dimensional planning problems such as MAV with 12 states. Different from A*, randomized search methods such as RRT* (rapidly exploring random tree) randomly sample nodes in configuration space and then search the path using the tree. They have been proven to have probabilistic completeness and asymptotic optimality in the limit of infinite samples (Karaman & Frazzoli, 2011), and they are effective for high-dimensional state spaces, which were used for MAV navigation by Shen et al. (2011).

3. SYSTEM OVERVIEW

To make our robot compact and have a high payload, our MAV is designed based on a quadrotor configuration. Due to the space constraints presented in the shipboard environment (the doorway is only 66 cm wide), we need a MAV that is as small as possible while still being able to carry the sensor payload. A picture of our system is shown in Figure 2(a). It is a quadrotor micro aerial vehicle with a size of 58 cm×58 cm×32 cm and a maximum payload of about 600 g.

There are two onboard computers. One is an ARM-based embedded computer (Odroid XU3, about 70 g) that is responsible for high-level task processing, such as odometry estimation, localization and motion planning, etc. The other is a Pixhawk Flight Controller Unit (FCU, about 38 g), which is used for multisensor data fusion and real-time control. A forward-looking RGB-D camera (Asus Xtion, about 100 g after being stripped down) is used for pose estimation and motion planning. A downward-looking optical flow camera (PX4Flow, about 18 g) is used for velocity estimation. Lastly, a point laser (SF02 Laser Rangefinder, about 69 g) and the sonar (HRLV-MaxSonar-EZ4) on PX4Flow are used for height estimation. Furthermore, a thermal camera (FLIR-tau, about 200 g) is used for fire detection. The forward-looking infrared (FLIR) camera has the ability to see through smoke conditions while traditional visible light cameras fail. We also carry our own inertial measurement unit (MicroStrain 3DM-GX3-35, about 23 g) to help estimate the pose of the vehicle.

To get robust, low-delay state estimates, a two-layer estimation architecture is designed, as shown in Figure 2(b). The first-layer pose estimation algorithms that estimate the 6DoF pose runs on the embedded computer, including odometry estimation and localization both running at 15 Hz. The second-layer state estimation module estimates the 6DoF pose and velocity by fusing the estimates from odometry, localization, optical flow, and raw data from sensors (IMU, sonar, and point LIDAR) on the FCU, which runs at 50 Hz to enable accurate position control. This redundant system improves the robustness and safety of our MAV. Even if some of the sensors do not work in certain challenging environments, the whole system can still work. The two layers communicate with each other through a serial interface. The software running on the embedded computer is developed based on the ROS middleware. The outputs (x, y, z, ψ) of the local planner (running on the embedded computer at 2 Hz) and the high-frequency state estimates (position and velocities) from the FCU are given to a hierarchical control approach for fast and accurate control. It is composed of an outer position controller running at 100 Hz and an inner attitude controller running at 250 Hz.

4. REAL-TIME STATE ESTIMATION ALGORITHMS

Real-time and robust state estimation is critical for tasks that require precise control, such as indoor flight, obstacle avoidance, and autonomous landing. In this paper, we first propose a fast and robust odometry estimation method that can work in different indoor environments. Then, we fuse the odometry with an IMU using an unscented Kalman filter (UKF), which can give us much faster and smoother velocity estimates. After that, we feed the odometry into a particle filter localization algorithm, which can estimate the absolute 6DoF state in a given 3D map, and we track it in real time during the mission. Finally, we fuse odometry, localization, and other sensor information in a KF framework to get very robust and fast state estimates for real-time control.

4.1. Real-time Odometry Estimation

In this paper, a fast depth odometry is proposed to calculate the frame-to-frame motion estimation. Our method is motivated by the pioneering work of Horn (Horn & Harris, 1991). It works directly on the depth image without detecting any features, which is much faster than state-of-the-art ICP-based methods (Pomerleau et al., 2013).

Let a 3D point $P = (X, Y, Z)^T$ (measured in the depth camera's coordinate system) be captured at pixel position $p = (x, y)^T$ in the depth image Z_t. This point undergoes a 3D motion $\Delta P = (\Delta X, \Delta Y, \Delta Z)^T$, which results in an image motion Δp between frames t_0 and t_1. Given that the depth of the 3D point will have moved by ΔZ, the depth value captured at this new image location $p + \Delta p$ will have consequently changed by this amount:

$$Z_{t+1}(p + \Delta p) = Z_t(p) + \Delta Z. \tag{1}$$

This equation is called a *range change/flow constraint equation* (Horn & Harris, 1991; Spies et al., 2002; Yamamoto et al., 1993). Taking the first-order Taylor expansion of the term $Z_{t+1}(p + \Delta p)$ in Eq. (1), we can obtain

$$\begin{aligned} Z_{t+1}(p + \Delta p) &= Z_{t+1}(p) + \nabla Z_{t+1}(p) \cdot \Delta p \\ &= Z_t(p) + \Delta Z, \end{aligned} \tag{2}$$

where $\nabla Z_{t+1}(p)$ is the gradient of the depth image $\nabla Z_{t+1}(p) = (Z_x, Z_y)$.

For a pinhole camera model, any small 2D displacement Δp in the image can be related directly to the 3D displacement ΔP, which gave rise to it by differentiating the perspective projection function with respect to the components of the 3D position P,

$$\frac{\partial p}{\partial P} = \frac{\Delta p}{\Delta P} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -X\frac{f_x}{Z^2} \\ 0 & \frac{f_y}{Z} & -Y\frac{f_y}{Z^2} \end{bmatrix}, \tag{3}$$

where f_x and f_y are the normalized focal lengths.

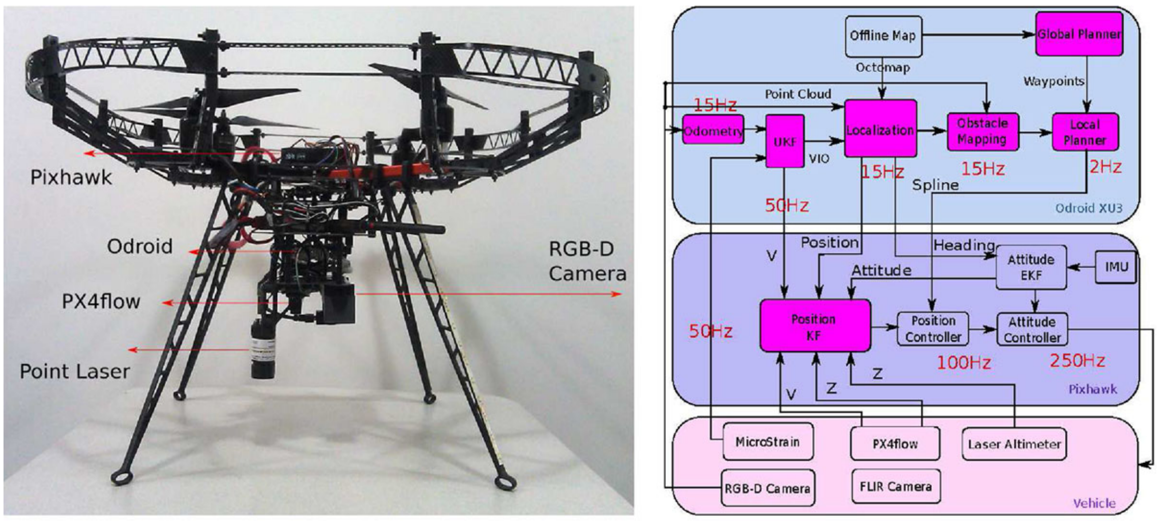


Figure 2. (a) Customized micro aerial vehicle platform. (b) Overview of the entire autonomous system. The high-level software includes odometry estimation, localization, planning, and obstacle avoidance running on an embedded computer. The low-level software includes state estimation, trajectory controller, and attitude controller running on the flight controller unit (FCU). The pink blocks are described in detail in this paper.

Under a small rotation assumption, if the sensor moves with instantaneous translational velocity $v = (v_x, v_y, v_z)^T$ and instantaneous rotational velocity $\omega = (\omega_x, \omega_y, \omega_z)^T$ with respect to the environment, then the point P appears to move with a velocity

$$\begin{aligned}\frac{dP}{dt} &= \Delta P = -v - \omega \times P \\ &= \begin{bmatrix} 0 & -Z & Y & -1 & 0 & 0 \\ Z & 0 & -X & 0 & -1 & 0 \\ -Y & X & 0 & 0 & 0 & -1 \end{bmatrix} \xi \end{aligned} \tag{4}$$

with respect to the sensor, where $\xi = [\omega_x, \omega_y, \omega_z, v_x, v_y, v_z]^T$. Substituting Eqs. (3) and (4) into Eq. (2), we can obtain

$$\underbrace{(Z_x, Z_y, -1) \begin{bmatrix} \frac{f_x}{Z} & 0 & -X\frac{f_x}{Z^2} \\ 0 & \frac{f_y}{Z} & -Y\frac{f_y}{Z^2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -Z & Y & -1 & 0 & 0 \\ Z & 0 & -X & 0 & -1 & 0 \\ -Y & X & 0 & 0 & 0 & -1 \end{bmatrix}}_{A} \xi$$

$$= Z_t(p) - Z_{t+1}(p), \tag{5}$$

which can be rewritten as

$$A\xi = Z_t(p) - Z_{t+1}(p). \tag{6}$$

This equation generates a pixel-based constraint relating the gradient of the depth image ∇Z_{t+1} and the temporal depth difference to the unknown camera motion ξ. If there are n pixels in the image, then we can get n such equations with only six unknowns. Here, we use a least-squares error minimization technique to solve the set of equations. After we get the rotational velocity $\omega = (\omega_x, \omega_y, \omega_z)^T$ and translational velocity $v = (v_x, v_y, v_z)^T$, the rigid body motion ΔT between t and $t+1$ can be calculated directly using the exponential map (Murray, Li, & Sastry, 1994):

$$\exp : \mathfrak{se}(3) \longrightarrow SE(3); \xi \longmapsto \Delta T, \tag{7}$$

$$\Delta T = e^{\hat{\xi}}, \tag{8}$$

where $\hat{\xi}$ is known as twist.

By computing the relative transformations ΔT from the depth images Z_t and Z_{t+1} and then concatenating the transformations, we can recover the full trajectory of the camera.

In practice, since least-squares estimators are particularly sensitive to outliers, we need to remove the potential outliers to get a robust and correct estimation. Therefore, pixels where the difference in depth between current and old images exceeds 50 mm are excluded from the estimator. In addition, all the edge points and isolated sparse points are removed from the estimator. Furthermore, in order to improve the computation speed, the depth image is uniformly downsampled to 80×60 for the final estimation. It should be noted that in our system, in order to improve the efficiency, no keyframe, image pyramid, or local bundle adjustment techniques are used to reduce the local drift. We will show later in our experiments that even without those techniques, our method still works very well in various environments.

4.2. Visual-inertial Odometry

To conserve CPU resources, we set the sampling rate of RGB-D images at 15 Hz. Therefore, our visual odometry outputs the transform at 15 Hz, and this estimation can be fed into FCU's position KF to fuse with other data for pose control. However, we found that only using the visual odometry is not smooth and fast enough for the KF filtering. The latency makes the filtering not smooth sometimes. In addition, because only depth images are used in odometry estimation, in places where there are not enough geometric constraints the odometry will suffer from degeneration, which will lead to incorrect estimation. To ameliorate those problems, we mount a Micro-Strain IMU, then use a loosely coupled UKF to fuse the odometry and IMU information, and feed the fused result to the FCU's position KF. Our filter robustly fuses measurements from visual odometry and IMU. The UKF is chosen based on the demonstrated improvements over the extended Kalman filter (EKF) for better estimation and filter consistency of nonlinear systems. The multisensor fusion framework is presented in detail in our previous work (Chambers et al., 2014). By doing so, our system became more stable than without the fusion. The possible reasons are as follows: (i) by fusing IMU and visual odometry, we can get a smoother odometry estimation; (ii) the latency is dramatically reduced when using the UKF output because our UKF runs at 50 Hz.

4.3. Real-time 6DoF Localization

From Section 4.2, we can get a robust odometry estimation, however it will definitely drift after a long running time. Although techniques such as keyframe, local map, or local bundle adjustment can reduce local drift, those techniques cannot reduce the global drift if there is no loop closure. To get an accurate absolute pose in the environment, we need a localization algorithm to locate the robot in a given 3D environment. In this paper, we propose a particle-filter-based localization algorithm to robustly localize our MAV in a given 3D octomap (Hornung et al., 2013; Maier et al., 2012). Although particle filter localization has been used successfully on ground mobile robots, the 6DoF pose $(x, y, z, \phi, \theta, \psi)$, which needs to be estimated for MAVs, increases the complexity of the problem. Here, we show that by designing an efficient observation model, this method can work very well on an embedded computer. For a general idea of a particle filter, readers are referred to Thrun, Fox, & Burgard (2000) for a detailed description.

4.3.1. Motion Model

For each subsequent frame, we propagate the previous state estimate according to the motion model $p(S_t|S_{t-1}, u_t)$ using the odometry ΔT_t computed by the fast range flow odometry proposed in Section 4.1. For each dimension, the propagation equations are of the form

$$S_t = S_{t-1} + \Delta T_t + e_t, \qquad e_t \sim N(0, \sigma^2), \tag{9}$$

where the final term in each equation adds a small amount of normally distributed noise so as to support unexpected motion.

4.3.2. Observation Model

In this section, we describe how to evaluate roll-θ_t, pitch-ϕ_t, height-z_t, and x, y, yaw-ψ_t in detail. The reason we estimate the 6DoF state, instead of just x, y, z, ψ while getting the roll and pitch from the IMU, is that our MAV works in a shipboard indoor environment that is not always level. Therefore, the roll and pitch angle from the IMU is not the true pose of the MAV inside the ship. In our system, one observation O_t is one frame of point cloud generated from the depth image. The belief of the MAV's 6DoF state is updated by using two different pieces of sensor information contained in one observation O_t. First, the belief is updated based on the ground plane information detected from the point cloud. Since our MAV works in indoor environments, most of the time it can see the ground plane. The ground plane is detected to get the roll $\tilde{\theta}_t$, pitch $\tilde{\phi}_t$, and height $\tilde{z}_t$ measurements to evaluate each particle's weight. Second, the belief is again updated based on the range measurements d_t contained in the same point cloud. Since the ground plane has no contribution for determining the x, y, and yaw, the ground plane is filtered out when updating the particle's position weight using range measurements. The final observation model is

$$\begin{aligned} p(O_t|S_t) &= p(d_t, \tilde{z}_t, \tilde{\phi}_t, \tilde{\theta}_t|S_t) \\ &= p(d_t|S_t) \cdot p(\tilde{z}_t|S_t) \cdot p(\tilde{\phi}_t|S_t) \cdot p(\tilde{\theta}_t|S_t). \end{aligned} \tag{10}$$

The formulation of the likelihood function $p(*|S_t)$ is defined by the Gaussian distribution:

$$\rho(e_r, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{e_r^2}{2\sigma^2}\right), \tag{11}$$

where σ is the standard deviation of the sensor noise and e_r is the error between predicted and measured values.

First, we detect the ground plane and use the ground plane information to update the belief of the state. To improve computation speed, the original input point cloud is uniformly downsampled to get the sparse point cloud C_t. Then, point cloud C_t is segmented into ground point cloud C_t^g and nonground point clouds C_t^w by using a RANSAC-based method (Fischler & Bolles, 1981). We assume that the ground plane is the biggest and furthest plane to the MAV and the closest to horizontal. After detecting the ground plane, roll, pitch, and height values can be easily computed from the ground plane equation. Then, the weight of each particle is updated according to the observed measurements and predicted measurements by using the following equations:

$$p(\tilde{z}_t|S_t) = \rho(z_t - \tilde{z}_t, \sigma_z),$$
$$p(\tilde{\phi}_t|S_t) = \rho(\phi_t - \tilde{\phi}_t, \sigma_\phi), \quad (12)$$
$$p(\tilde{\theta}_t|S_t) = \rho(\theta_t - \tilde{\theta}_t, \sigma_\theta),$$

where $\tilde{z}_t$, $\tilde{\phi}_t$, and $\tilde{\theta}_t$ are calculated from the detected ground plane, and σ_z, σ_ϕ, and σ_θ are determined by the noise characteristics of the ground plane.

Second, in order to evaluate the depth-sensing likelihood $p(d_t|S_t)$, we use a sparse subset of beams from the point cloud C_t. From our experiment, we found that how one selected the subset of beams really influences the robustness, efficiency, and accuracy of the localization algorithm. To efficiently use the points with the most constraints, we tried two ways to select points. First, since the ground point cloud C_t^g has little importance for determining the x, y, and yaw of the MAV, only very few points C_t^u from the ground part are selected by using the uniform downsampling. For the nonground part point cloud C_t^w, we found that most of the time in indoor environments, especially in long corridors, there are only a few points on the wall and the ceiling that are useful for determining the forward translation. If we use a uniform downsampling, then we will miss this valuable information. To use this information, we try to select those points C_t^n by using a normal space sampling method (Rusinkiewicz & Levoy, 2001). By doing so, we can select those points with the most constraints. Finally, we form points C_t^u selected from the ground part and points C_t^n from the nonground part to get the final filtered point cloud C_t^f. An illustrative picture is shown in Figure 3.

We assume that the sampled measurements are conditionally independent. Here, the likelihood of a single depth measurement $d_{t,k}$ depends on the distance d of the corresponding beam end point to the closest obstacle in the map:

$$p(d_{t,k}|S_t) = \rho(d, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{d^2}{2\sigma^2}\right), \quad (13)$$

where σ is the standard deviation of the sensor noise and d is the distance. Since a point cloud measurement consists of K beams $d_{t,k}$, the integration of a full scan is computed as the product of the each beam likelihood:

$$p(d_t|S_t) = \prod_{k=1}^{K} p(d_{t,k}|S_t). \quad (14)$$

In practice, the depth values of the RGB-D camera are very noisy when the measurement range is bigger than 4 m. To include this characteristic in our observation model, we use a changing σ, which increases with the measurement distance.

4.4. State Estimation on the Flight Controller Unit

For onboard planning and control, correct and robust estimate of the position and orientation of the platform is required. We fuse measurements from all the sources on the FCU itself to output a high-frequency state estimate. We run a very high rate attitude EKF estimator, which stabilizes the platform's angular motion. The heading is generally corrected using a magnetometer, but it would fail inside a ship, which is a metal body, hence we use gyroscopes to estimate the heading rate and correct the absolute heading using the output of the localization module running on an onboard embedded computer.

To achieve this, we have developed a robust state estimation algorithm on FCU. We used Pixhawk's original EKF for attitude estimation with modifications to use emulated magnetometer readings generated using heading estimates from the localization module. For position and velocity estimation, we changed from the original pixhawk's inav filter to a Kalman filter (KF). The algorithm leverages on real-time performance of a loosely coupled KF. By "loosely coupled," we mean that the attitude estimation is done by the EKF running on FCU, which is fed to the Kalman filter to estimate the final odometry explained in the following paragraph. Fusing data from various sensors allows us to keep track of our position even when one sensor fails but another has some qualitative estimate. All the inputs of the Kalman filter are shown in Figure 2(b). This estimate is used to update the position of the platform in a coordinate system independent of the global coordinate system. The FCU converges with the absolute pose estimate in a global frame as it arrives from the localization module running on the onboard embedded computer. In this way, there are no abrupt changes in the pose estimated on pixhawk required for a robust position control.

We use the KF only for position, velocity, and acceleration estimation because we can get the roll, pitch, and yaw estimates from the EKF attitude estimator and then transform all the input data of the KF to the global frame. Therefore, all the information is in one single frame, and because there are no angular corrections required, the filter becomes a linear problem that can easily be solved using a Kalman filter (Thrun, Burgard, & Fox, 2005). To estimate the final position, velocity, and acceleration for control, we fuse velocities from a downward-facing PX4Flow camera and visual-inertial odometry from the UKF running on the onboard embedded computer. The reason we used a UKF for velocity estimation on the Odroid embedded computer is that the KF running on the FCU needs fast and real-time measurements, and default VO output is slow. Our KF did not have a measurement buffer to sync the times of all the measurements. Measurements were fused as they arrived. Therefore, we used a UKF on the onboard computer to send out faster VO estimates using an accurate MicroStrain IMU, which solved the problem of erroneous fusion of delayed

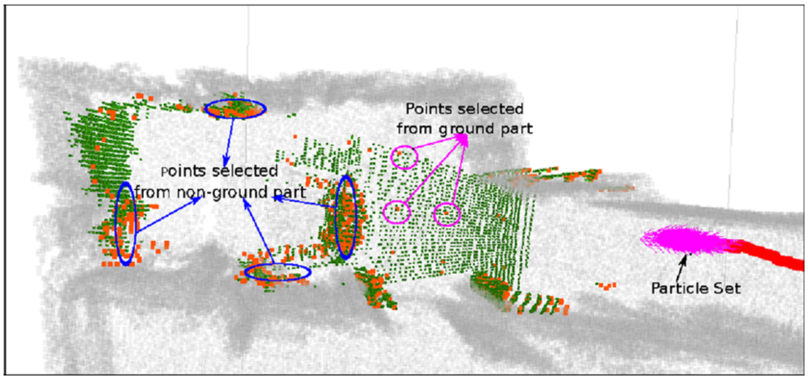


Figure 3. Filtered point cloud C_t^f for the depth-sensing likelihood updating. The robot's pose is denoted by the particle set. The filtered point cloud is composed of two parts. The first part points are selected from the ground plane using the uniform downsampling method. The second part points are those with the most constraints selected from the nonground part by using normal space sampling. Green points are the original downsampled point cloud C_t, and brown points are the final filtered points C_t^f with the most constraint information. As one can see from the picture, many points of door frames and protruding objects in the corridor are selected, while very few points from the ground are selected.

measurements. This KF also fuses absolute positions from the localization module with height measurements from the point LIDAR and the PX4flow sonar sensor. We chose to use LIDAR and sonar sensors because it is difficult to use a sonar sensor alone for height estimation. This difficulty stems from the fact that in constrained spaces, the sonar sensor would sporadically give erroneous measurements due to bounces from close walls or clutter along the corridors, leading to erroneous height estimates. To solve this problem, we added the point LIDAR sensor. The LIDAR has a measurement range from 0.1 to 40 m. However, the LIDAR sensor has worse resolution (1 cm) compared to the sonar sensor (1 mm). Hence we used both.

5. REAL-TIME MOTION-PLANNING ALGORITHMS

A motion-planning module needs to efficiently detect and avoid obstacles to maintain the safety of the vehicle. Even when the prior map for localization is accessible, it is only used in global path planning to generate some mission points for navigation in order to adapt to the unknown obstacles and environmental changes, and these points could also be provided by a human (Shen, Michael, & Kumar, 2013). Therefore, an online obstacle mapping and avoidance algorithm is designed to generate a local collision-free trajectory based on the mission points and the real-time sensor data independent of the prior map.

5.1. Global Path Planning

The global planner is called upon at the beginning or when the local planner gets trapped in local dead ends. It is composed of the following three steps:

(i) Generate a 2D grid map from a 3D point cloud. Since our MAV operates mainly in corridor environments, a 2D map is enough to provide high-level mission points with much faster processing speed. We manually remove the ceiling and ground using a height threshold, and we project all the 3D points onto the 2D grid map.

(ii) Build a Voronoi diagram of the grid map using the incremental brushfire algorithms (Kalra, Ferguson, & Stentz, 2009). This reduces the latter A* to search only on the Voronoi graph instead of the entire workspace.

(iii) Search the optimal path using A* on the 2D position x, y state space. Smoothing and collision checking are implemented later to get a feasible and smooth path, as shown in Figure 4(a). Finally, the path is downsampled to generate a series of mission points about 5 m away from each other.

5.2. Obstacle Mapping

An online updated 3D occupancy map (Scherer et al., 2012) is built to represent the local world for obstacle avoidance. It stores the probability of each grid being occupied and is updated according to the range sensor data from the RGB-D camera. If the probability exceeds a threshold, the grid would be considered as obstacles. Denote $b(m)$ as the belief of a grid m. Then the belief update under the sensor observation o is by

$$b(m|o) = b(m) + k * 20, \quad (15)$$
$$b(m|\bar{o}) = b(m) - k * 5.$$

The first equation indicates that if the sensor ray hits the cell, it increases the probability of being obstacles, and *vice versa* for the second equation. A detailed derivation of the update rule can be found in Scherer et al. (2008).

A scaling number $k \in [0, 1]$ is used to represent the quality of observation o, aiming to increase the accuracy of the occupancy estimate and reduce the unnecessary updates due to localization uncertainty. It is based on the following two components: (i) The uncertainty of state estimation. If the state estimation is of high quality with low uncertainty, k is set close to 1. The uncertainty is measured by the determinant $|Q|$ of the pose covariance matrix Q, which is computed from the particle distribution in the localization. We linearly rescale $|Q|$ to $k_1 \in [0, 1]$. (ii) The distance d of the grid to the vehicle. If d increases, sensor data are likely to be inaccurate. We linearly rescale d to $k_2 \in [0, 1]$. This is reasonable for an Asus RGB-D camera since its reliable range measurement is about 3.5 m. Finally, k is set as $k = k_1 k_2$. To keep memory and computation efficient, the occupancy map keeps a fixed dimension and moves along with the vehicle. The obstacle grid in the occupancy map is then used to compute a 3D distance map, which stores the distance of each grid to its closest obstacles, as shown in Figure 4(b).

5.3. Local Motion Planning

Given the 3D obstacle distance map and the mission point from the global planner, the local planner will generate an optimal dynamic feasible and safe trajectory in real time. The objectives include obstacle clearance, control input energy, execution time, and so on. As outlined in Section 2, searching algorithms such as RRT* or A* suffer from the curse of dimensionality, and they may take several minutes to converge (Bry et al., 2015). Utilizing the *differential flatness* property of the quadrotor (Mellinger & Kumar, 2011), the planning problem can be changed to optimize the polynomial splines of the *flat output* variable, in our case x, y, z, ψ(yaw). However, it is still intractable to directly optimize the high-dimensional polynomial coefficients subject to dynamic constraints using nonlinear optimization. Therefore, the problem is simplified into two steps: optimal path planning without considering dynamic constraints, and optimal dynamic trajectory generation through waypoints. This technique is widely used for computational efficiency [see Boeuf, Cortés, Alami, & Siméon (2014) and Richter, Bry, & Roy (2013)]. The closest work to ours is that of Richter et al. (2013) and Bry et al. (2015), which combines RRT* with polynomial trajectory generation and is also applied to a real MAV flight. The difference is that we use CHOMP (Ratliff et al., 2009) optimization instead of RRT*, and we utilize different polynomial trajectory generation methods. In addition, their trajectories are computed offline while our method can run at 30 Hz online, as shown in Section 7.3. The following two sections describe *path planning* and *trajectory generation*, respectively.

5.3.1. Path Planning

Compared to the search algorithms such as RRT* and A*, an optimization-based planner such as CHOMP usually generates smoother paths because it explicitly optimizes the smoothness term. However, CHOMP may get stuck in the local minima due to a bad initial guess. To efficiently generate a good initial guess, we use receding horizon control (RHC) to select the best path from an offline path library as the initial guess (Dey et al., 2015; Green & Kelly, 2007). We further modify the standard CHOMP to an end-point-free optimization because the end point of the initial path selected from the library may be close to obstacles and not optimal.

The path is composed of a series of waypoints. Each waypoint has 4DoF $\{x, y, z, \psi(\text{yaw})\}$, namely the *flat output* space of the quadrotor (Mellinger & Kumar, 2011). Let the path be $\xi : [0, 1] \mapsto \mathbb{R}^4$ mapping from time to 4DoF such that

$$\min_{\xi} \quad J(\xi) = w_1 f_{\text{obst}}(\xi) + w_2 f_{\text{smooth}}(\xi) + f_{\text{goal}}(\xi) \quad (16)$$

$$\text{s.t.} \qquad \xi(0) = \xi_0,$$

where w_1, w_2 are the weighting parameters. So the initial state is fixed while the ending state is freed.

$f_{\text{obst}}(\xi)$ is the obstacle cost function as defined in CHOMP:

$$f_{\text{obst}}(\xi) = \int_0^1 c_{\text{obs}}[\xi(t)] \left\| \frac{d}{dt}\xi(t) \right\| dt, \quad (17)$$

where $c_{\text{obs}}[\xi(t)] = \| \max\ (0, d_{\max} - d[\xi(t)])\|^2$. $d_{\max}$ is the maximum distance upon which obstacle cost is available, and $d[\xi(t)]$ is the distance to obstacles from the distance map.

$f_{\text{smooth}}(\xi)$ measures the smoothness of the path and penalizes the high derivatives:

$$f_{\text{smooth}}(\xi) = \frac{1}{2}\int_0^1 \left\| \frac{d}{dt}\xi(t) \right\|^2 dt. \quad (18)$$

$f_{\text{goal}}(\xi)$ is the cost-to-go heuristic measuring path end-point distance to the local goal point ξ_g:

$$f_{\text{goal}}(\xi) = \|\xi(1) - \xi_g\|^2. \quad (19)$$

The path ξ is discretized into 16 waypoints that are optimized together using CHOMP based on the cost function in Eq. (16).

The initial path library S of RHC contains 27 specially designed paths, as shown in Figure 5, including three categories: straight flight, turning, and route shifting. We did not utilize the maximum dispersion library of Green *et al.* (Green & Kelly, 2007) as they assume an environment with random obstacle configurations. Dey et al. (2015) utilized

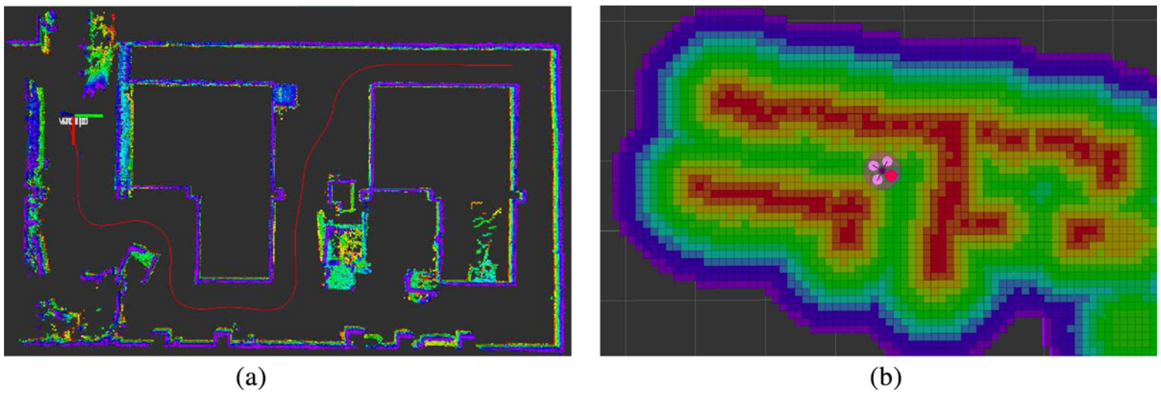


Figure 4. (a) Global path planning using A* search. (b) Distance map. The redder pixels are closer to obstacles with higher costs. The purple circle represents the quadrotor.

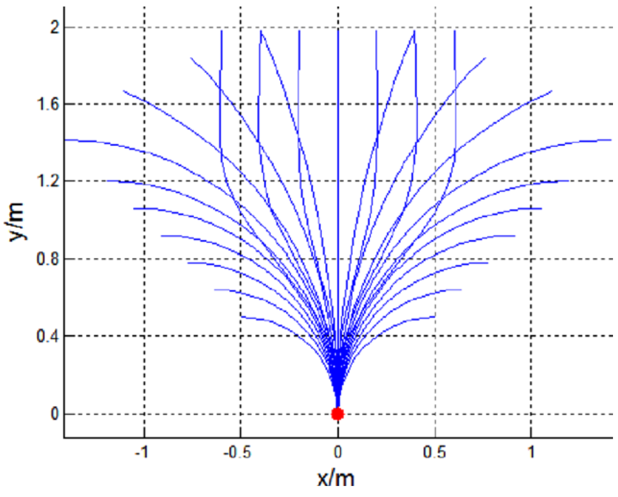


Figure 5. Initial path library. All the paths start at (0,0). The library is manually designed for the corridor environment where obstacles usually lie on two sides. It includes a straight line, turning arcs with different curvatures, and route-shifting curves with a parallel ending direction, corresponding to the three main flight modes in the corridor.

this library for a densely cluttered forest flight. For our corridor flight, however, obstacles usually lie on two sides, so only small amounts of initial paths are needed. The initial paths are around 2 m long within a short horizon, making the latter optimization faster and more reactive.

We first align the offline path library with the lookahead planning pose, and then we select the best path $\xi_0 = \arg\min_{\xi \in S} J(\xi)$ as an initial guess and optimize it using the above-defined cost functions. Although the initial path is in 2D x, y space, the optimization is performed in 4D x, y, z, ψ(yaw). For z, we add an additional cost of being close to the reference height, because maintaining height stability is beneficial for state estimation, such as for the optical flow sensor. Since f_{obst} and f_{goal} only depend on the position x, y, z while independent of ψ(yaw), we directly set the ending state's yaw as the initial path's final tangent direction and optimize the other states' yaw using f_{smooth}. An optimization example during turning is shown in Figure 6(a), where the gradient pushes the path including the ending state away from obstacles. The average cost per iteration of J in Eq. (16) during turning is shown in Figure 6(b), demonstrating the convergence of optimization.

5.3.2. Trajectory generation

After getting the path waypoints $\xi_0, \ldots, \xi_n$ of $\{x, y, z, \psi(\text{yaw})\}$, we need to generate a continuous trajectory with time profile through them so that we can use the velocity, acceleration, and higher-order derivatives to compute feedforward control input for the quadrotor (Mellinger & Kumar, 2011), which guarantees the exponential tracking stability of the controller. The polynomial spline is chosen as the basis function due to its analytic and computational tractability.

As suggested by Mellinger & Kumar (2011), snap, a fourth-order derivative (with respect to time), is related to the changes in the motor commands, which should be continuous and minimized for quadrotors. In our case, the waypoints from path planning are downsampled into five segments so the trajectory $\xi(t)$ is parametrized as five segments of sixth-order polynomials to ensure continuity of up to fourth derivative through the whole trajectory. In theory, higher-order polynomials could increase the degrees of freedom so as to get better optimization, but they may also increase computation complexity and are more likely to become ill-conditioned (Bry et al., 2015). The kth segment $1 \leq k \leq 5$ of the trajectory is expressed as

$$\xi(t) = \sum_{i=0}^{6} p_{ik} t^i, \quad t_{k-1} \leq t < t_k. \tag{20}$$

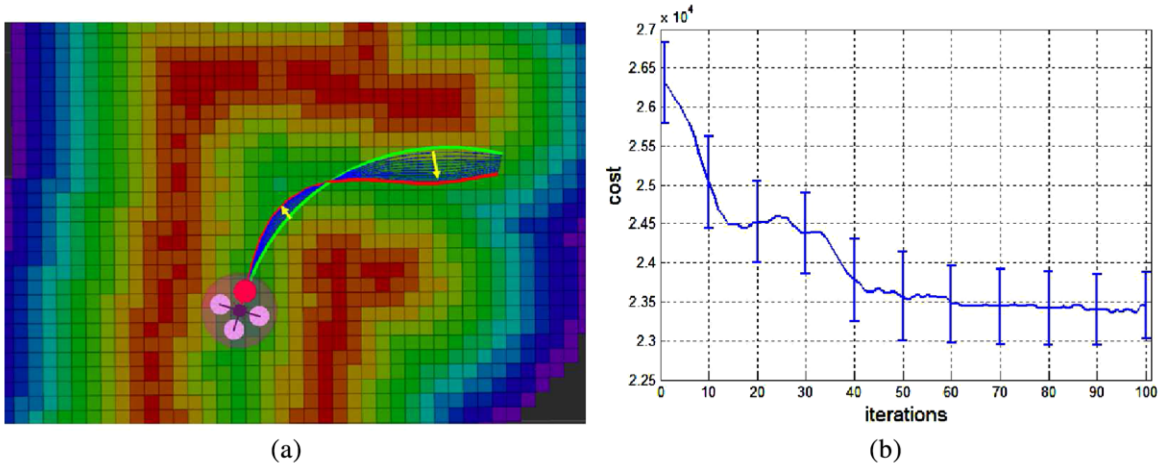


Figure 6. (a) Path optimization in turning. The color grid represents the distance map. The green curve represents the initial best path, while blue curves are the paths during optimization based on the gradient (yellow). The final optimized path is in red. (b) Cost iterations of path planning in a turning scenario.

(i) Minimize snap: The integration of snap to be minimized is defined as

$$f_{\text{snap}}(\xi) = \int_{t_0}^{t_n} \left\| \frac{d^4\xi(t)}{dt^4} \right\|^2. \tag{21}$$

There are two kinds of constraints for optimization. (i) The *equality* constraint imposed on the begin and end points of each segment, including passing through each waypoint and keeping derivatives continuous. (ii) The *inequality* constraint, including within the maximum velocity and the acceleration limits.

For each flat output variable, for example the x coordinate, it has five segments' trajectories and a total of 35 polynomial coefficients. We can stack the 35 coefficients as $p \in \mathbb{R}^{35}$ and rewrite the snap cost function and equality constraint in a matrix format. Then we obtain the quadratic programming problem:

$$\min_p \ p^T H p \qquad \text{s.t.} \ Ap = b, \quad Cp < d, \tag{22}$$

where H is the integration of the snap square. Therefore, H is a positive-semidefinite matrix. To obtain an analytical solution, we just consider *equality* constraints and assume that the time allocation of each segment is set based on a constant velocity model. A closed-form solution is found using Lagrange multipliers:

$$p = H^{-1}A^T(AH^{-1}A^T)^{-1}b. \tag{23}$$

Since the H matrix is sometimes ill-conditioned for inversion in practice, a regularization term εI is added to it to deal with the singularity problem:

$$H \leftarrow H + \varepsilon I, \tag{24}$$

where I is an identity matrix and ε is a small positive number. Here, it is set to 0.001. This Tikhonov regularization (Golub, Hansen, & O'Leary, 1999) provides an approximation to matrix inversion.

To deal with the *inequality* constraint, instead of the time-consuming nonlinear optimization through iterations, we check the *inequality* constraint after getting the polynomials from Eq. (23). If it is not satisfied, time duration is enlarged for the whole trajectory, and then Eq. (23) is solved again until the inequality constraint is satisfied. In practice, since our vehicle flies at low speed, the iteration process is usually not needed. A better method is proposed to optimize the time allocation, as follows.

(ii) Optimize time: Until this point, the time allocation for each segment has been fixed based on the constant velocity model, but actually it can be optimized to get a smaller snap while satisfying the *inequality* constraint. For example, if two waypoints are far away, then the segment should be allocated more time to avoid unnecessary acceleration and deceleration. So in this section, we optimize the time allocation $t = [\Delta t_1, \Delta t_2, \ldots, \Delta t_5]$ to minimize $w_3 f_{\text{snap}} + f_{\text{time}}$ in Eq. (25). There is no analytical solution to it, and the Levenberg-Marquardt method (Moré, 1978) is adopted to solve it through iterations,

$$f(t) = w_3 f_{\text{snap}} + f_{\text{time}} = w_3 p(t)^T H(t) p(t) + \sum_{i=1}^{5} \Delta t_i, \tag{25}$$

where $p(t)$ is computed analytically using Eq. (23). w_3 is the relative weight of snap cost compared to time length cost.

We change the cost to $F(t) = f(t)^2$ and compute Jacobian J numerically. Then the update of each iteration is

$$\begin{gathered}\{J^T J + \lambda[\operatorname{diag}(J^T J)]\}\delta = J^T[-f(t)] \\ t \leftarrow t + \delta.\end{gathered} \tag{26}$$

An example of time optimization is shown in Figure 7(a). Each spline is the minimum snap trajectory with some time allocations. After 10 iterations, the red trajectory becomes much smoother than the initial trajectory in blue. Total cost $f(t)$ iteration is shown in Figure 7(b).

For the inequality constraint $Cp < d$, we still need to check it after getting the optimized polynomials and time allocation from Eq. (25). If it is not satisfactory, we could increase the weight w_3 for the snap cost, which tends to enlarge the total time duration, and then optimize it again.

6. SMOKE AND FIRE DETECTION

The goal of this project was to detect and locate fire or smoke and alert the emergency response team based on this information. This involved detecting fire using onboard sensors and then determining its location using the vehicle's estimated pose inside the ship. We used a lightweight FLIR-tau thermal camera to measure the temperature of the environment. This provided us with a 640×480 pixel image with a pixel intensity corresponding to the temperature at the pixel location, as shown in Figure 8(a). We can segment the appropriate range of temperature for fire, people, etc. based on this image. Anything over $100\,^{\circ}$C is considered to have a high probability of fire or being situated very close to fire [e.g., Figure 8(b)]. Similarly, segmented blobs with temperature close to $30\,^{\circ}$C are considered to belong to human beings. Smoke detection was based on texture analysis of the color images received from the onboard RGB-D camera. The current approach is to detect the color homogeneity in the images. Therefore, the two deciding factors for smoke detection are the contrast and the average intensity of the image. Low contrast images represent a higher probability of smoke in the environment. The average intensity, or for this purpose the overall darkness of the image, provides the counterargument. A lack of illumination makes smoke detection imprecise due to the lower contrast in the image. Taking both of the above factors into account gives us an empirical idea of the smoke present in the environment.

Because this paper focuses mainly on autonomous navigation problems, we only used the aforementioned methods to detect fire and smoke. Those methods may not be very accurate and robust because only single pieces of information from thermal or color images are used. We believe that using all the information from thermal images, color images, and depth images can improve the robustness and accuracy of fire and smoke detection. In the future, we will investigate new fire and smoke detection algorithms and use the detected fire or smoke information to guide our MAV to search shipboards more efficiently.

7. REAL-WORLD EXPERIMENTS AND ANALYSIS

In this section, we evaluate the performance of our navigation system through various experiments. We first test the performance of some important modules separately using real-world datasets, and then we validate the performance of the whole system in a real shipboard environment under different environmental conditions. We develop our system using ROS Indigo, PCL 1.7, OpenCV 2.4, and C++.

7.1. Fast Odometry Experiments

We compare our range flow odometry method with other state-of-the-art methods using open-access benchmark datasets and author-collected datasets. Here, we compared the proposed fast range flow odometry method with Fovis[2] (Huang & Bachrach, 2011), DVO[3] (Kerl et al., 2013), and FastICP[4] (Pomerleau et al., 2013). We use the code that the authors published for the ROS platform. We use default parameters of each method for testing. We test the performance of each method from three aspects, namely robustness, runtime performance, and accuracy. As mentioned before, actually for our system we are not very concerned with the accuracy of the odometry estimation method because our localization algorithm will correct the drift of visual odometry. Compared to accuracy, robustness and efficiency in degraded visual environments are much more important for us. Therefore, we will first show the robustness and efficiency performance, and then the accuracy based on a benchmark dataset.

7.1.1. Robustness Validation in Challenging Environments

To evaluate the robustness, we recorded some datasets in challenging environments that are similar to shipboard environments. Although it is easy to obtain ground truth in outdoor (with a high-accuracy GPS) and small indoor environments (with a motion capture system, such as Vicon), it is very difficult to obtain accurate 6DOF ground truth in large indoor environments with many corridors. For this reason, the camera was started and stopped at the same position. Therefore, we can use closed-loop error to evaluate the estimation performance of each method to some extent. We define closed-loop error as the gap between the two ends of a trajectory output compared to the total length of the trajectory. We calculate the ratio of pixels in the image that contains a valid depth compared to the total number of pixels in the image to estimate the quality of the point cloud. We compute the average and standard deviation of the intensity values (0–255) of the RGB image as a measurement of the amount of light available in the image, which is a simple way to estimate the quality of an image. The data were recorded at 15 Hz, and sensor resolution was 640×480. The experimental results of two different environments are shown in Figure 9 and Table I.

[2]https://github.com/srv/fovis.git
[3]https://github.com/tum-vision/dvo_slam.git
[4]https://github.com/ethz-asl/ethzasl_icp_mapping.git

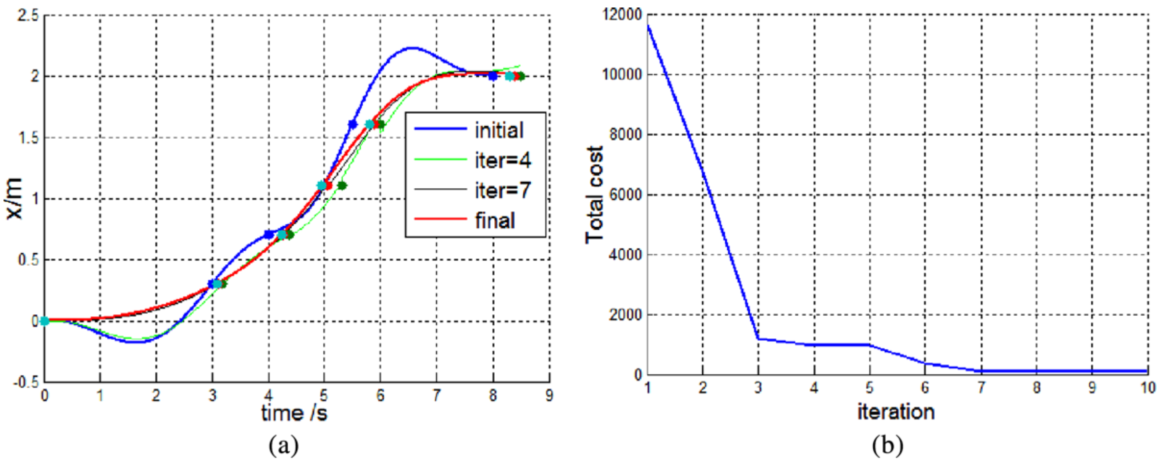


Figure 7. (a) Time optimization of minimum snap trajectory. All the trajectories pass through the same waypoints but with different time set points. The blue curve is the initial trajectory of raw time allocations. The red curve is the final trajectory with optimized time. (b) Cost iterations during time optimization.

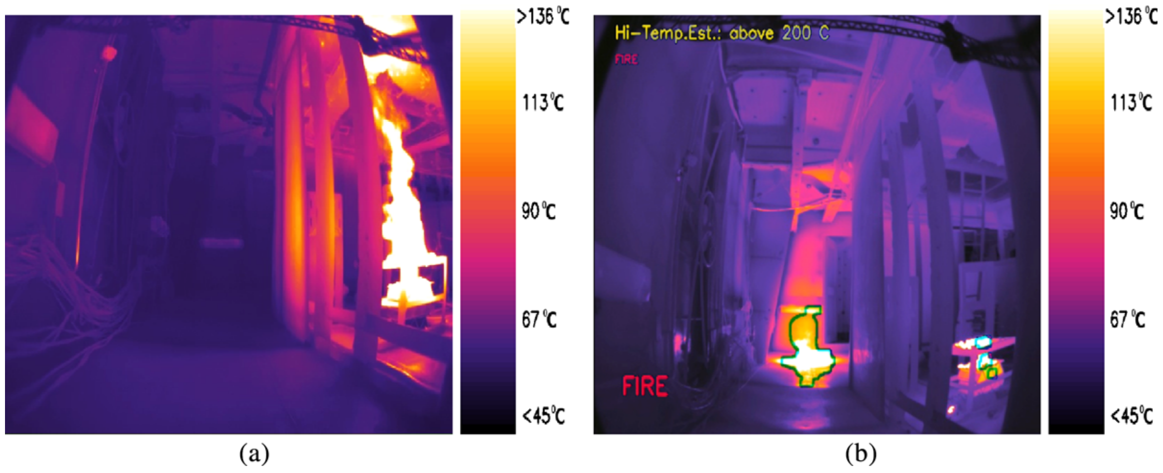
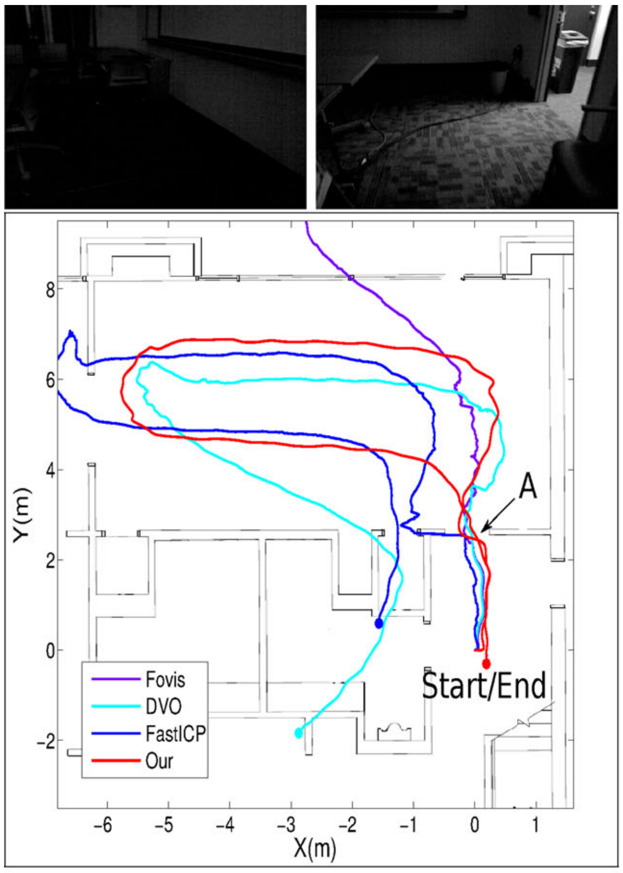


Figure 8. (a) A colorized image output from FLIR thermal camera (wood fire). (b) Fire segmentation based on temperature (oil and wood fire). The detected fire sources are outlined in green.

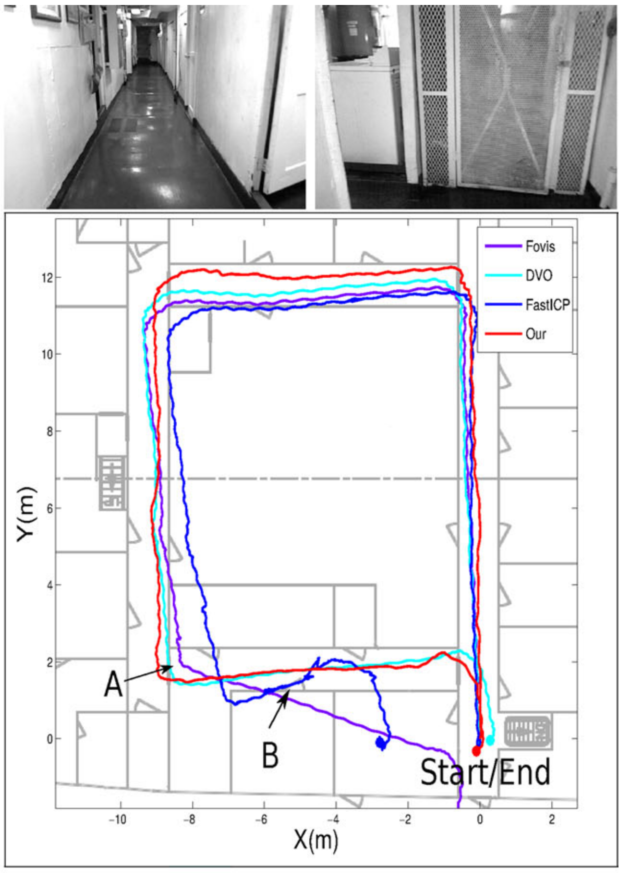
Table I. Closed-loop error.

Method	Dark room	Long corridor
Fovis	failed	7.4%
DVO	17.09%	**2.0%**
FastICP	7.03%	6.9%
Our	**4.59%**	2.50%

The first experiment was in a conference room containing a large desk and many chairs. In this experiment, dramatic illumination changes occurred when the robot entered and exited the conference room. The mean intensity changed from 169.2 to 3.5. The conference room was very dark where the intensity was just about 3.5, which is very challenging for visual-based methods. Fortunately, we could still get good point clouds in this environment, where the depth coverage ratio changed from 71.5% to 90.1%. There is no doubt that depth-based methods would be better than visual-based methods in this environment. In this experiment, we found that our method achieved the best performance. To our surprise, dense visual odometry could still work almost all the time except in very dark areas (the mean intensity is less than 10). It seems that DVO is much more robust than sparse feature-based methods in this test. However, even though depth-based methods achieved better results in this test, they both encountered a degeneration problem. As you can see around position A in Figure 9(a), both FastICP and our method slide far away from the true



(a)



(b)

Figure 9. Robustness evaluation in different environments. (a) Estimated trajectory in a low illumination conference room environment. (b) Estimated trajectory in a challenging long corridor of a shipboard environment.

Table II. Runtime performance.

	VGA on Laptop					QVGA on Odroid				
Methods	Mean	Min	Max	StdDev	Avg CPU	Mean	Min	Max	StdDev	Avg CPU
Fovis	20.3	10.8	47.9	4.5	13.5%	25.30	8.2	113.9	11.7	23.5%
DVO	52.4	20.1	242.8	11.2	22.6%					
FastICP	50.3	13.3	350.0	34.0	26.1%	77.03	15.7	262.4	34.5	45.6%
Our	**10.9**	**3.5**	**35.6**	**3.5**	**12.5%**	**15.9**	**5.9**	**36.5**	**3.9**	**13.8%**

position. This is a common issue for all geometry-based VO methods, like the ill-conditioned problem for visual-based methods in featureless environments. This can be associated with a phenomenon known as “sliding,” where the algorithm cannot determine where the best fit lies along a plane or line. Usually, other sensor information such as RGB can be incorporated to deal with the ill-conditioned problem. Unfortunately, our practical application environment is a visually degraded environment where RGB information is not available. Therefore, in our algorithm we do not use any RGB information. In our practical system, IMU information is used to ameliorate this problem.

The second experiment was in long corridors inside of a shipboard environment. In this experiment, the mean intensity of images changed from 164.5 to 59.4 and the depth coverage ratio changed from 90.9% to 79.3%. It seems that both RGB and depth information are very good. However, in this long corridor, the floors and walls are very smooth.

There are many places where only small objects on the wall or door frames can be used to estimate the translation. Therefore, this environment is very challenging for both visual and depth-based methods because there are only a few visual and geometric features. It can be seen that in this environment, DVO achieved the best result while Fovis failed (around place A) before the robot entered the last corridor section. It should be noted that Fovis does not output any estimation when the algorithm finds that there are not enough feature correspondences. The FastICP method also has a similar failure-detection technique. When the overlapping of two consecutive point clouds is too small, or not enough point correspondences are available, FastICP will not output any results. Fovis's failure is due to many repetitive textures around the corner A of the last corridor section, as shown in the top right picture in Figure 9, where Fovis failed to detect and track features when the camera turned quickly around that corner. The reason that DVO can succeed is that it depends on all of the image information other than sparse visual features, which are difficult to detect in this environment. Sometimes, even though some sparse visual features can be detected, they may be discarded due to the lack of corresponding depth value. The FastICP method also failed to estimate the translation in the last corridor section (around position B). The reason for this failure is that there are not enough constraints in the last corridor for the ICP method to estimate the translation. Our method achieved very good results, which were similar to those of DVO in this test. The reason that our method can achieve as good an estimation as DVO is that actually they are both direct motion estimation methods using whole image information.

7.1.2. Runtime Performance on an Embedded Computer

The computational performance is very important in our system, since our onboard computer has limited computational resources while all navigation modules should run onboard in real time. If the odometry is too slow or takes too much CPU time, then it is impossible to run other modules online, such as localization, mapping, and path planning. Our odometry algorithm is a single-threaded program. Therefore, it only takes one core. In our experiments, we tested the runtime performance on two computers by using two datasets. One dataset was recorded at a frame rate of 15 Hz with VGA resolution and tested on an Asus UX31E Ultrabook, which has an Intel 64-bit Quad-core 1.7 GHz CPU and 4 GB Memory. Another dataset was recorded at a frame rate of 15 Hz with QVGA resolution and tested on our onboard embedded computer Odroid XU3. This embedded computer is very compact and is based on a 32-bit ARM architecture. It has two processors: a Cortex−A7 quad core CPU and a Cortex−A15 quad core CPU. Each core has one thread. This embedded computer seems very powerful, but since it is based on ARM architecture its computation performance is far behind that of the x86 Intel CPU. For those two datasets, the testing results are shown in Table IX. It should be noted that since the DVO[4] uses many Intel SSE instructions to optimize the code, we failed to compile it on our ARM-based Odroid embedded computer. Therefore, we make use of the implementation[5] included in OpenCV and test it on the Odroid computer (the average processing time is around 65 ms for QVGA). From the experimental results, we can see that our method is the fastest and also consumes the least amount of CPU resources. FastICP cannot work in real time on the embedded computer. Although we failed to run the original DVO[4] on Odroid, from the experimental results of each method compared on the laptop and Odroid, we can deduce that DVO is also a bit difficult to run in real time on the Odroid embedded computer. Although Fovis also has very good speed and efficiency performance, unfortunately it cannot work in dark or visually degraded environments.

7.1.3. Accuracy Comparison using Benchmark Dataset

In this section, we use TUM RGB-D datasets[6] to test the estimation accuracy of each method because it can give us a very accurate ground-truth trajectory. The data were recorded at a full frame rate (30 Hz) and sensor resolution (640 × 480). The ground-truth trajectory was obtained from a high-accuracy motion capture system with eight high-speed tracking cameras (100 Hz). We use the relative pose error metric (Sturm et al., 2012) to measure the drift of the visual odometry system.

We used two datasets to compare the accuracy of each method. Actually, for the TUM datasets, it is not very suitable to compare our method to other methods because our method is designed for dark or light smoky environments and a computation-limited embedded system, while all the sequences provided by TUM datasets have very good illumination. Here, two datasets, freiburg2/desk and freiburg1/room, were selected for the experiments. For the freiburg2/desk sequence, the RGB-D data were recorded in a typical office scene with two desks, a computer monitor, chairs, etc. The Kinect was moved around two tables so that the loop was closed. The average translational and angular velocity were 0.193 m/s and 6.338 deg/s, respectively. The mean intensity changed from 73.1 to 153.8. However, the depth coverage changed from 85.3% to 53.9%. Therefore, the RGB information is good while the depth information is not very good. In the freiburg1/room dataset, the sequence was filmed along a trajectory through a typical office. It started with the four desks but continued around the wall of the room until the loop was closed. The depth

[5]https://github.com/Itseez/opencv/blob/2.4/modules/contrib/src/rgbdodometry.cpp
[6]http://vision.in.tum.de/data/datasets/rgbd-dataset

Table III. Translational and rotational errors of each method.

	freiburg2/desk				freiburg1/room			
	$\bar{x}$ (m)	σ	$\bar{\theta}$ (deg)	σ	$\bar{x}$ (m)	σ	$\bar{\theta}$(deg)	σ
Fovis	**0.012**	0.007	**0.526**	0.307	**0.056**	0.035	**2.377**	1.328
DVO	0.024	0.012	0.982	0.512	0.058	0.045	2.396	1.539
FastICP	0.022	0.022	0.942	0.618	0.066	0.103	3.012	2.704
Our	0.025	0.018	1.253	0.842	0.063	0.069	3.113	3.082

coverage changed from 83.8% to 54.9% and the mean intensity changed from 169.5 to 77.8. In this dataset, there are some fast rotations. The average translational velocity is 0.334 m/s and the average angular velocity is 29.882 deg/s, which is much faster than the freiburg2/desk dataset. The mean relative pose error of each method is shown in Table III. From the experimental results, we can see that even though our odometry is not as accurate as the Fovis method, it can achieve similar accuracy to that of other state-of-the-art visual odometry methods by using only a fraction of the computational resources.

7.2. Real-time Localization Experiments

To realize localization in a given 3D map, the Lidar odometry and mapping (LOAM) (Zhang & Singh, 2014) system was used to create the offline 3D map. The LOAM system can build a very accurate point cloud map by using a rotating 2D laser scanner. In all the experiments, we set our map resolution to 4 cm. We tested the localization algorithms in different kinds of environments by carrying or semiautonomously flying our customized MAV.

7.2.1. Robustness in Challenging Environments

We tested our localization algorithm in two degraded visual environments. In both experiments, the illumination was very low. The difference is that one was a cluttered and narrow environment, while the other was more structured but almost completely dark.

The first experiment was in a narrow and cluttered environment inside of a ship, which has a size of 16 m $\times$25.6 m $\times$ 4.04 m. In this environment, most of the time the RGB images are very dark, as shown in Figure 10(a), while depth images are still very good. However, there are some places that are very challenging for depth-based odometry estimation methods. For example, when the robot entered the spacious room around position A or made a turn around position B, as shown in Figure 10(a), the depth camera could only see the ground plane and the side wall. In both scenarios, the odometry estimation became ill-conditioned because of the poor geometric structure, which made it very difficult to determine all 6DoF. It can be seen that the odometry estimations around positions A and B are not very accurate, inducing incorrect translations. Because this error will be accumulated, the whole odometry trajectory in this environment does not look very good. As mentioned in Section 7.1.1, a robust solution that can solve this problem is to incorporate RGB information. Unfortunately, our environment is a visually degraded environment where RGB information is not available. That is one reason why we need a prior map for navigation, because if we only use odometry, this failure will lead to tragic results, and it is inevitable for geometry-based methods. However, by using robust particle filtering, our localization algorithm can successfully localize the robot in those challenging areas. Although the localization accuracy is not very good around those places, our algorithm can work robustly in those areas, which is very important for control and motion planning. The localization result in this environment is shown in Figure 10(a). As one can see, our localization algorithm can successfully survive those challenging areas and quickly track the true position again.

The second experiment was in a structured but almost completely dark environment with a size of 11.8 m $\times$ 19.2 m $\times$ 2.8 m. In this environment, we cannot get any useful information from RGB images. There are also some challenging locations where the RGB-D camera can only see the ground plane, one wall or two parallel walls, or even detect nothing when it is very close to the wall, where the depth image returns nothing because the minimum measurement range of the RGB-D camera is around 0.5 m. In those places [around positions A and B, as shown in Figure 10(b)], the depth-based odometry will also suffer from the degeneration problem again. As can be seen from Figure 10(b), incorrect translations occurred in the odometry estimation. In our experiments, we found that if the odometry failure was relatively short in duration (less than 3 s), it was possible for the localization algorithm to overcome this failure. The localization result of this experiment is shown in Figure 10(b).

In both experiments, we just used the depth odometry as the motion model. Experimental results show that even without very good odometry, our localization algorithm can still work very well in different challenging environments.

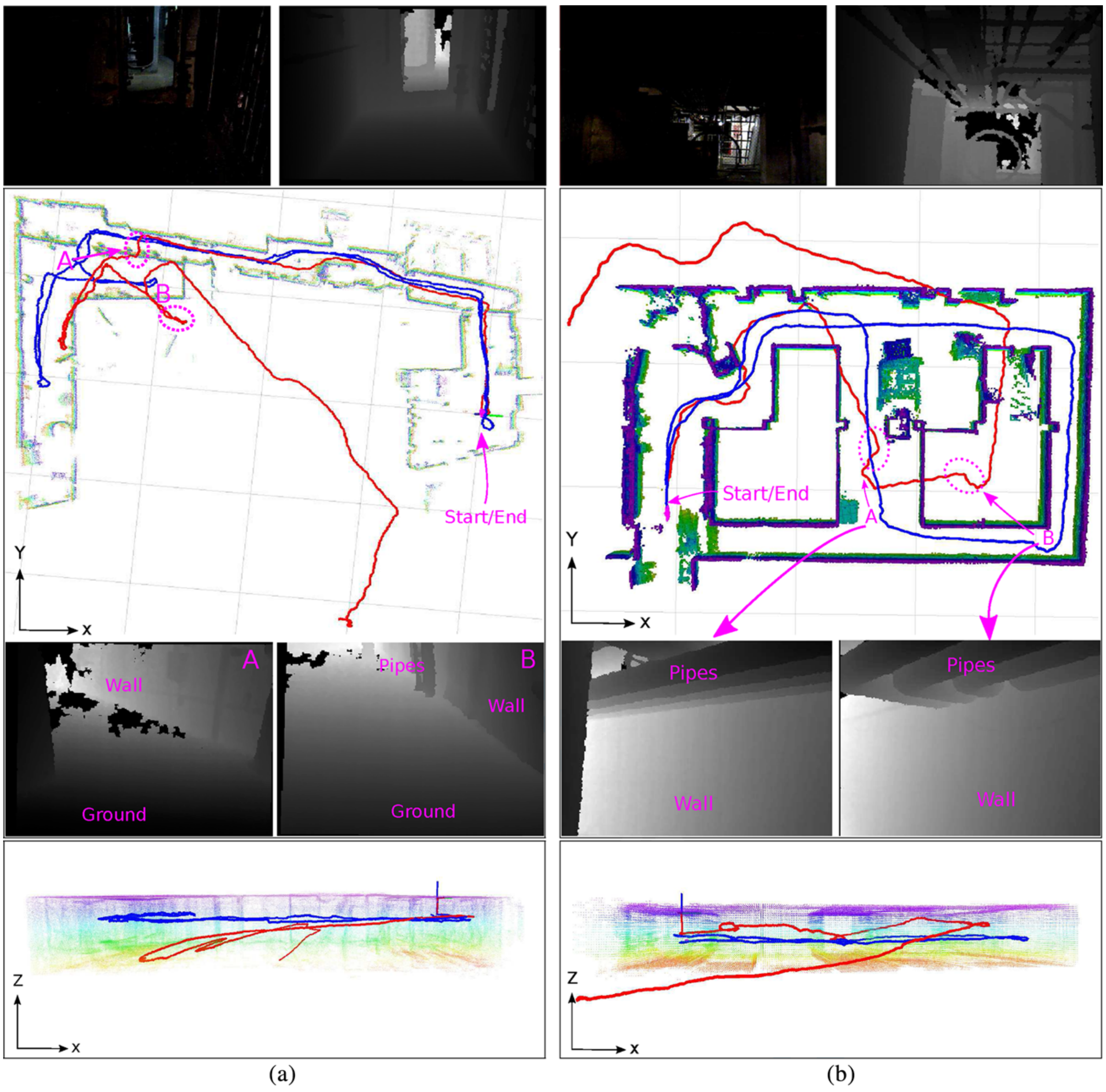


Figure 10. Localization (blue) and odometry trajectory (red) projected onto the 3D Octomap in two visually degraded environments. [We first create the 3D point cloud map using a hand-held laser mapping device described in Zhang & Singh (2014), and then we covert the point cloud map into the 3D Octomap.] The first row shows some images from the RGB-D camera. The second row shows the localization results compared to only odometry estimation (x-y). The third row shows the depth images of some challenging areas. From the depth images, you can see that only a few geometry features or constraints (pipes, etc.) are very useful for pose estimation in those challenging areas because smooth ground and wall does not provide enough constraints for determining all 6DoF. The last row figures show the side view of the localization and odometry trajectories. From the figures, one can see the drift in the Z direction of the odometry estimation (red). In contrast, one can see that the localization result (blue) is quite good. A video of those experiments can be found in https://www.youtube.com/watch?v=zYpjd55rlD8.

Table IV. Localization accuracy for datasets shown in Figure 11.

Environments	Distance	RSME	Mean	Std
Office	47.2 m	0.161 m	0.152 m	0.056 m

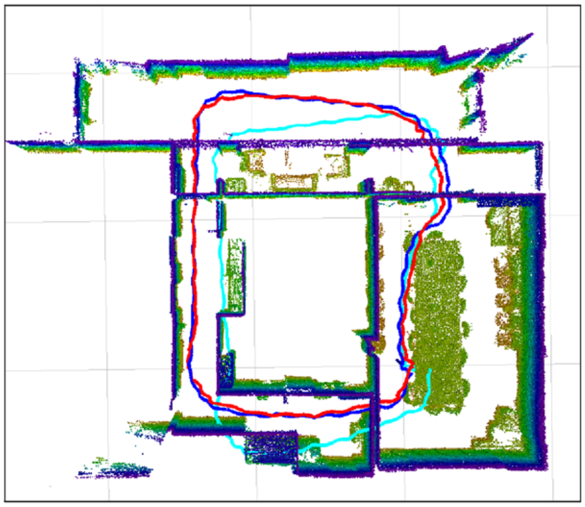


Figure 11. Accuracy comparison with ground truth in an office environment: Cyan: odometry. Red: localization. Blue: ground truth.

7.2.2. Localization Accuracy

In this subsection, we compare the localization accuracy with the ground truth from the LOAM mapping system. We attached the sensors to the LOAM system and recorded the datasets for offline comparison. Since the estimation accuracy of the LOAM system is very high, we could consider its trajectory as ground truth. We tested our localization algorithm in an office environment similar to the ship, where there were some chairs, tables, office furniture, and long corridors. For the experiment, the map resolution was 4 cm and the particle number was set to 500. The localization algorithm updates the pose when the robot moves every 10 cm or turns 0.1 rad. The experimental results are shown in Table IV and Figure 11. The accuracy of our localization algorithm is better than that of others [Fallon et al. (2012) and Biswas & Veloso (2012)]. In their work, their mean localization error is about 40 cm, while ours is about 16 cm (note that in our localization algorithm, the observation update is executed only after the robot moves every 10 cm or turns 0.1 rad). It should be noted that the localization accuracy changes in different environments or when the robot moves at different speeds because those factors affect the accuracy of odometry estimation dramatically.

Table V. Runtime performance on an embedded computer.

	Algorithm Runtime			
Name	Mean	Min	Max	StdDev
Odometry	18.3 ms	8 ms	25.8 ms	5.2 ms
Localization	65.8 ms	45.8 ms	97 ms	16.5 ms
Total CPU usage	34.5%	30.5%	44%	2.80%

7.2.3. Runtime Performance Evaluation

Runtime performance is very important for MAVs because the onboard computational abilities are limited. In our experiment, we tested the runtime performance of our system on an Odroid XU3 embedded computer. Our odometry and localization algorithms are both single-threaded programs. Therefore, each algorithm takes one core. In our experiment, the RGB-D data were recorded at a frame rate of 15 Hz with QVGA resolution. For the experiment in Figure 10(a), the runtime performance is shown in Table V. In our experiment, we used 300 particles. When it is running at 15 Hz, the CPU usage is very low, which leaves many computational resources for path planning and obstacle avoidance.

7.3. Local Motion-planning Experiments

The motion-planning algorithm is validated in a set of experiments using real-world point cloud data of a ship model. We simulated a depth camera using ray tracing, built obstacle mapping online, and then generated trajectories accordingly. The lookahead planning pose is 0.5 s from now on, about 0.25 m away. The local planner keeps replanning from the lookahead state until the mission points are reached.

7.3.1. Path-planning Experiments

In this experiment, we compared our method with a commonly used RRT* planner (Karaman & Frazzoli, 2011). Our path planning does not fix the end points for optimization, but RRT* requires a fixed goal point. Therefore, to bias RRT* by decreasing the search space, the local mission points were set 2 m from each other, which is shorter than the original 5 m. The RRT* cost function is set as the path length and obstacle cost. All the algorithms are implemented on the embedded computer, and the planning example in one scenario is shown in Figure 12. A detailed comparison of the whole run is shown in Table VI. The snap cost comes from the same polynomial trajectory generation module. RRT* requires more time than our method to generate a valid path, and it has lower quality in terms of obstacle clearance and snap cost. This is mostly due to the fact that the corridor is a structured environment where obstacles usually lie on two sides. Therefore, our path library can quickly generate

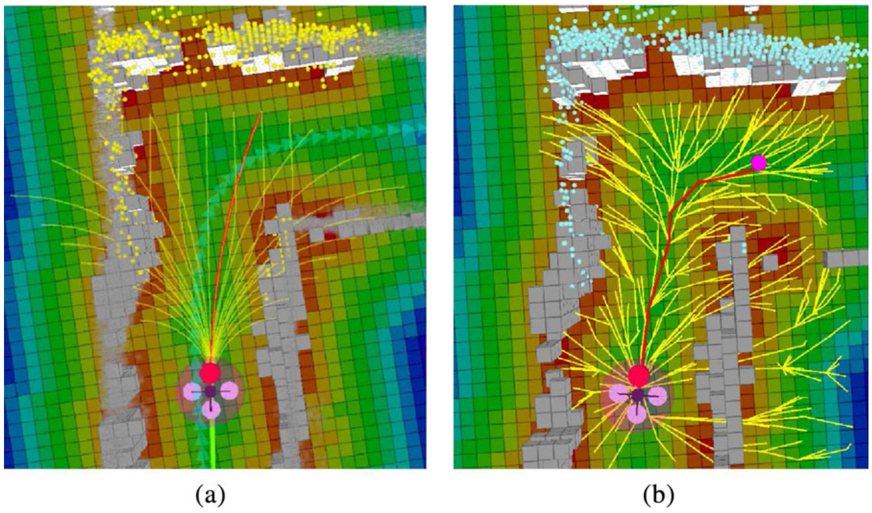


Figure 12. (a) Our path library planning example. The gray cubes represent the occupancy grid. Colorful square represents the distance map. Yellow curves are the offline path library, and the best one is shown in red. (b) RRT* planning. Yellow segments show the RRT tree. The best searched path is shown in red.

Table VI. Path-planning comparison with RRT*. "Dist" is the vehicle distance to the obstacle.

Methods	Time (ms)	Mean dist (m)	Min dist	Mean snap (m/s^4)	Max snap
RRT*	70	0.46	0.16	1.46	14.02
Our method	30	0.47	0.18	0.58	2.50

a smooth and safe path, while RRT* needs many random samples (see Figure 12).

The mean obstacle clearance is 0.47 m, nearly half the corridor width (1 m). The closest distance is 0.18 m at the door. From the point cloud, the door is about 0.44 m wide so the vehicle is nearly in the center of the door.

Table VII. Trajectory generation comparison.

Method	Time (ms)	Snap (m/s^4)
Unconstrained	2.6	27.26
Matlab quadprog.m	16.4	14.62
Our method	0.5	16.38

7.3.2. Trajectory Generation Experiments

We designed some simulation scenarios to evaluate and compare the performance of minimum snap trajectory generation. Our method solves the ill-conditioned problem of quadratic programming by Tikhonov regularization, while Bry *et al.* (Richter et al., 2013) propose an unconstrained optimization method using the substitution technique. Their method first finds the optimal waypoint derivatives, and then computes the polynomial coefficients. Due to the special formulation, it requires at least a ninth-order polynomial in each segment, while we only require fifth-order.

We solve a batch of 10 randomized polynomial optimization examples. Six waypoints are chosen randomly between $[0, 2]$, and time allocation is predefined. Our method uses sixth-order polynomials, while the unconstrained method uses ninth-order polynomials. The comparison is implemented in Matlab on a desktop CPU (Intel i7, 4.0 GHz), and the result is shown in Table VII. Our method achieves nearly the same snap cost as the Matlab QP (quadratic programming) solver. If we do not use a regularization term, it always returns a failure because the matrix is nearly singular for inversion. The unconstrained method uses more computation time, but the quality of the trajectory is lower compared to ours. The possible reason is that it uses higher order (ninth) and still includes some matrix inversion, which might be unstable. But our method is only effective for small segments' trajectory generation; for more segments with a large horizon, regularization may not solve the ill-conditioned problem.

Through the continuous lookahead replanning mechanism, the vehicle is able to reach the goal. The vehicle pose history during simulation is shown in Figure 13. From the image, the vehicle lies nearly in the middle of a corridor.

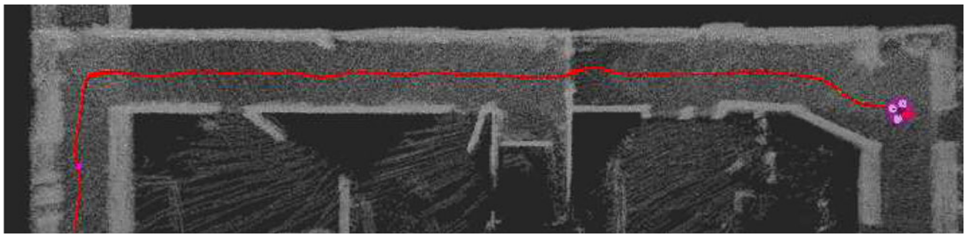


Figure 13. Vehicle history poses using a replanning mechanism in a simulated shipboard environment. The vehicle starts from bottom left to right.

Table VIII. Motion planning runtime performance on an embedded computer.

Name	Mean	Min	Max	Std
CPU usage	10.96%	7.25%	15.62%	1.68%
Planning time	29.2 ms	15.2 ms	37.8 ms	6.7 ms

7.3.3. Runtime Performance Evaluation

Runtime performance of motion planning was evaluated on the Odroid XU3 embedded computer, shown in Table VIII. We can see that obstacle mapping and local motion planning take only 10% of CPU resources. The proposed algorithm could run at 30 Hz, but we find that a 2 Hz trajectory update is sufficient for corridor flight.

7.4. Field Demonstration Experiments

7.4.1. Mission Description

For the demonstration, the mission of the MAV is to search in a partially known shipboard environment, locate the areas with fire, and create the temperature map of the whole environment. To accomplish this goal, our robot uses an onboard RGB-D camera (only depth images are used) for autonomous navigation, and a FLIR infrared camera to detect the environment temperature and mark the high-temperature areas. We first used a hand-held mapping device (Zhang & Singh, 2014) to create the global map of the whole demo area. This global map was used for both localization and global planning. The resolution was set to 4 cm. Figure 14 shows the created point cloud map. Our goal is to launch the MAV around a “start point,” then let the robot autonomously explore the 1-m-wide and 20-m-long corridor, go through the narrow doorway, and go to the “end point.” At the same time, the robot records infrared images and detects the fire and high-temperature areas and puts markers on the map for rescuers to see where the fire is.

We performed three kinds of experiments to test our system. For each kind of experiment, we carried out at least four successful experiments. We simulated the fire-fighting scenario, where some places have good illumination while some places are fully dark and others are filled with smoke and fire. Those environments pose different challenges for our robot. We wanted to test the performance of our system in those different kinds of practical environments.

7.4.2. Experimental Results of Autonomous Flights

We performed a total of 20 experiments of autonomous flights in this demo area under different environmental conditions, including four experiments under normal conditions, seven under completely dark conditions, and nine under smoky conditions.

(i) Test 1: Normal Conditions. In this test series, all lights in the hallway were on, which was a little bit better for the optical flow estimation. However, in our localization system we did not use any RGB information, therefore it did not influence the localization performance too much. Actually, having all the lights on is not good for the depth camera, because there are no depth values returned from those bright lamps. The reason is that our depth camera is based on structured lights. Figure 15 shows the experimental results, and it can be seen that our robot can realize reliable localization in this test. It should be noted that the robot trajectory is not smooth, which does not mean bad localization results. Actually, our robot was well-localized in this test. One reason why the robot was not very stable is that the hallway is very narrow, therefore the airflow really influenced the control performance of the robot. We believe that we still need to improve the robustness and accuracy of our position controller in this environment.

(ii) Test 2: Dark Conditions. For this test series we turned off all the lights in the test area, and we wanted to test the ability of autonomous navigation in a totally dark environment. The reason for doing this is that nowadays most navigation systems of MAVs are using visual information, which cannot work in a dark environment. In contrast, our method uses depth information only, which works not only in environments with abundant illumination but also in environments where there are no visual features. In this environment, the downward-looking optical flow sensor was influenced dramatically because the illumination was very poor. However, in our experiments we found that our robot was still well-localized and successfully went through the very narrow doorway several times, as shown in Figure 16.

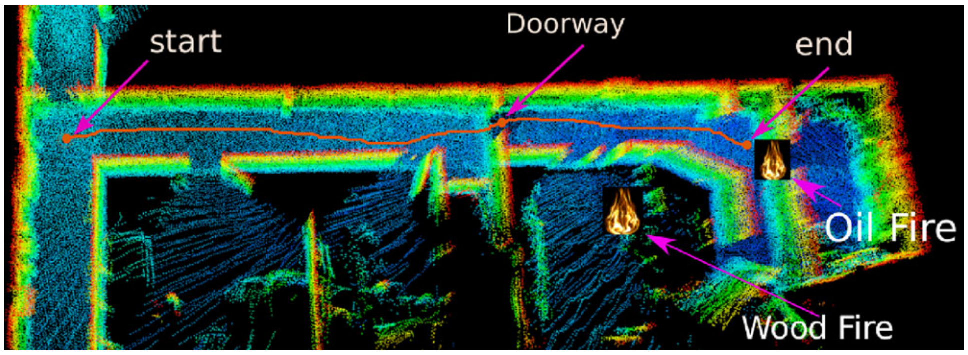


Figure 14. Global Point Cloud Map. From the start point, the vehicle flies autonomously in the corridor, goes through the narrow doorway, and reaches the end point. At the same time, the vehicle needs to detect fire and high-temperature areas using infrared cameras.

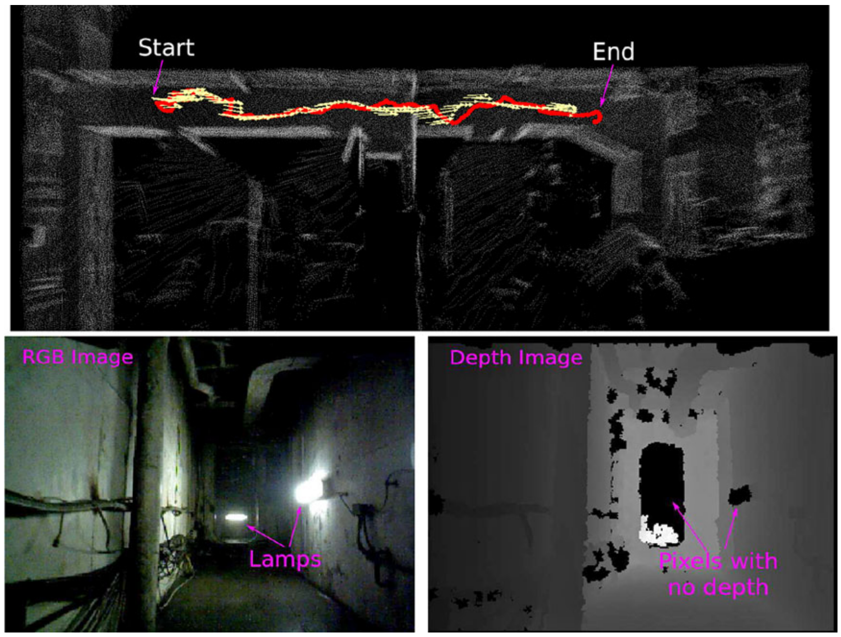


Figure 15. The first row shows odometry estimation (yellow) and localization (red) results in normal environment (with lights on) during autonomous flights. The second row shows the RGB and corresponding depth images of the dark environment.

Experimental results showed that our system could work very well in fully dark environments. We think our multi-sensor fusion framework really improves the robustness of the whole system.

(iii) Test 3: With Fire and Smoke. In this test series, we simulated the fire-fighting scene inside the shipboard environment and carried out several experiments to test the performance of our autonomous MAV. We carried out eight experiments with a wood fire and an oil fire together with light smoke, and one experiment with very dense smoke. Figure 17 shows the results of one experiment from the light smoke test series. We put some wood fire and oil fire in the side room and at the end of the passageway, as shown in Figure 17. The smoke in this test was not very dense, and we found that even though the performance of the depth camera was reduced dramatically (the quality of the depth images became quite poor, as shown in Figure 17), our system could still work. As can be seen from Figure 17, the drift of the odometry is much bigger than that in previous tests (especially the drift in the Z direction). However, the localization system still worked well. Our MAV successfully went through the narrow doorway five times in eight experiments. Then we added some oil fire, which gave us much denser smoke. In this experiment, we found that our system could not work very well under those environmental conditions (i.e., dense smoke generated by the wood fire and the

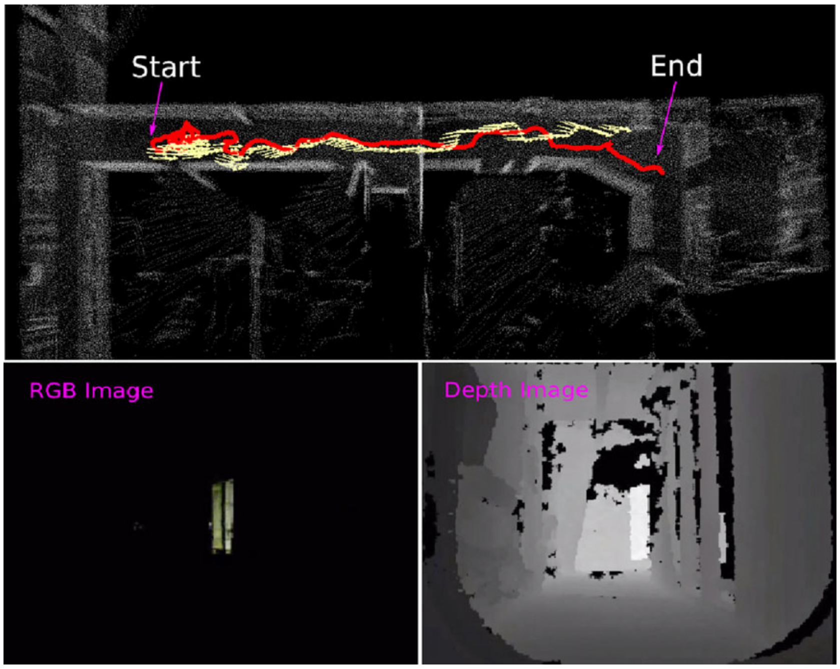


Figure 16. The first row shows odometry estimation (yellow) and localization (red) results in a dark environment during autonomous flights. The second row shows the RGB and corresponding depth images of the dark environment.

Table IX. Success rate of experiments.

Environment	Total run	Succeed	Rate
Normal	4	4	100%
Dark	7	5	71.4%
Smoky	9	5	55.5%

oil fire). When the smoke was very dense, the depth camera returned almost no useful depth information, which made the whole system fail. Our MAV could not successfully go through the narrow doorway when the smoke was very dense. Therefore, autonomous navigation in dense smoky environments is still an open problem.

We also recorded the CPU usage during the field experiments. In all the experiments, we used 300 particles for particle filtering localization. When all modules were running, the total CPU usage was between 60% and 65%. The experimental results showed that our navigation system could run in real time by only using the onboard computation resources. Some snapshots of the flight are shown in Figure 18. We performed a total of 20 experiments of autonomous flights in this demo area under different environmental conditions. The success rate of 20 runs is shown in Table IX. Failure cases are usually due to quadrotors being slightly rotated and stuck in the narrow doorway. It is difficult to cross the doorway in smoky environment because the depth image is corrupted by smoke, making it difficult to perform state estimation, obstacle detection, and trajectory tracking. Experimental results show that our robot can work very well in all conditions except dense smoke. These experiments also show that despite the difficulties associated with flying robustly in challenging shipboard environments, it is possible to use a MAV to fly autonomously into a confined ship environment to rapidly gather situational information to guide firefighting and rescue efforts.

In this field demonstration, fire/smoke detection was not used as a guiding process. It was a completely open-loop system that would output the locations of fire seen by the camera without actively searching for it. The next step is to combine fire detection with planning by trying to get to areas with increasing temperature gradient. The video of a field experiment can be found at https://www.youtube.com/watch?v=g3dWQCECwlY.

8. LIMITATIONS AND LESSONS LEARNED

In this work, we have successfully developed a micro aerial vehicle that can autonomously fly into a constrained and visually degraded shipboard environment even with moderate smoke. However, our system also has some limitations. In addition, during the development and field experiments, we experienced numerous instructive failures. In this section, we discuss the limitations and what we learned from those lessons.

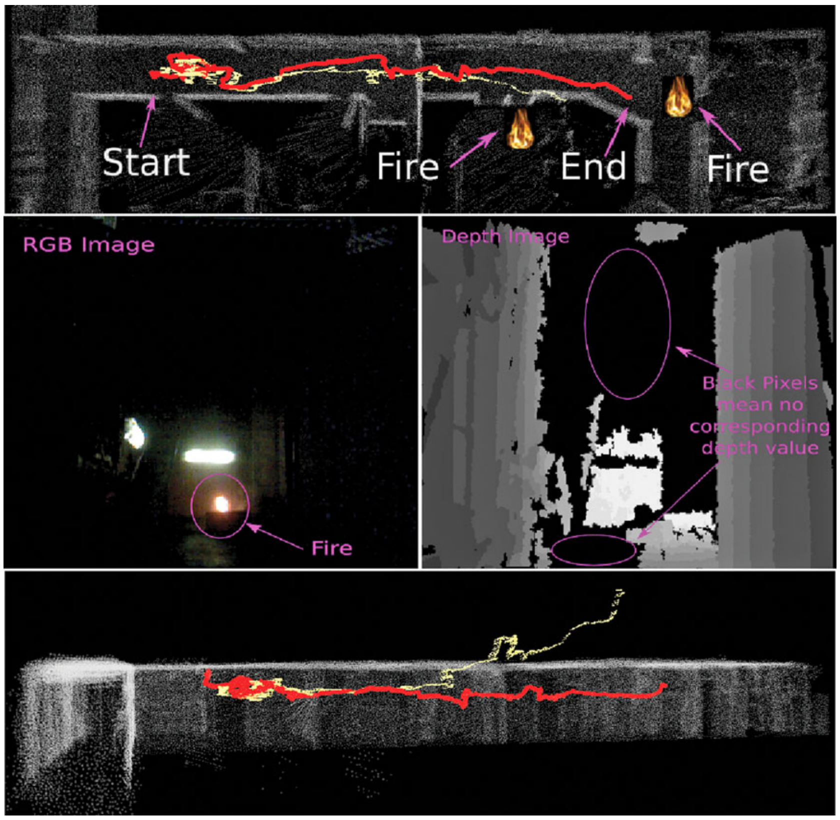


Figure 17. The first row shows odometry estimation (yellow) and localization (red) results in a light smoke environment during autonomous flights. The second row shows RGB and corresponding depth images from an onboard RGB-D camera. As can be sees, the quality of the depth image became quite poor since many pixels do not have a depth value due to the smoke. The third row shows the side view of odometry and localization. As can be seen, the odometry drifted dramatically in the Z direction.

8.1. Limitations

Currently, our system is still not an optimal design due to some existing constraints. First, our vehicle is still a little large for some narrow hatches in a real shipboard environment. However, given the current payload and flight duration requirements, we have the narrowest quadrotor design we can imagine. We are working on moving from a quadrotor to a single/coaxial rotor design to decrease size but increase flight time efficiency. Second, for the sensor systems, we have not incorporated the thermal camera information with other sensors to improve the pose estimation or control. The current FLIR thermal camera only provides image frames at 9 Hz, which is difficult for real-time pose estimation alone. In the future, we will choose a higher frame rate thermal camera to fuse its information with other sensors in order to get more robust pose estimation, especially in the case of dense smoke. Furthermore, we will investigate new fire and smoke detection algorithms and use the detected fire or smoke information to guide our MAV to search the shipboard more efficiently. Third, the controller needs to take into account the aerodynamic disturbance force in the narrow space due to the ground or wall effect, which could pose a big difficulty for accurate trajectory tracking. An adaptive controller or model predictive controller could be utilized. Lastly, our visual odometry is fused with an IMU via a loosely coupled strategy. This is easy to implement, but the information is not fully used. In the future, we want to try a tightly coupled visual-inertial SLAM idea to make the estimation more accurate and robust.

8.2. Calibration Problems

In experiments, we found that our depth camera always underestimated the actual distance. If we did not do the depth calibration, then the close points from the point cloud were correct but the far points were severely underestimated. That means that far objects in the point cloud could not be aligned to the global map correctly, which often caused the whole localization system to fail.

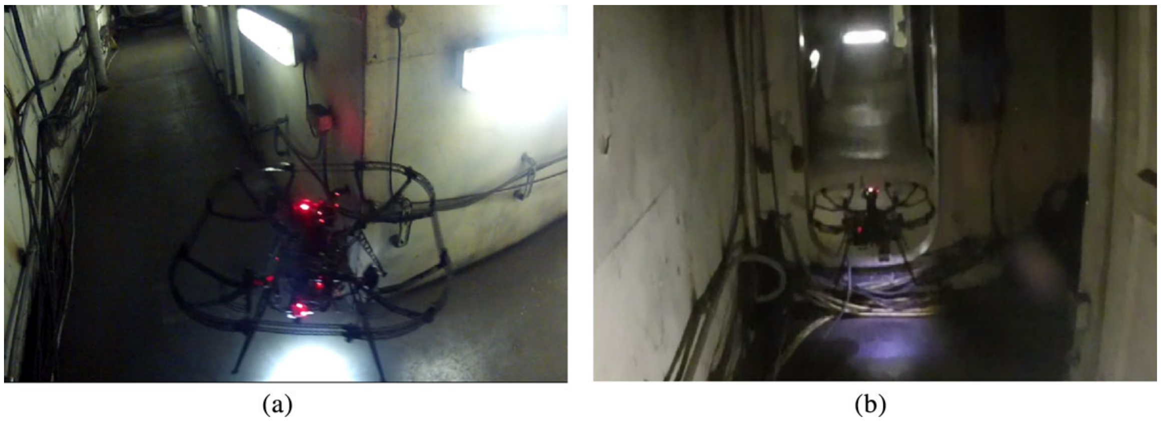


Figure 18. Experiment demonstration. Left: turning. Right: crossing doorway.

8.3. Delay Problems

Our visual odometry outputs the transform at 15 Hz, and this estimation could be fed into FCU's KF filter to fuse with other data for pose control. We found that if we just use visual odometry, it is not fast enough for KF filtering. The latency sometimes made the filtering not smooth. Therefore, we mounted a Micro-Strain IMU, then used UKF to fuse the odometry and IMU information, and fed the fused result to the FCU. By doing so, our system became much more stable than before.

8.4. Hardware Problems

We encountered several hardware problems that caused our system to fail. For example, the wifi communication between the ground station computer (for remote monitoring and visualization) and the MAV was sometimes quite poor. To alleviate this problem, we used the Ethernet port instead of the onboard USB wifi and changed our communication protocol from TCP to UDP. Another communication problem is between Pixhawk and Odroid, i.e., we found that the USB communication was also not very stable. In addition, the Sonar sensor readings provided by the PX4flow were also not very stable. Therefore, in order to improve the robustness of the system, we mounted a point laser for robust height estimation.

8.5. Computational Resources

All of the software components of our system currently run on a single ARM-based embedded computer on the vehicle. Although most of our algorithms can run at a higher rate, we observed that if we increased the sampling rate of depth images to 30 Hz, the whole system would consume almost 100% of the CPU resources. Therefore, we set the sampling rate at 15 Hz, which can still give us good estimates but with the CPU usage of the whole system around only 65%. However, this limited the moving speed of our MAV, which only moves at a maximum speed of 1 m/s. If it moves too fast, our navigation system will fail.

8.6. Dense Smoke Problems

Our MAV failed to go through the narrow doorway in dense smoke environments because the smoke corrupted the depth image dramatically. Therefore, autonomous navigation in challenging visually degraded environment (such as dense smoke environments) is still an open problem. It is hard to get useful RGB or depth images in dark and dense smoke environments even when we put some lights on the MAV. However, we found that the thermal camera could usually get very good images in dark and even heavy smoke environments. We think that the thermal images are very useful for autonomous navigation in such challenging environments. In the future, we will use more thermal information together with other sensor information to improve the robustness of our pose estimation, fire, and smoke detection algorithms, and we will also use that information for autonomous searching and detection of people, fire, and smoke in challenging environments.

9. CONCLUSION AND FUTURE WORK

In this paper, we have shown the feasibility of an autonomous fire detection MAV system in a GPS-denied environment with tough visibility conditions. This was achieved without the need for any additional infrastructure on the ship, reducing the installation and maintenance costs of the system. We achieved an autonomous flight with fully online and onboard control, planning, and state estimation in complete darkness through 1-m-wide passages while crossing doorways with only 8 cm clearance. We demonstrated 10 consecutive runs where the vehicle crossed a lit, completely dark, and smoky passageway, respectively, and ended by detecting wood and diesel fires.

Future challenges will be to increase to robustness and safety of the vehicle while increasing flight time. This will involve improvements in both software and hardware. The current vehicle size is a bit large, resulting in a very tight fit through ship doorways. In the future, we intend to move from a quadrotor design to a single/coaxial ducted rotor design to decrease size but increase flight time efficiency. Currently, our sensor suite loses reliability in dense smoke conditions, leaving the robot inoperable. We plan on adding sensors that extend the range of environments that our robot can successfully navigate and inspect. On the software side, one important goal is to decrease the dependency on a prior map for state estimation to make the system more adaptable to changing or damaged environments. Pursuing exploration and mapping in a damaged environment poses many interesting research challenges.

ACKNOWLEDGMENTS

The work presented in this paper is supported by the Office of Naval Research GS100000006 (http://dx.doi.org/10.13039/100000006) (GS100000006N00014-13-C-0259), China Scholarship Council, and the Natural Science Foundation of China GS501100001809 (http://dx.doi.org/10.13039/501100001809) (No. 61573091). The authors would also like to thank J. Zhang for his help in creating the offline map.

REFERENCES

Achtelik, M., Bachrach, A., He, R., Prentice, S., & Roy, N. (2009). Stereo vision and laser odometry for autonomous helicopters in gps-denied indoor environments. In Proceedings of SPIE, Unmanned Systems Technology (vol. 7332, pp. 733219–733219–10).

Angeletti, G., & Valente, J. (2008). Autonomous indoor hovering with a quadrotor. In International Conference on Simulation, Modeling, and Programming for Autonamous Robots (pp. 472–481). Venice, Italy.

Besl, P., & McKay, H. (1992). A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(2), 239–256.

Biswas, J., & Veloso, M. (2012). Depth camera based indoor mobile robot localization and navigation. In 2012 IEEE International Conference on Robotics and Automation (pp. 1697–1702). IEEE.

Boeuf, A., Cortés, J., Alami, R., & Siméon, T. (2014). Planning agile motions for quadrotors in constrained environments. In 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014) (pp. 218–223). IEEE.

Bonnin-Pascual, F., Garcia-Fidalgo, E., & Ortiz, A. (2012). Semi-autonomous visual inspection of vessels assisted by an unmanned micro aerial vehicle. IEEE International Conference on Intelligent Robots and Systems (pp. 3955–3961).

Bry, A., Bachrach, A., & Roy, N. (2012). State estimation for aggressive flight in GPS-denied environments using onboard sensing. In 2012 IEEE International Conference on Robotics and Automation (pp. 1–8). IEEE.

Bry, A., Richter, C., Bachrach, A., & Roy, N. (2015). Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. The International Journal of Robotics Research, 34(7), 969–1002.

Chambers, A., Scherer, S., Yoder, L., Jain, S., Nuske, S., & Singh, S. (2014). Robust multi-sensor fusion for micro aerial vehicle navigation in GPS-degraded/denied environments. In 2014 American Control Conference (pp. 1892–1899). IEEE.

Cupec, R., Nyarko, E. K., Filko, D., Kitanov, A., & Petrović, I. (2013). Global localization based on 3D planar surface segments detected by a 3D camera. In Proceedings of the Croatian Computer Vision Workshop, Year 1 (pp. 31–36). Zagreb, Croatia.

Dey, D., Shankar, K. S., Zeng, S., Mehta, R., Agcayazi, M. T., Eriksen, C., Daftry, S., Hebert, M., & Bagnell, J. A. D. (2015). Vision and learning for deliberative monocular cluttered flight. In Field and Service Robotics (FSR), IV (pp. 391–409).

Droeschel, D., Nieuwenhuisen, M., Beul, M., Holz, D., Stueckler, J., & Behnke, S. (2015). Multilayered mapping and navigation for autonomous micro aerial vehicles. Journal of Field Robotics, 7(Pt. 1), 81–86.

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

Eich, M., Bonnin-Pascaul, F., Gracia-Fidalgo, E., & Ortiz, A. (2014). A robot application for marine vessel inspection. Journal of Field Robotics, 31(2), 319–341.

Fallon, M. F., Johannsson, H., & Leonard, J. J. (2012). Efficient scene simulation for robust monte carlo localization using an RGB-D camera. In Proceedings of the IEEE International Conference on Robotics and Automation (pp. 1663–1670).

Fang, Z., & Scherer, S. (2014). Experimental study of odometry estimation methods using RGB-D cameras. In 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 680–687). IEEE.

Fang, Z., & Scherer, S. (2015). Real-time onboard 6DoF localization of an indoor MAV in degraded visual environments using a RGB-D camera. In 2015 IEEE International Conference on Robotics and Automation (ICRA) (pp. 5253–5259). IEEE.

Fang, Z., Yang, S., Jain, S., Dubey, G., Maeta, S. M., Roth, S., Scherer, S., Zhang, Y., & Nuske, S. T. (2015). Robust autonomous flight in constrained and visually degraded environments. In Field and Service Robotics, IV (pp. 411–425).

Ferguson, D., Howard, T. M., & Likhachev, M. (2008). Motion planning in urban environments. Journal of Field Robotics, 25(11-12), 939–960.

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6), 381–395.

Flores, G., Zhou, S., Lozano, R., & Castillo, P. (2014). A vision and GPS-based real-time trajectory planning for a MAV in unknown and low-sunlight environments. Journal of Intelligent and Robotic Systems: Theory and Applications, 74(1-2), 59–67.

Forster, C., Pizzoli, M., & Scaramuzza, D. (2014). SVO: Fast semi-direct monocular visual odometry. In 2014 IEEE International Conference on Robotics and Automation (ICRA) (pp. 15–22). IEEE.

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

Garcia-Fidalgo, E., Ortiz, A., Bonnin-Pascual, F., & Company, J. P. (2015). A mosaicing approach for vessel visual inspection using a micro-aerial vehicle. IEEE International Conference on Intelligent Robots and Systems.

Golub, G. H., Hansen, P. C., & O'Leary, D. P. (1999). Tikhonov regularization and total least squares. SIAM Journal on Matrix Analysis and Applications, 21(1), 185–194.

Green, C., & Kelly, A. (2007). Toward optimal sampling in the space of paths. In 13th International Symposium of Robotics Research, November 26–29, Hiroshima, Japan.

Grzonka, S., Grisetti, G., & Burgard, W. (2009). Towards a navigation system for autonomous indoor flying. In 2009 IEEE International Conference on Robotics and Automation (pp. 2878–2883). IEEE.

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

Horn, B. K. P., & Harris, J. G. (1991). Rigid body motion from range image sequences. CVGIP Image Understanding 53(1), 1–13.

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

Huang, A. S., & Bachrach, A. (2011). Visual odometry and mapping for autonomous flight using an RGB-D camera. International Symposium on Robotics Research (pp. 1–16).

Kalra, N., Ferguson, D., & Stentz, A. (2009). Incremental reconstruction of generalized Voronoi diagrams on grids. Robotics and Autonomous Systems, 57(2), 123–128.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research, 30(7), 846–894.

Kerl, C., Sturm, J., & Cremers, D. (2013). Robust odometry estimation for RGB-D cameras. In 2013 IEEE International Conference on Robotics and Automation (pp. 3748–3754). IEEE.

Kr, P., Colas, F., Furgale, P., & Siegwart, R. (2014). Long-term 3D map maintenance in dynamic environments. Proceedings of the 2014 IEEE International Conference on Robotics & Automation (ICRA) (vol. 1, pp. 3712–3719).

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2008). Anytime search in dynamic graphs. Artificial Intelligence, 172(14), 1613–1643.

Maier, D., Hornung, A., & Bennewitz, M. (2012). Real-time navigation in 3D environments based on depth camera data. 2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012) (pp. 692–697).

Mayne, D. (1966). A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. International Journal of Control, 3(1), 85–95.

Mellinger, D., & Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. In 2011 IEEE International Conference on Robotics and Automation (ICRA) (pp. 2520–2525). IEEE.

Moré, J. J. (1978). The Levenberg-Marquardt algorithm: Implementation and theory. In Numerical Analysis (pp. 105–116). Springer.

Murray, R. M. (2009). Optimization-based control. California Institute of Technology.

Murray, R. M., Li, Z., & Sastry, S. S. (1994). A Mathematical Introduction to Robotic Manipulation (vol. 29). Boca Raton: CRC Press.

Murray, R. M., Rathinam, M., & Sluis, W. (1995). Differential flatness of mechanical control systems: A catalog of prototype systems. In ASME International Mechanical Engineering Congress and Exposition. Citeseer.

Nieuwenhuisen, M., Droeschel, D., Beul, M., & Behnke, S. (2015). Autonomous navigation for micro aerial vehicles in complex GNSS-denied environments. Journal of Intelligent & Robotic Systems, 1, 1–18.

Nüchter, A., Lingemann, K., Hertzberg, J., & Surmann, H. (2007). 6D SLAM'3D mapping outdoor environments. Journal of Field Robotics, 24(8-9), 699–722.

Oishi, S., Jeong, Y., Kurazume, R., Iwashita, Y., & Hasegawa, T. (2013). ND voxel localization using large-scale 3D environmental map and RGB-D camera. In 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO) (pp. 538–545). IEEE.

Ortiz, A., Bonnin-Pascual, F., & Garcia-Fidalgo, E. (2013). Vessel inspection: A micro-aerial vehicle-based approach. Journal of Intelligent and Robotic Systems: Theory and Applications, 1–17.

Pomerleau, F., Colas, F., Siegwart, R., & Magnenat, S. (2013). Comparing ICP variants on real-world data sets. Autonomous Robots, 34(3), 133–148.

Ratliff, N., Zucker, M., Bagnell, J. A., & Srinivasa, S. (2009). CHOMP: Gradient optimization techniques for efficient motion planning. In IEEE International Conference on Robotics and Automation, 2009. ICRA'09 (pp. 489–494). IEEE.

Richter, C., Bry, A., & Roy, N. (2013). Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In Proceedings of the International Symposium of Robotics Research (ISRR) (pp. 1–16).

Rusinkiewicz, S., & Levoy, M. (2001). Efficient variants of the ICP algorithm. In Proceedings of the Third International Conference on 3-D Digital Imaging Modelling (pp. 145–152). IEEE.

Scaramuzza, D., Achtelik, M. C., Doitsidis, L., Friedrich, F., Kosmatopoulos, E., Martinelli, A., Achtelik, M. W., Chli, M., Chatzichristofis, S., Kneip, L., Gurdan, D., Heng, L., Lee, G. H., Lynen, S., Pollefeys, M., Renzaglia, A., Siegwart, R., Stumpf, J. C., Tanskanen, P., Troiani, C., Weiss, S., & Meier, L. (2014). Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in GPS-denied environments. IEEE Robotics & Automation Magazine, 21(3), 26–40.

Schauwecker, K., & Zell, A. (2014). On-board dual-stereo-vision for the navigation of an autonomous MAV. Journal of Intelligent and Robotic Systems: Theory and Applications, 74, 1–16.

Scherer, S., Rehder, J., Achar, S., Cover, H., Chambers, A., Nuske, S., & Singh, S. (2012). River mapping from a flying robot: State estimation, river detection, and obstacle mapping. Autonomous Robots, 33(1-2), 189–214.

Scherer, S., Singh, S., Chamberlain, L., & Elgersma, M. (2008). Flying fast and low among obstacles: Methodology and experiments. The International Journal of Robotics Research, 27(5), 549–574.

Shen, S., Michael, N., & Kumar, V. (2011). Autonomous multi-floor indoor navigation with a computationally constrained MAV. In Proceedings of the IEEE International Conference on Robotics and Automation (pp. 20–25). IEEE.

Shen, S., Michael, N., & Kumar, V. (2013). Obtaining liftoff indoors: Autonomous navigation in confined indoor environments. IEEE Robotics & Automation Magazine, 20(4), 40–48.

Spies, H., Jähne, B., & Barron, J. L. (2002). Range flow estimation. Computer Vision and Image Understanding, 85, 209–231.

Stoyanov, T., Magnusson, M., Andreasson, H., & Lilienthal, A. J. (2012). Fast and accurate scan registration through minimization of the distance between compact 3D NDT representations. The International Journal of Robotics Research, 31(12), 1377–1393.

Sturm, J., Engelhard, N., Endres, F., Burgard, W., & Cremers, D. (2012). A benchmark for the evaluation of RGB-D SLAM systems. IEEE International Conference on Intelligent Robots and Systems (pp. 573–580).

Thrun, S., Burgard, W., & Fox, D. (2005). Probabilistic robotics (intelligent robotics and autonomous agents). Cambridge, MA: MIT Press.

Thrun, S., Fox, D., & Burgard, W. (2000). Proceedings of the National Conference on Monte Carlo Localization with Mixture Proposal Distribution (pp. 859–865). AAAI Press.

Valenti, R. G., Dryanovski, I., Jaramillo, C., & Str, D. P. (2014). Autonomous quadrotor flight using onboard RGB-D visual odometry. 2014 IEEE International Conference on Robotics and Automation (pp. 5233–5238).

Weiss, S., Achtelik, M. W., Chli, M., & Siegwart, R. (2012). Versatile distributed pose estimation and sensor self-calibration for an autonomous MAV. 2012 IEEE International Conference on Robotics and Automation (pp. 31–38).

Weiss, S., Scaramuzza, D., & Siegwart, R. (2011). Monocular-SLAM-based navigation for autonomous micro helicopters in GPS-denied environments. Journal of Field Robotics, 28(6), 854–874.

Wu, A. D., Johnson, E. N., Kaess, M., Dellaert, F., & Chowdhary, G. (2013). Autonomous flight in GPS-denied environments using monocular vision and inertial sensors. Journal of Aerospace Information Systems, 10, 172–186.

Yamamoto, M., Boulanger, P., Beraldin, J.-A., & Rioux, M. (1993). Direct estimation of range flow on deformable shape from a video rate range camera. IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(1), 82–89.

Zhang, J., & Singh, S. (2014). LOAM: Lidar odometry and mapping in real-time. In Proceedings of Robotics: Science and Systems, Berkeley, CA.