High-precision Autonomous Flight in Constrained Shipboard Environments

Shichao Yang, Zheng Fang, Sezal Jain,
Geetesh Dubey, Silvio Maeta, Stephan Roth,
Sebastian Scherer, Yu Zhang and Stephen Nuske

CMU-RI-TR-15-06

Feb 2015

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

This paper addresses the problem of autonomous navigation of a micro aerial vehicle (MAV) inside of a constrained shipboard environment to aid in fire control, which might be perilous or inaccessible for humans. The environment is GPS-denied and visually degraded, containing narrow passageways, doorways and small objects protruding from the wall, which makes existing 2D LIDAR, vision or mechanical bumper-based autonomous navigation solutions fail. To realize autonomous navigation in such challenging environments, we first propose a fast and robust state estimation algorithm that fuses estimates from a direct depth odometry method and a Monte Carlo localization algorithm with other sensor information in a two-level fusion framework. Then, an online motion planning algorithm that combines trajectory optimization with receding horizon control is proposed for fast obstacle avoidance. All the computations are done in real-time onboard our customized MAV platform. We validate the system by running experiments in different environmental conditions. The results of over 10 runs show that our vehicle robustly navigates 20m long corridors only 1m wide and goes through a very narrow doorway (only 4cm clearance on each side) completely autonomously even when it is completely dark or full of light smoke.

Contents

1 Introduction

Over the past few years Micro Air Vehicles(MAV) have gained a wide popularity in both military and civil domains. Surveillance and reconnaissance is one area where they have made a huge impact because of greater mobility and cost effectiveness as compared to ground vehicles. In this paper, we aim to develop a MAV that is capable of autonomously navigating through a shipboard to aid in fire control, damage assessment and inspection, which might be perilous and or inaccessible for humans. Such a constrained environment poses various challenges for navigating though narrow corridors, tight doorways and also because they can be visually degraded: potentially dark, smoke-filled and GPS-denied. An illustrative picture is shown in Fig. 1. To successfully fulfil this mission, the vehicle must localize itself accurately and avoid obstacles safely. Besides, all the computation of state estimation, motion planning and control needs to be done onboard in real-time.

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

For successful operation in such environments, we need to address these challenging problems. *First*, the MAV should be small enough to travel in the narrow corridors with tight doorways (65cm width, 8cm clearance). Therefore, it can only use lightweight sensors with limited range and computers with limited computational resources. *Second*, visually degraded environments and some areas free of clutter(long corridors) provide scanty visual and geometry features and cues respectively, posing great difficulty for accurate state estimation. In some areas,

there are many obstacles on the wall, making it difficult for fast obstacle avoidance. In addition, air turbulence in confined spaces poses difficulty for precise control.

To address the above challenges, we build a robust and efficient autonomous navigation system with the following contributions. We first propose a fast depth odometry method which directly recovers the ego-motion from a series of depth images. Then, a Monte Carlo localization algorithm is used to estimate the absolute pose of the MAV in a given 3D map. After that, the pose estimates from odometry and localization are fused with other sensor information in a two-level fusion framework to get fast and robust state estimation for real-time control. Finally, a fast online motion planning algorithm that combines trajectory optimization with receding horizon control is proposed for obstacle avoidance. We demonstrate the effectiveness of our system through both simulation and field experiments. The experiment is performed in a constrained shipboard containing a 20m long, 1m wide corridor and a 8cm clearance doorway. We conducted more than 10 runs under various environment conditions, from normal to complete dark and also in smoke-filled environments to demonstrate autonomous navigation capabilities of the MAV.

The rest of paper is organized as follows. In Section 2, we present the related work. System setup and diagram is shown in Section 3. Section 5 and 6 describe the novel approach for a robust and reliable state estimation and realtime smooth motion planning correspondingly. Controller is described in Section 7. Experiments demonstrating each module is presented in Section 8 and conclusion is made 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 to solve the autonomous navigation problem, the amount of processing that is performed onboard/offboard and the assumptions made about the environment (outdoor/indoor, structured/unstructured).

2D Laser scanner has been extensively and successfully used for autonomous navigation for its accuracy and speed, for example, in [1, 2, 3]. However, those systems are usually only suitable for structured or 2.5D environments. Recently, there are also many vision-based navigation systems since cameras can provide rich information and have low weight, low power and low price. For example,

stereo camera is used in [4] and [5], monocular camera with IMU is used in [6, 7, 8], but vision is sensitive to illumination changes and could not work in darkness. Recently, RGB-D cameras have become very popular for autonomous navigation of indoor MAVs [9, 10, 11] because they can provide both image and depth. For example in [10], a RGB-D visual odometry method is proposed for real-time pose estimation of a MAV and a 3D map is create offline. In [11], a fast visual odometry method is used for pose estimation and 3D visual SLAM is used for constructing a 3D octomap in real-time.

Unfortunately, the existing autonomous navigation methods can not work in our case since 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 [10, 8] could not work in our case. For obstacle avoidance and motion planning, many above papers compute paths offline [2] [11], or heavily rely on prior map [1]. Some papers online generates steering angle by vector field histogram [5] or waypoints by sampling planner (RRT*) [3]. But these commands are not suitable for precise control in constrained narrow environment. In our proposed system, we mainly use depth images for odometry estimation, localization. Online motion planning generates optimal dynamic-feasible smooth trajectories suitable for precise control. Therefore, they are not sensitive to illumination and can work in completely dark even light smoke-filled environments.

3 System Overview

The platform we use for our experiment is a customized MAV as shown in Fig. 2. It's mainly composed of two computation units. One unit is an ARM based Quadcore embedded computer (Odroid XU3), responsible for high-level task processing, such as odometry estimation, localization and motion planning, etc. The other one is the Pixhawk flight controller unit (FCU) which is used for multi-sensor data fusion and real-time control. A fordward-looking RGB-D camera is used for pose estimation and motion planning. A downward-looking optical flow camera is used for velocity estimation and a point laser is used for height estimation. Besides, a FLIR camera is used for fire detection.

To get the robust, low-delay state estimates, a two-level estimation architecture is designed as shown in Fig. 2. The odometry (15Hz) is first fused with MicroStrain IMU in UKF. Then, the outputs of UKF (50Hz) and localization (15Hz) are fused with optical flow, raw data from sensors (FCU IMU and point laser) in an

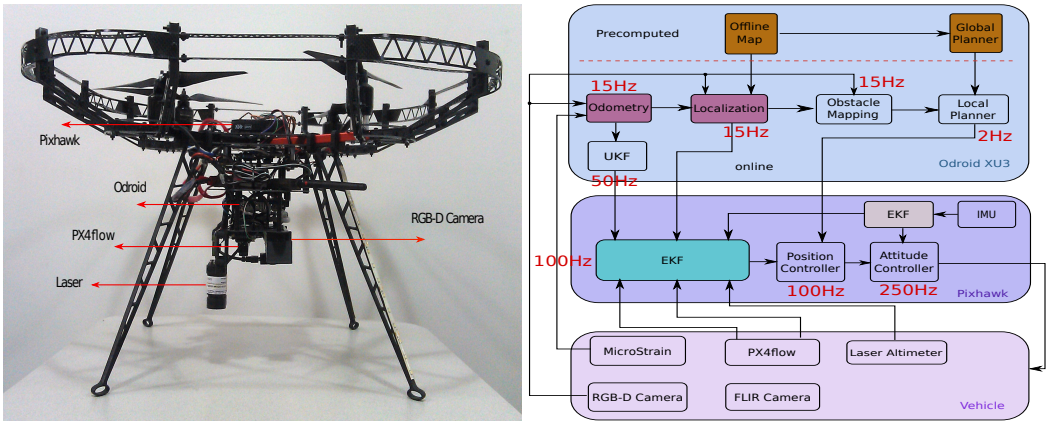


Figure 2: System Overview: Left image shows the MAV platform, right image shows the software architecture

EKF framework on the FCU to provide a high frequency (100Hz) state estimate for real-time control. This a bit redundant estimation strategy improves the robustness and safety of our system in challenging environments. RGB-D camera is also used for 3D occupancy mapping (15Hz) [12] and motion planning to generate collision-free trajectories (2Hz). Trajectories are fed into a hierarchical control approach for accurate tracking [13] based on differential flatness. It is composed of outer position control running at 100Hz and inner attitude control running at 250Hz.

4 Real-time Pose Estimation

Stable and precise control of an autonomous MAV demands fast and accurate estimates of the vehicle's pose and velocity. Especially, MAVs operating in cluttered indoor environments need pose updating at high rates with little latency for position control. Real-time and accurate state estimation is most important during tasks that require high accuracy, such as indoor flight, obstacle avoidance and autonomous landing. In this project, we want to solve two problems, **relative pose and absolute pose estimation**, by only using onboard sensors and computation resources. For relative pose estimation, namely odometry estimation, we want to develop a robust and real-time odometry estimation method that can adapt to different kinds of indoor environments, such as structured environments, cluttered environments and illumination changing and smoky environments. For absolute

pose estimation, we want to estimate the absolute 6DOF state of the robot in a given 3D map and track it in real-time during the mission.

4.1 Low-frequency Pose Estimation

Current RGB-D odometry estimation methods mainly use RGB information for motion estimation. Therefore, they cannot work in featureless or degraded visual environments. We propose to do the opposite. We try to estimate the pose using as little RGB information as possible. Our method uses depth images to calculate the frame-to-frame motion. In this paper, we use a direct method to compute the frame to frame motion from depth images directly, which is robust and much faster than traditional ICP based methods. However, if only depth image are used, the odometry will suffer from a degeneration problem in some challenging environments. Our method only tries to use RGB information when the degeneration is very severe. When severe degeneration happens, we try to use dense visual odometry method to calculate the frame-to-frame motion. By doing so, our odometry method can work very robustly in degraded visual environments.

4.1.1 Robust Direct RGB-D Odometry Estimation

1) Direct Motion Estimation from Depth Images Most existing depth-based motion estimation methods are based on registration algorithms, such as ICP [14], 3D NDT [15] or 3D geometric feature based methods [16]. Those methods can get very accurate relative pose estimation if a dense point cloud is available. However, they are too slow and computationally heavy to run on a MAV. In this paper, a direct method [17] is used to calculate the frame-to-frame motion estimation. It directly works on the depth image without detecting any features or doing any additional transform.

Let a 3D point $R = (X, Y, Z)^T$(measured in the depth camera's coordinate system) is captured at pixel position $r = (x, y)^T$in the depth imageZ_t. This point undergoes a 3D motion $\Delta R = (\Delta X, \Delta Y, \Delta Z)^T$, which results in an image motion Δr 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 $r + \Delta r$ will have consequently changed by this amount:

$$Z_1(r + \Delta r) = Z_0(r) + \Delta Z \tag{1}$$

This equation is called **range change constraint equation**. Taking the first-order Taylor expansion of the term $Z_1(r + \Delta r)$ generates a pixel-based constraint relat-

ing the gradient of the depth image ∇Z_1 and the temporal depth difference to the unknown pixel motion and the change of depth as follows:

$$Z_1(r + \Delta r) = Z_1(r) + \nabla Z_1(r) * \Delta r = Z_0(r) + \Delta Z \tag{2}$$

As it is know to all, any small 2D displacement Δr can be related directly to the 3D displacement ΔR which gave rise to it by differentiating the perspective projection equation with respect to the components of the 3D position R

$$\frac{\partial r}{\partial R} = \frac{\Delta r}{\Delta R} = K(r) = \begin{bmatrix} \frac{f_x}{Z} & 0 & -X\frac{f_x}{Z^2} \\ 0 & \frac{f_y}{Z} & -Y\frac{f_y}{Z^2} \end{bmatrix} \tag{3}$$

Substituting Eq. 3 into Eq. 2, we can get a linear constraint equation of three unknowns relating the motion ΔR of a 3D point imaged at pixel r to the gradient of the depth image $\nabla Z_1 = (Z_x, Z_y)$ and the temporal depth difference:

$$(Z_x, Z_y, -1) \begin{pmatrix} \Delta r \\ \Delta Z \end{pmatrix} = Z_0(r) - Z_1(r) \tag{4}$$

Let $R = (X, Y, Z)^T$ be a vector to a point on the surface (measured in a senor-centered Cartesian coordinate system). If the sensor moves with instantaneous transnational velocity v and instantaneous rotational velocity ω with respect to the environment, then the point R appears to move with a velocity

$$\frac{dR}{dt} = -v - \omega \times R \tag{5}$$

with respect to the sensor. Substituting Eq. 5 into Eq. 4, we can get:

$$\begin{bmatrix} -Y - Z_y f_y - Z_x XY\frac{f_x}{Z^2} - Z_y Y^2\frac{f_y}{Z^2} \\ X + Z_x f_x + Z_x X^2\frac{f_x}{Z^2} + Z_y XY\frac{f_y}{Z^2} \\ -Z_x Y\frac{f_x}{Z} + Z_y X\frac{f_y}{Z} \\ Z_x\frac{f_x}{Z} \\ Z_y\frac{f_y}{Z} \\ -1 - Z_x X\frac{f_x}{Z^2} - Z_y Y\frac{f_y}{Z^2} \end{bmatrix}^T \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_x \\ v_y \\ v_z \end{bmatrix} = Z_0(r) - Z_1(r) \tag{6}$$

2) Dealing with Degeneration Problem The direct depth-based method can estimate the frame-to-frame transform very fast. However, in environments with few geometric features, this method will suffer from the degeneration problem, for example when the camera can only see a ground plane or parallel walls. In these "ill-conditioned" cases which are really common in indoor environments, the direct depth-based method will produce wrong estimates. In such cases, the only way to solve the problem is to try to use additional information, such as RGB information or IMU information.

In our algorithm, we try to detect when the degeneration happens. And, when the degeneration happens, we try to use visual information to calculate the frame-to-frame transform. Since we want to get a very fast odometry estimation with low CPU usage, we do not always incorporate the visual information into the depth image based odometry estimation method. Though joint optimization methods [18],[19] may get more accurate estimation, it is much more expensive to compute. In our system, we are not too concern with the accuracy of odometry estimation method since our localization algorithm can correct the drift of visual odometry. Compared to accuracy, robustness is more important for localization since a sudden odometry failure or wrong estimation will influence the localization performance much more than an inaccurate odometry estimation.

As you can see from Eq. 6, the frame-to-frame transform is calculated by using a least square regression method. Here, we try to analyze the eigenvalues of Eq. 6 to determine whether the equation is "ill-conditioned". A rule of thumb is that the greater the number of eigenvalues near zero, the greater the number of linear dependencies among the variables. In mathematics, we can use a measure called *condition number*, which is the ratio of the largest to the smallest eigenvalue, to detect the degeneration. A high condition number indicates the presence of collinearity. A low condition number indicates near perfect collinearity. The guidelines for assessing condition number are shown in Table 1.

We use the above condition number to measurement the degeneration degree of depth image based odometry estimation method. When severe degeneration happens, we incorporate RGB information to estimate the frame to frame motion.

3) Dense Visual Odometry from Color Images The dense visual odometry method [20] is used to estimate the frame-to-frame motion when degeneration happens in the depth based method described in Section III.A. In contrast to sparse feature based methods, dense visual odometry methods want to fully exploit both the intensity and the depth information provided by RGB-D sensors. Dense visual odometry uses all color information of the two images and the depth information of the first image. In our previous study [21], we found that dense visual odom-

Table 1: Degree of Collinearity Base on Condition Number

Condition Number $(\lambda_{max}/\lambda_{min})$	Degree of Collinearity
If (CN $<$ 100)	Weak
If (100 $<$ CN $<$ 1000)	Moderate to Strong
If (CN $>$ 1000)	Severe

etry is more robust than sparse feature based visual odometry methods in some challenging environments.

The brightness change constraint equation relates how the luminance at a 3D scene point is captured in temporally separated intensity images. A 3D point R is imaged at pixel position $r = (x, y)^T$ in the intensity map I_t. This point undergoes a 3D motion ΔR which results in an image motion Δr between frames t_0 and t_1 and reprojects with the same intensity at the new image location $r + \Delta r$, thus the **brightness change constraint equation** can be described as:

$$I_1(r + \Delta r) \quad = \quad I_0(r)$$

Taking the first-order Taylor expansion of term $I_1(r + \Delta r)$, we can get following equation:

$$\nabla I_1(r)\Delta r \quad = \quad I_0(r) - I_1(r) \tag{7}$$

Substituting Eq. 3 and Eq. 5 into Eq. (7), we can get:

$$\nabla I_1(r) \begin{bmatrix} \frac{f_x}{Z} & 0 & -X\frac{f_x}{Z^2} \\ 0 & \frac{f_y}{Z} & -Y\frac{f_y}{Z^2} \end{bmatrix} \begin{bmatrix} 0 & Z & -Y & 1 & 0 & 0 \\ -Z & 0 & X & 0 & 1 & 0 \\ Y & -X & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_x \\ v_y \\ v_z \end{bmatrix} \quad = \quad I_0(r) - I_1(r)$$

where $\nabla I_1(r) = [E_x, E_y]$

Finally, we can write the intensity constraint equation as:

$$\begin{bmatrix} -E_x f_y - E_x XY\frac{f_x}{Z^2} - E_y Y^2\frac{f_y}{Z^2} \\ E_x f_x + E_x X^2\frac{f_x}{Z^2} + E_y XY\frac{f_y}{Z^2} \\ -E_x Y\frac{f_x}{Z} + E_y X\frac{f_y}{Z} \\ E_x\frac{f_x}{Z} \\ E_y\frac{f_y}{Z} \\ -E_x X\frac{f_x}{Z^2} - E_y Y\frac{f_y}{Z^2} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_x \\ v_y \\ v_z \end{bmatrix} = I_0(r) - I_1(r) \qquad (8)$$

In practice, the estimation based on Eq. 7 can not recover an accurate estimate of the camera motion in a single iteration. The reason is that sometimes there is a fast motion which breaks the small motion assumptions, then the nonlinearity is sever. Therefore, we should iteratively estimate the motion. To deal with fast motion problem, this implementation uses a coarse to fine optimization approach, trying to find the warping function that maximizes the photoconsistency between two consecutive RGBD frames at different image scales. That is, the optimizer starts computing a first pose approximation at a low resolution image scale, and then uses the estimated solution to initialize the optimization at a greater resolution image scale to refine the solution.

4.1.2 Particle Filtering Based Localization

From Section 4.1.1, we can get a robust odometry estimation, however it will definitely drift after a long time of running. In order to get an accurate absolute pose in the environment, we need a localization algorithm to locate the robot in a given 3D environment.

For 6DoF absolute localization in a given 3D Map, there are two important things that we should consider. First, what kind of 3D map representation should we use? Second, what kind of localization algorithm should we use? There are several different kinds of 3D map representation approaches, such as point cloud, planes, NDT map [22], 3D octomap [23] and 3D Polygonal Map [24]. Some of them are raw data based maps, while some of them are feature-based maps. Here, we select 3D octomap as our map representation since it is compact and

can represent many kinds of environments. For localization, a particle filter algorithm (also known as Monte Carlo Localization, MCL) is selected since it is very robust, which has already verified extensively on ground mobile robots [25]. Thought the MCL has been successfully used on ground mobile robots, 6DoF pose $(x, y, z, \phi, \theta, \psi)$ which needs to be estimated for MAVs increases the complexity of the problem. In this paper, we show that by carefully designing the motion and observation model, MCL can work very well on an embedded computer.

1) Particle Filtering Algorithm Particle Filtering Localization is a Bayes filtering technique which recursively estimates the posterior about the robot's pose X_t at time t:

$$p(X_t|Z_{1:t}, u_{1:t-1}) = \eta \cdot p(Z_t|X_t) \cdot \int_{x_{t-1}} p(X_t|X_{t-1}, u_t) \cdot p(X_{t-1}|Z_{1:t-1}, u_{1:t-1}) dX_{t-1}$$

Here, η is a normalization constant resulting from Bayes' rule, $u_{1:t}$ is the sequence of all motion commands up to time t, and $Z_{1:t}$ is the sequence of all observations. $p(X_t|X_{t-1}, u_t)$ is called motion model, which means the probability that the robot ends up in state X_t given it executes the motion command u_t in state X_{t-1}. And, $p(Z_t|X_t)$ is the observation model, which means the likelihood of obtaining observation Z_t given the robot's current pose is X_t. The particle filter approximates the belief of the robot with a set of particles:

$$X_t = \left\{ \left(x_t^1, w_t^1\right), ..., \left(x_t^n, w_t^n\right) \right\} \tag{9}$$

where, each x_t^i is one pose hypothesis and w_t^i is the corresponding weight. The particle set is updated iteratively by sampling those particles from the motion model and compute a weight according to the observation model. Particles are then resampled according to this weight and the process iterates.

2) Motion Model For each subsequent frame, we propagate the previous state estimate according to the motion model $p(X_t|X_{t-1}, u_t)$ using the odometry computed by the fast direct RGB-D odometry proposed in Section III. For each dimension, the propagation equations are of the form:

$$x_k = x_{k-1} + \Delta x + e_{x,k} \qquad e_{x,k} \sim N(0, \sigma_x^2) \tag{10}$$

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

usually the above noise model works well. However, during abrupt accelerations or sharp turning close to the wall (the minimum and maximum measurement range are around 50cm and 700cm respectively) or in ill-conditioned cases, the odometry algorithm may suffer from periods of total failure. In such cases, we will propagate the particle set using a noise-driven dynamical model replacing Eq. 10 with

$$x_k = x_{k-1} + e_{\dot{x},k} \qquad e_{\dot{x},k} \sim N(0, \sigma_{\dot{x}}^2) \tag{11}$$

where $\sigma_{\dot{x}}$ is much bigger than σ_x.

3) Observation Model In this section, we describe how to evaluate the roll, pitch, height and x, y, yaw in details. Since our MAV works in indoor environments, most of the time it can see the ground plane. Here, the ground plane is used in two ways. First, the ground plane is detected to get the roll, pitch and height measurement to evaluate each particle's weight. Second, since the ground plane has no contribution for determining the x,y, yaw, the ground plane is filtered out when updating the particle's position weight using range measurement. The final observation model is:

$$\begin{aligned} p(Z_t|X_t) &= p(d_t, \tilde{z}_t, \tilde{\phi}_t, \tilde{\theta}_t|X_t) = \\ & p(d_t|X_t) \cdot p(\tilde{z}_t|X_t) \cdot p(\tilde{\phi}_t|X_t) \cdot p(\tilde{\theta}_t|X_t) \end{aligned} \tag{12}$$

The likelihood formulation is given by:

$$\rho(d, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{d^2}{2\sigma^2}) \tag{13}$$

- Roll,Pitch and Height Likelihood Measurement

In order to detect the ground plane from the point cloud, a RANSAC based method [26] is used. We assume that the ground plane is the furthest plane to the MAV and the closest to horizontal. After detecting the ground plane, roll, pitch and height value can be easily computed from the ground plane equation. Then, the weight of each particle is updated according to the observed measurement and predicted measurement by using following equations.

$$\begin{aligned} p(\tilde{z}_t|X_t) &= \rho(z_t - \tilde{z}_t, \sigma_z) \\ p(\tilde{\phi}_t|X_t) &= \rho(\phi_t - \tilde{\phi}_t, \sigma_\phi) \\ p(\tilde{\theta}_t|X_t) &= \rho(\theta_t - \tilde{\theta}_t, \sigma_\theta) \end{aligned} \tag{14}$$

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.

- Depth Sensing Likelihood Measurement

In order to evaluate the depth sensing likelihood $p(d_t|X_t)$, we use a sparse subset of beams from the point cloud. From our experiment, we found that how one selects the subset of beams really influences the robustness and accuracy of the localization algorithm. In order to efficiently use the points with most constraints, we try two ways to select points. First, the point cloud is segmented into ground and non-ground point clouds. Since the ground part has little importance to determine the x, y and yaw of the MAV, only very few points from the ground part is selected. For the non-ground part, we found that most time in indoor environments especially in long corridors, there are only few points on the wall are useful for determining the forward translation. If we use a uniform downsampling, then we will miss this valuable information. In order to use this information, we select those points using a Normal Space Sampling method [27]. By doing so, we can select those points with most constraints.

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 endpoint to the closest obstacle in the map:

$$p(d_{t,k}|X_t) = \rho(d, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{d^2}{2\sigma^2}) \tag{15}$$

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:

$$p(d_t|X_t) = \prod_{k=1}^{K} p(d_{t,k}|X_t). \tag{16}$$

Another issue that should be considered is that depth values of the RGB-D camera are very noisy when the measurement range is bigger than 4 meters. For example, when the measurement distance is less than 3 meters, usually the measurement error is less than 2.5cm. However, when the measurement distance is at 5 meters, the measurement error could be around 7cm. Therefore, the sensor noise is quite different at different distances. In order to include this characteristic into our observation model, we use a changing σ which increases with the measurement distance.

4.2 High-frequency Pose Estimation

For onboard planning and control correct and robust estimate of the position and orientation of the platform is required. The correct estimate of attitude is imperative to keep the platform stable and fly safely. This estimate needs to be updated at a very high rate so as to generate stabilization commands frequently. On the other hand position estimate is required to allow platform to be self aware of its placement in the surroundings and hence allow it to plan appropriate paths to maneuver around obstacles and in corridors.

This can be achieved using various sensors but they provide heterogeneous information and they need to be fused appropriately to derive the required results. The state estimation task we currently worked on involves fusion of data coming from various sensors and asynchronously running state estimation algorithms.

We fuse data from all of these sources on the FCU itself to output high frequency state estimate state Estimation. We run a very high rate attitude estimator which stabilizes the platform's angular motion. The heading is generally corrected using a magnetometer but since such a sensor would fail inside a ship which is a metal body we use gyroscopes to estimate heading rate and correct the absolute heading using the onboard localizer. The position estimates are computed using data from optical flow, visual odometry and inertial sensors. Since the odometry computed from VO is not smooth we further fuse it with a secodary IMU conected to Odroid and filter it using a UKF which outputs a smoother, reliable and more frequent odometry. Fusing data from various sensors allows us to keep track of our position even when one sensor fails but other has some qualitative estimate. This estimate is used to update the position of the platform in a coordinate system agnostic to the localizer's coordinate system. The localizer uses this ballpark idea of current motion to spread the particles for absolute pose estimate. Once the absolute pose is estimated the localizer maintains a track of motion estimate generated from the FCU and its own output. feeds it to the estimator running on FCU. The FCU subsequently converges with the absolute pose estimate.

We have developed a much robust odometry algorithm on FCU. The algorithm leverages realtime performance of a loosely coupled Kalman Filter. By loosely coupled we mean that the attitude estimation is generated by the EKF running onboard which is then used by the Kalman filter to estimate the odometry. The new filter gets data from the various sensors in sensor centric frames and then transforms them to NED frame. Now that all the information is in one single frame and since there are no angular corrections required anymore the filter becomes a linear problem easily solved using a Kalman Filter [28]

A linear system is of type:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{17}$$

Here x_t and x_{t-1} are state vectors, and u_t is the control vector at time t. Both of these vectors are vertical vectors but we do not use u_t. x_t is of the following form

$$x_t = (x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}\ \ddot{x}\ \ddot{y}\ \ddot{z}) \tag{18}$$

A_t and B_t are matrices but since we do not use u_t, B_t is also omitted. A_t is a square matrix of size $n \times n$, where n is the dimension of the state vector x_t, i.e. 9 because we use a 9DOF state. Generally B_t is of size $n \times m$, with m being the dimension of the control vector u_t. By multiplying the state and control vector with the matrices A_t and B_t, respectively, the state transition function becomes linear in its arguments. Thus, Kalman filters assume linear system dynamics. The random variable ϵ_t in Eq. 17 is a Gaussian random vector that models the randomness in the state transition. It is of the same dimension as the state vector. Its mean is zero and its covariance will be denoted R_t.

Algorithm Kalman Filter($x_{t-1}, \Sigma_{t-1}, x_t, z_t$)

1. $\overline{x}_t = A_t x_{t-1} + B_t u_t$
2. $\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
3. $K_t = \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}$
4. $x_t = \overline{x}_t + K_t(z_t - C_t \overline{u}_t)$
5. $\Sigma_t = (I - K_t C_t)\overline{\Sigma}_t$
6. return x_t, Σ_t

Kalman filters represent the belief at time t by the mean x_t and the covariance Σ_t. K is called the *Kalman Gain*.

Kalman Filter(KF) takes input from the following:

1. **EKF:** Provides the current attitude or orientation of the vehicle. EKF estimates the roll, pitch and yaw/heading using the IMU on FCU. Heading is frequently corrected by Global Pose estimator so as to align the vehicle heading with absolute heading.

2. **Laser Altimeter:** Provides the altitude above ground level(AGL) information.

3. **PX4Flow:** Provides velocities in lateral direction in vehicle frame. These velocities are projected in Global frame or NED using the attitude estimate from EKF. This helps in linearising the measurements used in the KF. This sensor also provides Sonar data which also helps in determining the AGL.

4. **UKF:** It provides velocities estimated by the UKF running on computer. These are also projected in NED frame as explained above.

5. **Global Pose Estimator:** Provides absolute position and heading of the platform.

Given all the above inputs we construct a measurement vectorz_t as:

$$z_t = \begin{pmatrix} x_{globalpose} \\ y_{globalpose} \\ z_{opticalflow} \\ z_{laser} \\ \dot{x}_{visualodomtery} \\ \dot{y}_{visualodomtery} \\ \dot{z}_{visualodomtery} \\ \dot{x}_{opticalflow} \\ \dot{y}_{opticalflow} \\ \ddot{x}_{imu} \\ \ddot{y}_{imu} \\ \ddot{z}_{imu} \end{pmatrix} \tag{19}$$

All the measurements in Eq. 19 are in NED or the defined global frame.

We included a laser ranger in addition to the sonar to estimate reliable altitude above ground. The motivation was to counter the risk of noisy output from Sonar in congested corridors. Also we wanted to eliminate usage of barometer. Barometer is highly sensitive to pressure changes and in indoor scenarios that can result in poor altitude hold performance. In a ship with varying temperature profiles due to fire this is definitely expected. We sought to use other sensors for information regarding height.

5 Motion Planning

Online motion planning is needed to keep MAV safe by quickly avoiding the obstacles, which are represented by an online 3D occupancy grid [12]. Local goal

points for planning are specified by a human or a high level mission planner. Here, we focus on local motion planning to generate collision-free trajectories, which is divided into two steps: *path planning* to generate optimal waypoints and *spline fitting* to generate optimal polynomial trajectories through waypoints.

5.1 Path Planning

We first search an optimal path, containing a series of safe waypoints to avoid the obstacles. Each waypoint contains 4 DOF $\{x, y, z, \psi(yaw)\}$, namely the flat output space of quadrotor [13]. Let the path be $\xi : [0, 1] \mapsto R^4$ mapping from time to 4 DOF such that:

$$\begin{aligned} \min_{\xi} \quad & J = w_1 f_{obst}(\xi) + w_2 f_{smooth}(\xi) + f_{goal}(\xi) \\ \text{s.t.} \quad & \xi(0) = \xi_0 \end{aligned} \tag{20}$$

where w_1, w_2 are the weighting parameters.

$f_{obst}(\xi)$ is the obstacle cost function as defined in [29]:

$$f_{obst}(\xi) = \int_0^1 c_{obs}(\xi(t)) \| \frac{d}{dt} \xi(t) \| dt \tag{21}$$

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

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

$$f_{smooth}(\xi) = \frac{1}{2} \int_0^1 \| \frac{d}{dt} \xi(t) \|^2 dt \tag{22}$$

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

$$f_{goal}(\xi) = \|\xi(1) - \xi_g\|^2 \tag{23}$$

Receding horizon control (RHC) is a common way to solve the problem in Eq. 20 due to easy implementation. It directly chooses the best path among an offline library [30]. But RHC is only optimal within the library, and it usually needs large amounts of paths so as to get a high quality path which is time consuming to check. So we combine RHC with path optimization CHOMP [29]. RHC serves to provide a good initial guess and CHOMP further optimizes it. The path library S contains 27 specifically designed paths shown in Fig. 3.

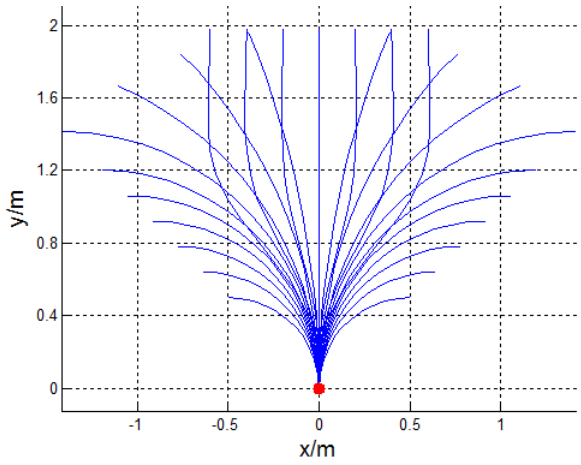


Figure 3: Initial path library. All the paths start at (0,0). The library is manually designed for the corridor environment where obstacles usually lie on two sides. It includes straight line, turning arcs with different curvatures and lane changing curves with parallel ending direction, corresponding to the three main flight modes in the corridor.

We first align the offline paths with current pose then the best path $\xi^* = \arg\min_{\xi \in S} J(\xi)$ is selected as the initial guess and optimized through CHOMP. An optimization example during turning is shown in Fig. 4, where the gradient pushes the path away from obstacles. Note that the end point is freed for optimization, different from standard CHOMP algorithm because our method is only planning within a horizon. A short horizon makes the optimization faster and more reactive.

The average cost per iteration of J in Eq. 20 during turning is shown in Fig. 5.

5.2 Spline Fitting

After getting path waypoints $\xi_0, ...\xi_n$, we need to fit a continuous spline with time profiler through them. The polynomial spline allows us to analytically compute feedforward control input for quadrotor [13], which guarantees exponential tracking stability of the controller while waypoint following or steering angle methods cannot.

Since snap, 4^{th} order derivative (wrt. time) of spline, is proportional to control input of quadrotor, it needs to be continuous and minimized. We parameterize the

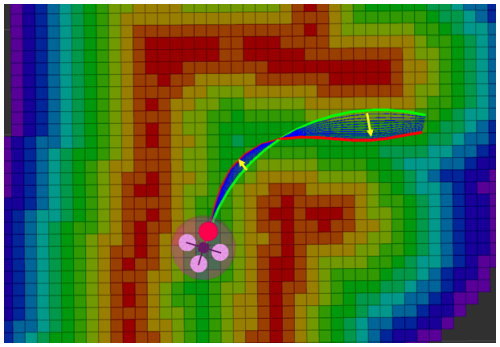


Figure 4: Path optimization in turning. The color grid represents the distance map, computed from online 3D occupancy map [12]. The green curve represents the initial best path, blue curves are the paths during optimization based on the gradient (yellow). The final optimized path is in red.

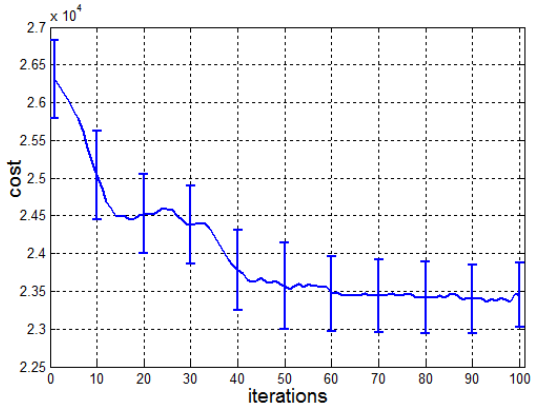


Figure 5: Totoal cost changes with each optimization iteration.

spline $\xi(t)$ as 5 segments of 6^{th} order polynomials to ensure continuity of up to 4^{th} derivative through the whole trajectory. The k^{th} segment $1 \leq k \leq 5$ of the trajectory is expressed as:

$$\xi(t) = \sum_{i=0}^{6} p_{ik} t^i, \quad t_{k-1} \leq t < t_k \tag{24}$$

5.2.1 Minimize snap

The integration of snap to be minimized is defined as:

$$f_{snap}(\xi) = \int_{t_0}^{t_n} ||\frac{d^4\xi(t)}{dt^4}||^2 \tag{25}$$

There are two kinds of contraints of optimization. (1) The *equality* constraint of optimization are imposed on the endpoints of each segment, including passing through each waypoint and keeping derivative continuity. (2) The *inequality* constraint includes within the maximum velocity and acceleration limits. In order to get an analytical solution, we first just consider *equality* constraints and assume time allocation of each segment is set based on a constant velocity model.

Let $p \in R_{35\times1}$ be the polynomial coefficients vector of one flat output, then quadratic programming problem is formulated:

$$\min_{p} \ \ p^T H p \qquad \text{s.t. } Ap = b \tag{26}$$

Where H is the integration of snap square so it is a positive semidefinite matrix. A closed form solution is found using Lagrange multipliers:

$$p = H^{-1}A^T(AH^{-1}A^T)^{-1}b \tag{27}$$

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

$$H \leftarrow H + \varepsilon I \tag{28}$$

where I is identity matrix, ε is a small positive number, here it is set to be 0.001. This Tikhonov regularization [31] provides an approximation to matrix inversion.

To deal with the *inequality* constraint, instead of time-consuming optimization through iterations, we check the *inequality* constraint after getting the polynomials from Equ. 27 and if it doesn't satisfy, we enlarge the time duration for the whole trajectory then solve Equ. 27 again. In practice, since our vehicle flies at low speed, we usually don't come to this step. A better method is to optimize the time allocation as follows.

5.2.2 Optimize time

Until this point, the time allocation for each segment is specified based on constant velocity model but actually we can optimize them to get smaller snap and better

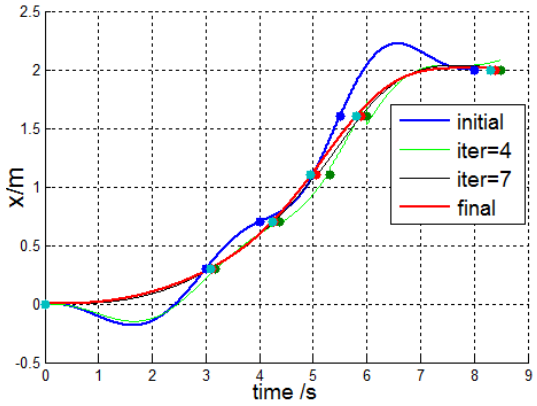


Figure 6: Minimum snap trajectory optimization. All the trajectories pass through the same waypoints but with different time setpoints. Blue curve is the initial trajectory of raw time set points. Red curve is the optimized trajectory.

satisfy *inequality* constraint. So we jointly optimize $w_3 f_{snap} + f_{time}$. But there is no analytical solution to it. Levenberg-Marquardt method [32] is adopted to optimize the time allocation $t = [\Delta t_1, \Delta t_2, ..., \Delta t_5]$ through iterations based on the combined cost function:

$$f(t) = w_3 f_{snap} + f_{time} = w_3 p(t)^T H(t) p(t) + \sum_{i=1}^{5} \Delta t_i \tag{29}$$

Where $p(t)$ is computed using Eq. 27.

We first compute Jacobian J numerically, then augment cost as $F(t) = f(t)^2$, update equation of each iteration step is:

$$\begin{gathered}(J^T J + \lambda(diag(J^T J)))\delta = J^T(-f(t)) \\ t \leftarrow t + \delta\end{gathered} \tag{30}$$

An optimization example of a minimum snap trajectory is shown in Fig. 6. The optimized trajectory in red is much smoother than initial trajectory in blue. Total cost $f(t)$ iteration is shown in Fig. 7.

We still need to check the inequality constraint after getting the optimized polynomials and time allocation from Eq. 29 and if it doesn't satisfy, we just need to increase the weight w_4 for snap cost, which tends to enlarge the total time duration, then optimize it again.

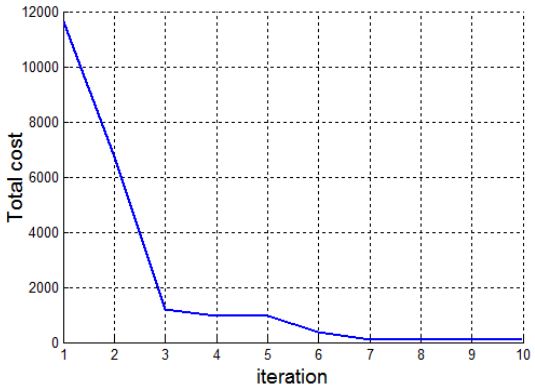


Figure 7: Cost iterations of snap cost plus time cost in during time optimization.

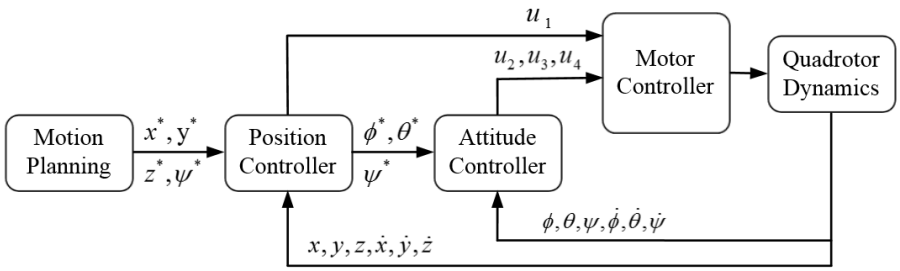


Figure 8: Control diagram of quadrotor

6 Control

A hierarchical control approach in Fig .8 is implemented to track the trajectory output from motion planning. It is implemented based on the differential flatness property of quadrotor [13], which is proved to be exponential stable. It is composed of outer position control running at 100Hz and inner attitude control running at 250Hz. Position controller includes feedforward term analytically computed from the flat outputs splines $x, y, z, \psi(yaw)$ mentioned in Section 5.2 and PID feedback term. It also computes the desired attitude $\phi(roll), \theta(pitch), \psi(yaw)$, which will be stabilized by attitude controller using PID feedback.

6.1 Trajectory Controller

To safely maneuver the platform in flight we should follow smooth trajectories. A controller has been developed which can take smooth polynomial splines as

input and follow them using thrust, velocity and position control. The splines are generated onboard and the FCU does the control part. These trajectories are sent referenced in time and the FCU does appropriate concatenation of the fragments to follow one trajectory after the other continuously. FCU does the task of asking for a plan and replan as it follows the splines.

A spline provides a polynomial each for x, y, z and $heading$. The polynomial is a function of time so for each time step the FCU can sample the new setpoint for position, velocity and thrust. Equations (31,32,33) show how the setpoints are obtained from a spline on x axis and similarly on others.

$$x_t = f_x(t) \tag{31}$$

$$\dot{x}_t = \frac{df_x(t)}{dt} \tag{32}$$

$$\ddot{x}_t = \frac{d^2 f_x(t)}{dt^2} \tag{33}$$

The obtained $\ddot{x}$ is used to compute feedforward thrusts which is directly applied to the motors and a feedback controller monitors the current position and velocities as estimated by the state estimation filter running on FCU to achieve the desired setpoints as shown in figure. We use **P** only controller for position control and **PID** controller for velocity tracking.

6.2 Autonomous Takeoff and Landing

We use the velocity controller to do fully autonomous take offs and landings. We can initiate auto take off and do position hold once reached the desired altitude. Similarly the multirotor can be commanded to do auto landing at the commanded position. This is achieved by setting the desired velocity setpoint along vertical axis to a fixed velocity and the velocity controller makes the vehicle to climb in altitude till the desired altitude is reached. Also while the vehicle is climbing the controller can also maintain the lateral position to allow takeoff and landing around a fixed spot.

7 Smoke and Fire Detection

A major aspect of this project was to detect and locate fire or smoke and alert the emergency response team based on this. This involved detecting fire using

Figure 9: A colorized image output from FLIR thermal camera (Wood Fire).

onboard sensors and then determining its location using the vehicle's estimated pose inside the ship.

7.1 Fire Detection

We use a lightweight FLIR-tau thermal camera to measure the temperature of the environment. This provides us with a 720x480 pixel image having pixel intensity corresponding to the temperature at the pixel location as shown in fig(9). We can segment the appropriate range of temperature for fire, people etc. based on this image. Anything over $100°$C is considered to have a high probability of being fire or being situated very close to the fire (eg. fig 10). Similarly segmented blobs with temperature close to $30°$C is considered to belong to a human being. As we are using a thermal camera, its possible to detect a fire even in complete dark/smoky environments making the emergency alert system independent of ambient light conditions.

7.2 Smoke Detection

Smoke detection was based on texture analysis of the color images received from the RGB-D camera onboard.The current approach is to detect the color homo-

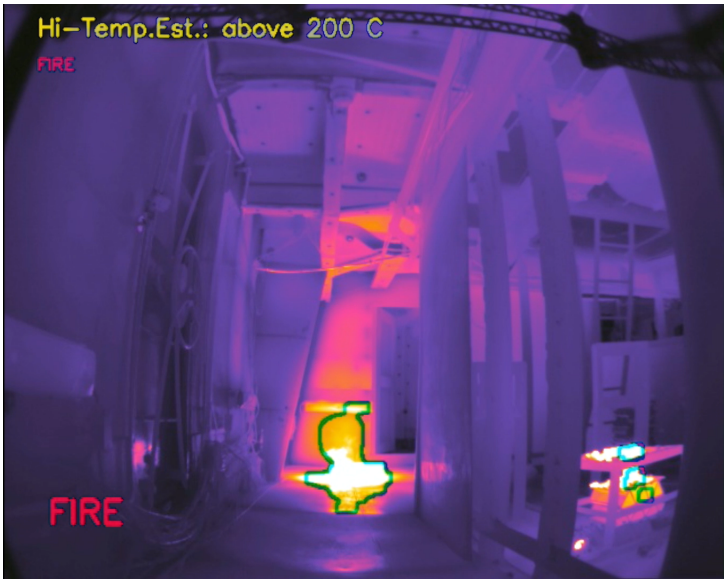


Figure 10: Fire segmentation based on temperature (Oil and Wood Fire)

geneity in the images. So the two deciding factors for smoke detection are the contrast and 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 counter argument. Lack of illumination makes smoke detection imprecise due to lowering of contrast in the image. Taking both of the above factors into account gives an empirical idea of the smoke present in the environment.

During the field demonstration at Shadwell in Nov 2014, the fire/smoke detection was not used as a guiding process. It was a completely open loop system which 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.

8 Real-world Experiments

In this section, we demonstrate each module's performance from state estimation, motion planning to control.

8.1 Real-time Pose Estimation Experiments

We test our localization algorithm in different kinds of environments. In order to realize localization in a given 3D map, we need to create the global map. In our system, LOAM [33] is used to create the 3D map. The LOAM algorithm can build very accurate point cloud map by using a rotating 2D laser. In all the experiments, we set our map resolution to 4cm. We test the odometry and localization algorithms in different kinds of environment by carrying or semi-autonomously flying our customized MAV. Our customized quadrotor is equipped with a forward-looking Asus Xtion Pro Live RGB-D camera and Odroid XU embedded computer. The RGB-D camera is used for odometry estimation and localization. We develop our localization system using ROS Indigo, PCL 1.7, OpenCV 2.4 and C++ language.

8.1.1 Fast Direct RGB-D Odometry

Since the odometry performance influences the localization significantly, we test our RGB-D odometry method with other state-of-the-art method to shown its excellent performance. Here, we compared the proposed RGB-D odometry method to Fovis, DVO and FastICP. We test them both in normal indoor environments and visual-degraded environments. We show that our odometry method totally outperform state-of-the-art RGB-D odometry methods in dark environment and can achieve similar performance in normal environment. The first experiment was in relative dark and smoky environments, which is the actual environment our MAV will operate in. The second experiments was in long corridors, which is very challenge for both visual or depth based odometry estimation methods since there are only very few visual and geometric features. We evaluate our odometry estimation pipeline wth RGB-D data recorded two indoor environments. The experimental results are shown in Fig.1. The camera is moved along a loop, and placed back at its starting point. Images are streamed at VGA resolution. We use the loop-closing error to evaluate the performance of each method. The loop-closing error and runtime performance is the following table.

8.1.2 Illustrative Localization Examples

In this part, the localization algorithm is tested in visual degraded environment and natural office environment. In the degraded visual environments, some locations

Table 2: Loop-closing Error and Runtime Performance

Method	Test1	Test2	Mean Runtime(ms)
Fovis	failed	6.2%	20.5
DVO	17.09%	2.60%	51.2
FastICP	7.03%	5.6%	50.3
Our	3.89%	1.50%	10.9

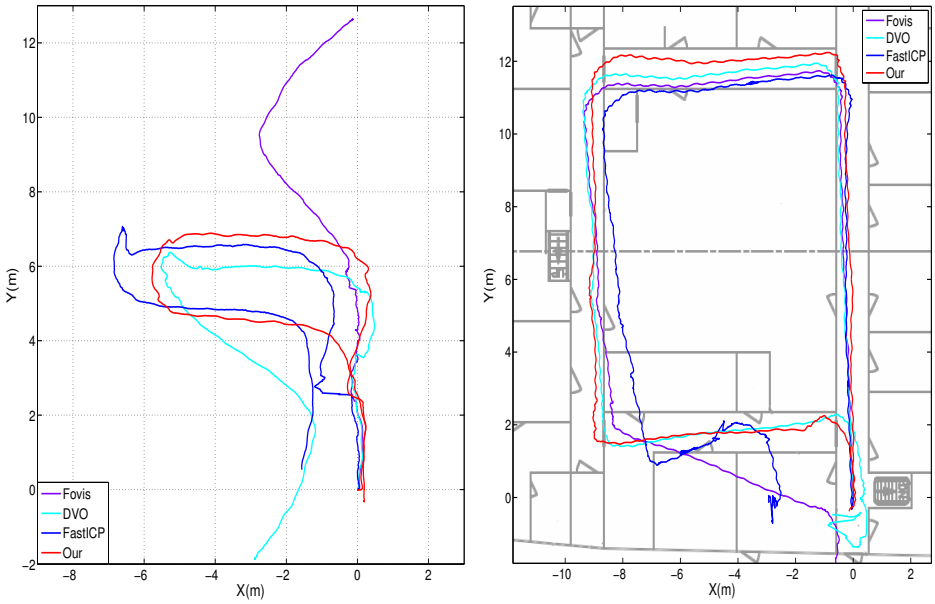


Figure 11: Odometry evaluation in different environments: The left plot shows estimated trajectory in an low illumination environment. The right plot shows the estimated trajectory in a challenging long corridor environment.

are very dark and some locations have very few visual or geometric features. In the natural office environment, there are some long clear corridors which pose great challenge for odomtry estimation and localization. We show that our localization system can work well in those environments. In all experiments, the RGB-D images are streamed at frame rate of 15Hz with resolution of 320×240

1) Degraded Visual Environment We test our localization algorithm in two degraded visual environments. In both two experiments, the illumination is very low. The difference is that one is a cluttered and narrow environment, while the other one is more structured but almost totally dark.

The first experiment is in the narrow and cluttered environment, which has a size of 16m $\times$ 25.6m $\times$ 4.04m. In this environments, most of the time the RGB images are very dark as shown in Fig. 12, while the depth images are still very good. However, there are some locations that the robot can only see one flat wall in front of it. For example, when the robot turns left at corner (a), since the corridor is very narrow (less than 1m), the robot can only see the wall in front of it. Another example is that when the robot is in the spacious room (b), the depth camera can only see the ground plane and cannot see the wall in front of it. In both scenarios, if just depth images are used for odometry estimation, it will suffer from the degeneration problem. In our system, we always monitor the degeneration status, when the degeneration is severe, RGB information is considered to estimate the odometry. By doing so, our odometry method will not only avoid suffering from a severe degeneration problem, but also has a very fast speed and low CPU usage. The localization result in this environment is shown in Fig. 12.

The second experiment is in a structured but almost completely dark environment, which has size of 11.8m $\times$ 19.2m $\times$ 2.8m. In this environment, we cannot get any useful information from RGB images. Therefore, we can only use depth images for odometry estimation and localization. There are also some challenging locations where 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 that depth image returns nothing because the minimum measurement range of the RGB-D camera is around 0.5 meters. In such situations, the depth-based odometry will also suffer from the degeneration problem. In this experiment, we also detect the degeneration status. If the degeneration is severe, the odometry estimation method will not output motion estimation results, but a failure indicator. Then, our localization algorithm will use the noise-driven motion model to propagate particle set. In our experiment, we find that if the odometry failure is relatively short in duration (less than 3 seconds), it is possible for the localization algorithm to overcome this failure entirely. The localization result in this experiment is shown in Fig. 13.

2) Typical Office Environment

In this experiment, we want to show that our localization system not only works in degraded visual environments, but also works well in normal challenging environments. The test environment is a typical office environment with long clear corridors, which has a size of 64.2m $\times$ 21.2m $\times$ 3.9m. In this environment, the illumination is very good. However, there are also several challenges in this environment for odometry estimation and localization using RGB-D camreas. First, the corridors are very clear. Therefore, there are only few visual features

Figure 12: Localization in degraded visual environment: Pink: Odometry, Red: Localization. The bottom pictures show some snapshots of the environment. The top figure shows the odometry, localization results with the 3D octomap. Note that ceiling and ground are cropped for visualization purpose (same in the later figures).

and geometric features in the corridors, which poses big challenges for odometry estimation and localization. Second, the corridors are very narrow, therefore when the robot turns from one corridor to another corridor, the RGB-D camera can only see a part of the wall. Therefore, the localization system must be robust enough, otherwise it will easily fail around each corner. The third challenge is that the maximum measurement range of RGB-D camera is about 6$\sim$7m and the

Figure 13: Localization in completely dark environment: Pink: Odometry, Red: Localization. The top figures shows the odometry, localization results with the 3D octomap. The bottom pictures show some snapshots of the environment.

measurement noise increases along with the distance. However, in this environment there are several corridors whose length are longer than 10 meters. Both the odometry estimation method and localization method should find useful constraints for estimation. In our experiment, we found our localization system can robustly locate the robot in the map. Fig. 14 shows the localization results.

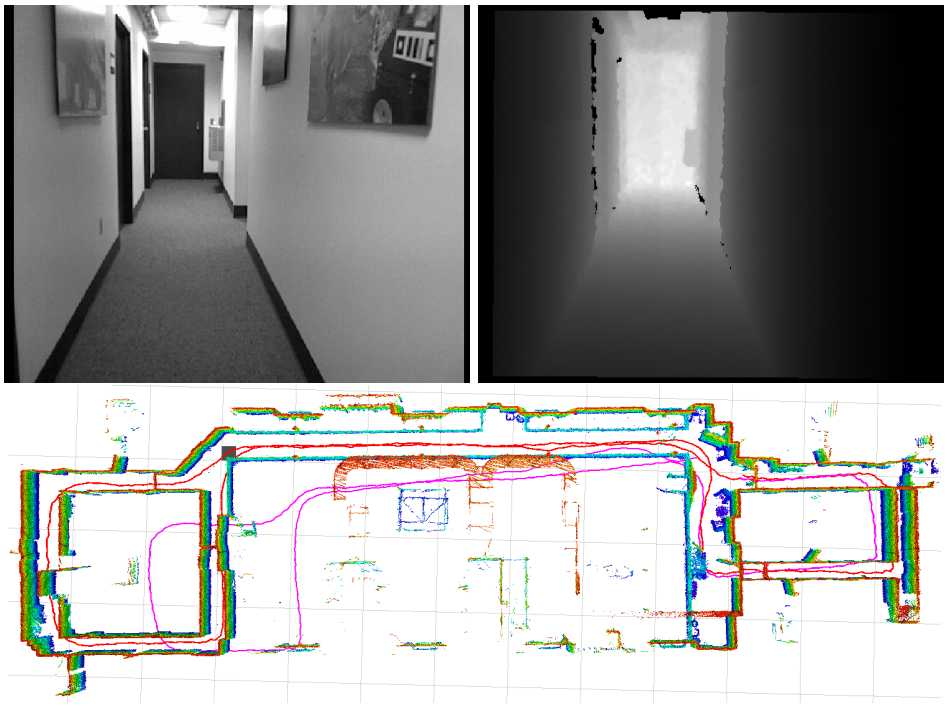


Figure 14: Localization in typical office environment: Pink: Odometry, Red: Localization. The top pictures show some snapshots of the environment. The bottom figure shows the odometry, localization results with the 3D octomap.

8.1.3 Localization Accuracy

In this part, we compare the localization accuracy with ground truth from LOAM mapping system. We attached an Xtion RGB-D camera to the LOAM system and recorded the datasets for offline comparison. Since the estimation accuracy of LOAM system is very high, we could consider its trajectory as ground truth. We test our localization algorithm in two environments. One is a general office environment, where there are many chairs, long tables, long corridors and a lot of office furnitures. This environment is much easier for odometry estimation and localization, since there are lots of visual and geometric features. The other one is in a long tunnel, which is very difficult for odometry estimation and localization using a RGB-D camera since it is very clear. For both experiments, the map resolution is 4cm and the particle number is set to 500. The localization algorithm updates the pose when the robot moves every 10cm or turns 0.1 radians. The experimental results are shown in Table. 3. From the experimental results, we also

can see that localization accuracy in office environment is better than in long tunnel environment. In long tunnel environment, the biggest error is in the x direction since sometimes there are not enough constraints to determine its position. But our localization algorithm can quickly converge to the true position once there are enough constraints available. The accuracy of our localization algorithm is better than others work [24] and [34]. In their work, their mean localization error is about 40cm, while ours is about 17cm (Note that in our localization algorithm, the observation update is executed only after the robot moves every 10cm or turns 0.1 radians). It should be noted that the localization accuracy changes in different environments or moving at different speeds because it influences the accuracy of odometry estimation dramatically.

Figure 15: Accuracy comparison with ground truth in two different kind of environments: Pink: RGB-D Odometry Red: Localization Blue: Ground truth.

Table 3: Localization Accuracy for Datasets Shown in Fig.15

Environments	Distance	RSME	Mean	Std
Office	47.2m	0.161m	0.152m	0.056m
Tunnel	46.1m	0.235m	0.194m	0.107m

8.1.4 Runtime Performance Evaluation

Runtime performance is very important for MAVs since the onboard computation abilities are limited. In addition to odometry and localization, it is also necessary

to run path planning and obstacle avoidance for completing a given task. In our experiment, we test the runtime performance of our system on the Odroid XU system, which has two CPUs. One is a 1.6GHz, quad core CPU. The other one is a Cortex−A7 quad core CPU. Each core has one thread. Our odometry and localization algorithms are both single-threaded programs. Therefore, each algorithm takes one core. In our experiment, the RGB-D data are all recorded at frame rate 15Hz with QVGA resolution. For the experiment in Fig. 12, the runtime performance is shown in Table. 8. In our experiment, we use 300 particles. Our algorithm can run up to 30Hz on the embedded system. When it is running at 15Hz, the CPU usage is very low which leaves many computation resources for path planning and obstacle avoidance.

Table 4: Runtime Performance on an Embedded Computer

Name	Algorithm Runtime			
	Mean	Min	Max	StdDev
Odometry	30.3ms	5ms	110ms	20.2ms
Localization	65.8ms	45.8ms	97ms	16.5ms
Total CPU Usage	34.5%	30.5%	44%	2.80%

8.2 Planning Experiments

To test motion planning, we need to first provide the two inputs to motion planning: local goal generated by mission planner and distance map generated by obstacle mapping. Local goal could also be specified by a human.

8.2.1 Mission Planner

Mission planning serves to provide local goal for local planning, which are about 5m away from each other. It is computed by the following three steps:

(1) Generate 2D grid map from 3D point cloud ship model. Firstly, project all the 3D points onto 2D grid map. Then count the number of points falling onto each grid. If it exceeds a threshold, the grid will be considered as occupied. An example of 3D point cloud and corresponding 2D grid map is shown in Fig .16(a) and Fig .16(b).

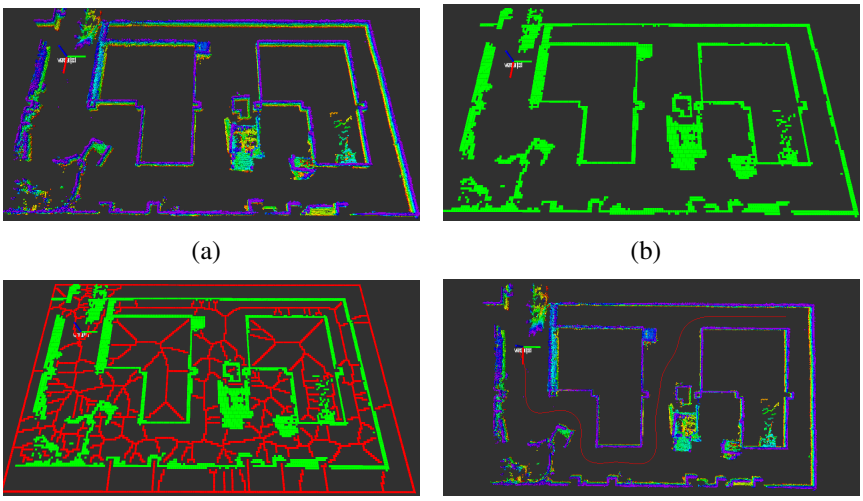


Figure 16: (a) 3D offline point cloud of an indoor environment (b) 2D grid map projected from 3D point cloud. Green grids represent the obstacles. (c) Voronoi diagram shown as red line. It represents the grids farthest away from green obstacles. (d) Final path from start to goal using A* search algorithm.

(2) Build Voronoi diagram of the grid map. Firstly, build a distance map from the grid map, which stores the distance of each grid to the closest obstacle using incremental brushfire algorithm. Then find the grid with equal distance to obstacles and put it one the voronoi diagram as as to keep the vehicle most away from obstacles shown as red lines in Fig .16(c).

(3) Search the optimal path using A* from current location to goal. Smoothing and collision detection is implemented later to get a feasible and smooth path shown in Fig .16(d). Then the path is downsampled by about 5m away from each other to provide local goal for local planning.

Note that though we have the prior map, we cannot directly use it to generate an offline path because it doesn't adapt to the unknown obstacles. More seriously, if there is big state estimation error in dark or smoky environments, blindingly following offline path can be disastrous to MAV because it has no sense where the obstacles are. Instead, our online obstacle mapping and local planning can guarantee the safety.

8.2.2 Obstacle Mapping

The corridor our vehicle is flying contains complicated 3D protruding objects such as lamps, wires and stairs, posing difficulty for obstacle avoidance. So a 3D occupancy map is needed to represent the world. It stores the probability of each grid being occupied, which is updated using Bayes' rule according to the range sensor data from depth camera. If the probability exceeds a threshold, the grid will be considered as obstacles. An example of occupancy map is shown in Fig .17(a). Suppose $b(m)$ stores the belief of a grid being occupied, o represents sensor observation at this grid. Then the update of belief is given by:

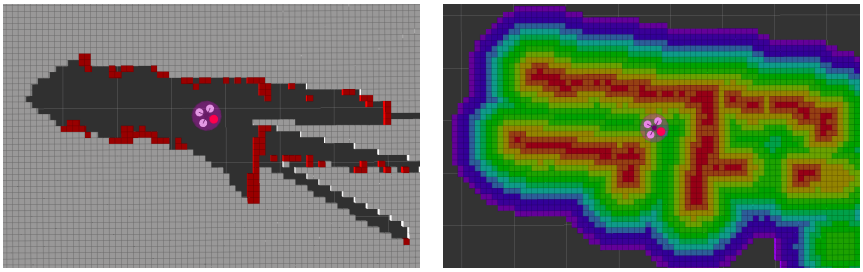
$$\begin{aligned} b(m|o) &= b(m) + k * 20 \\ b(m|\bar{o}) &= b(m) - k * 5 \end{aligned} \tag{34}$$

The derivation of update rule could be found at [35]. The first equation means if sensor ray hits the cell, it increases the probability of being obstacles. The second equation means if ray passes through the cell, probability decreases. The value of 20 and 5 is tuned through experiments depending on the threshold set for obstacle state. The improvement over [35] is that we add a scaling number k between 0 and 1 to represent the quality of sensor observation. It could increase the accuracy of occupancy estimate and reduce the unnecessary updates due to bad state estimation. It is determined based on the following two factors:

1) The quality of state estimation. If the state estimation is of high quality, meaning that the vehicle is very certain about current its position and obstacles position, k is set close to 1. The quality of estimation is measured by the determinant $|Q|$ of pose covariance matrix Q, computed by the particle distribution in MCL localization in Section 4.1.2.

2) The distance to the vehicle. If the cell is further away from vehicle, sensor data is likely to be un-accurate, so k is set small. This is reasonable for Kinect camera since its reliable range estimation is about 3.5m.

To keep memory and computation efficient, the occupancy map keeps a fixed dimension and moves along with the vehicle. The obstacle grid in occupancy map is then used to compute 3D distance map, which stores the distance of each grid to its closest obstacles shown in Fig .17(b). It is computed by the incremental brushfire algorithm [12]. Distance map is a key element required for local planning to avoid obstacles.



(a) (b)

Figure 17: (a) Occupancy map. Four spheres are the four rotors of quadrotor. Red cubes represent obstacles. Hollow area presents free space. Grey cubes means the unknown area. (b) Distance map. The redder cube, the more closer to obstacles.

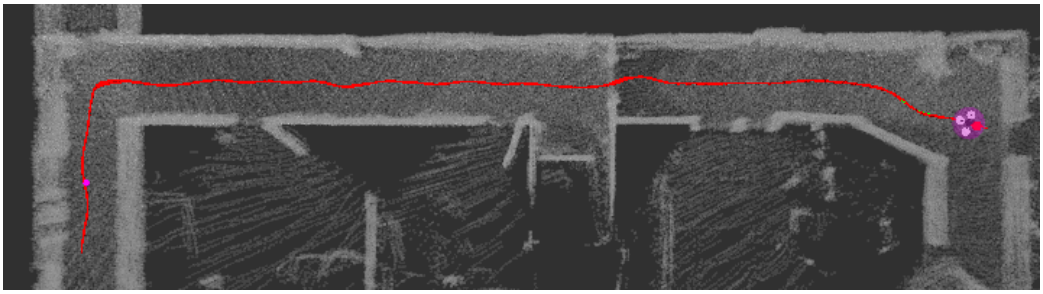


Figure 18: An example trajectory calculated using path optimization with receding horizon control through a simulated shipboard environment.

8.2.3 Local Planner

Real point cloud data is used to test motion planning. A depth camera is simulated by raycasting, then mission planner generates a series of local goals and obstacle mapping provides the occupancy map and distance map. Local planning keeps replanning to reach them while avoiding obstacles. The pose history during simulation is shown as red curve in Fig. 18.

Running performance on Odroid and quality of trajectory is evaluated in Table . 5. The proposed algorithm can run at 30Hz. Obstacle detection and motion planing takes about 10% on the onboard CPU. Our mean obstacle distance is 0.47 m which lies nearly in the center of a 1 m wide corridor. The closest distance is 0.18 m happening at the door. From the prior point cloud, the door is about 0.44 m wide so the vehicle is nearly in the center of door.

To demonstrate the performance, we make comparison by replacing our path

Table 5: Motion planning simulation performance on 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
Obstacle distance	0.47 m	0.18 m	0.66 m	0.07 m

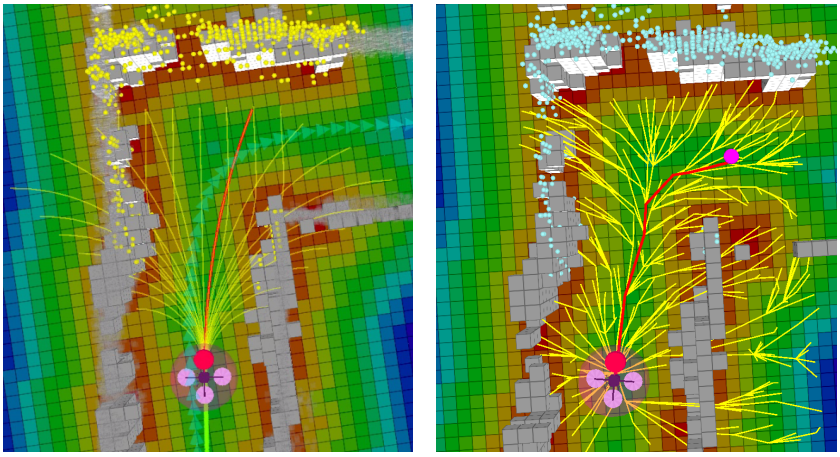
planner with RRT* [36] and keeping other modules such as spline fitting and obstacle mapping as the same. RRT* cost function is set as path length and obstacle cost. To bias RRT*, the local goal point provided by global planner is set close to the start position (about 2 m) to greatly decrease the search space. The comparison is implemented on the embedded computer and the result is shown in Table .6. RRT* needs more time than our method to stably generate a valid path. It is closer to obstacles and has higher snap cost compared to our method. The reason our method outperforms RRT* is mostly due to the fact that corridor is a structured environment where obstacles usually lie on two sides so the provided path set is easy to get a smooth and safe path while RRT* needs many random samples in order to get a valid path shown in Fig. 19.

Table 6: Comparison with RRT*. Dist stands for vehicle distance to the obstacle. Opti means CHOMP optimization.

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
RRT*+Opti	94	0.46	0.17	0.79	4.16
Our	30	0.47	0.18	0.58	2.50

8.3 Control Experiments

In this experiment, the robot is programmed to continuously fly through a circle pattern, with 1m radius and 0.32m/s average speed. State estimation only comes from flight controller unit using inertial sensors and optical flow sensor. RGB-D Visual odometry and localization is not employed in this control test. The command polynomial trajectory is continuously generated onboard based on the predicted pose of quadrotor. It not only needs to follow position x, y, z but also



(a) (b)

Figure 19: (a) Our path library planning. The grey cubes are the occupancy grid. Colorful square is the distance map. Yellow curves are the offline path set and the best one is shown in red. (b) RRT* planning. Yellow segments shows the RRT tree. The best searched path is shown in red.

heading $\psi(yaw)$, which is tangential to the circle. The tracking result is shown in Fig .20. Evaluation of each DOF tracking is shown in Fig. 21. The mean tracking error from state estimation is 0.13m with standard deviation 0.05m.

8.4 Final Demo Experiments

8.4.1 Mission Description

For the demo, the mission of the MAV is to search in a partial-known environment and find the fire burning places and create the temperature map of the whole environment. To accomplish this goal, our robot uses onboard RGB-D camera (only depth images are used) for autonomous navigation and FLIR infrared camera to detect the temperature of environment and mark the high temperature areas.

We first use a hand-held mapping device to create the global map of the whole demo area. This global map is used for both localization and global planning. The global map created from the mapping device is converted to octomap and 3D Euclidean distance map for global localization. The resolution is 4cm. Fig.22 shows the created point cloud map. Our goal is to launch the MAV around "start point", then let the robot autonomously explore the 1 m wide and 20 m long

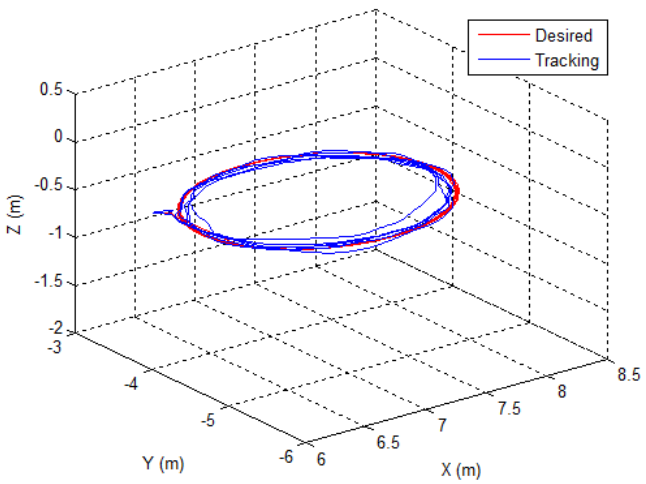


Figure 20: Circular flight control test

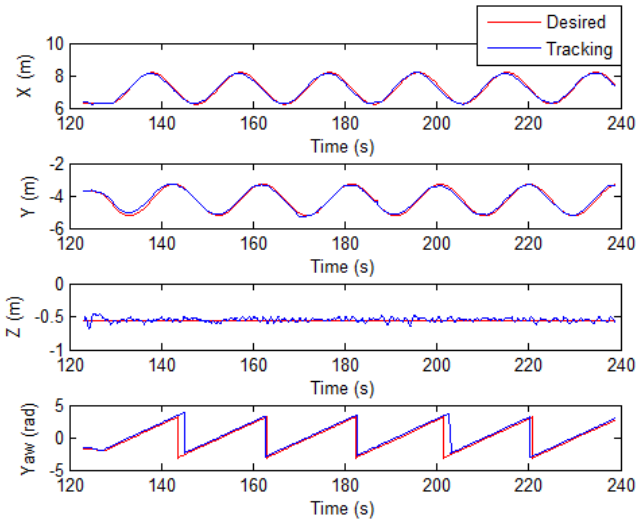


Figure 21: Tracking result of x, y, z, yaw

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.

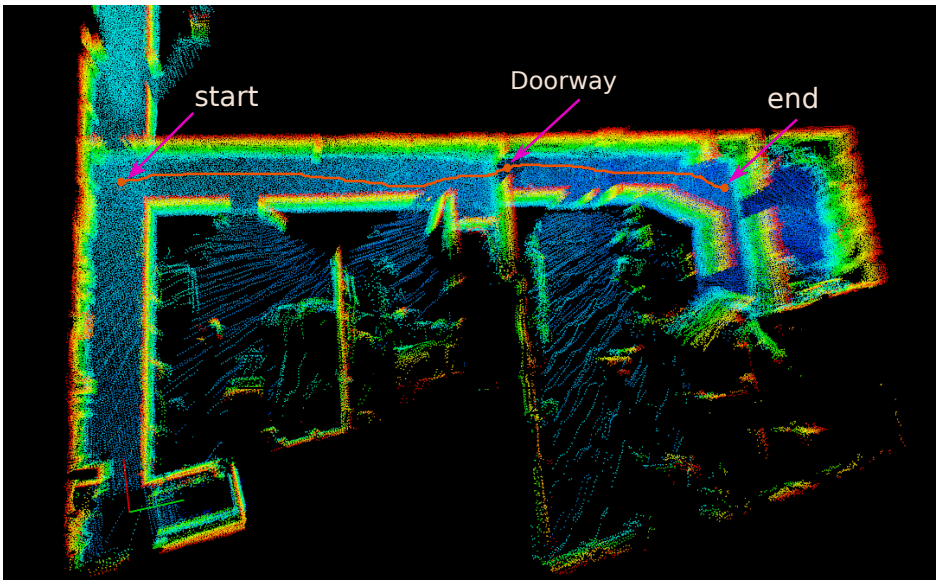


Figure 22: Global Point Cloud Map: The goal is to launch the MAV around ”start point”, then let the robot autonomously explore the long corridor, go through the narrow doorway and go to the end point. At the same time, the robot records infrared images and detects the fire and high temperature areas and puts markers on the map for rescuers to see where the fire is.

We did three kinds of experiments to test our system. And for each kind of experiment, we did at least 4 successful experiments. We simulated the fire fighting scenario, where some places have good illumination, while some places are fully dark and some places are filled with smoke and fire. Those environments pose different challenges for our robot. We want to test the performance of our system in those different kinds of practical environments.

8.4.2 Experimental Results of Autonomous Flights

We performed totally 20 experiments of autonomous flights in this demo area with different environmental conditions.

1) Test1: With illumination In test1, all the lights in the hallway are on, which is a little bit better for the optical flow estimation. However, in our local-

ization system we don't use any RGB information, therefore it doesn't influence the localization performance too much. Actually with all lights on, it is not good for the depth camera, since there are no depth values return from those bright lamps. The reason is that our depth camera is based on structured lights. Fig. 23 shows the experiment results, our robot can realize reliable localization in this test area. As you can see from the Fig2, the visual odometry drifted to the outside of the map, while localization is still good. It should be noted that the robot trajectory is not smooth which not means bad localization results. Actually, our robot was well localized in this test. One reason why the robot is not very stable is that the hallway is very narrow, therefore the airflow really influenced the control performance of the robot. Besides, there are some small objects and thin lines on the wall, every time the robot hit those objects the trajectory of the robot was changed.

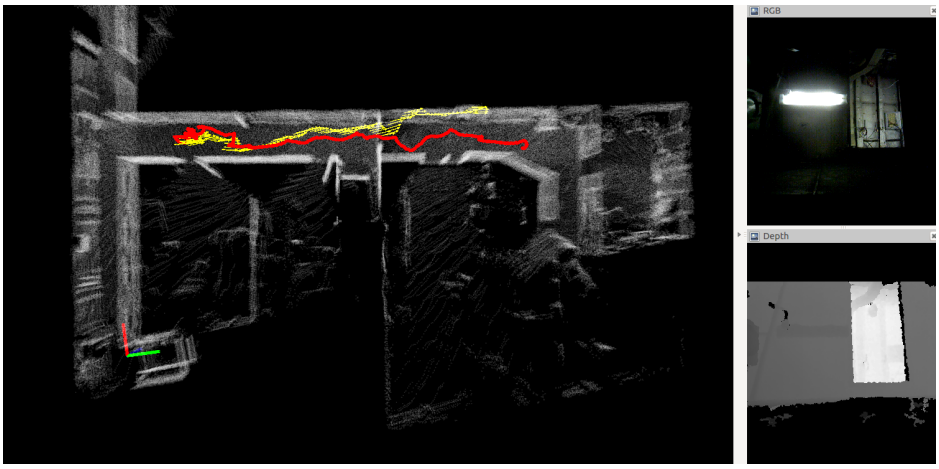


Figure 23: Visual odometry and localization results in normal environment during autonomous flights (Yellow arrows are visual odometry and red trajectory is localization result)

2) Test2: With no illumination For test2, we turned off all the lights in the test area and wanted to test the ability of autonomous navigation in totally dark environment. The reason of doing this is that nowadays most navigation system of MAVs are using visual information which couldn't work in dark environment. While our method uses depth information only, which not only work in environment with abundant illumination but also work in environments where are no visual features. In this experiment, our robot was well localized and success-

fully went through the very narrow doorway for several times as shown in Fig.24. Experimental results show that our system are work very well in fully dark environments.

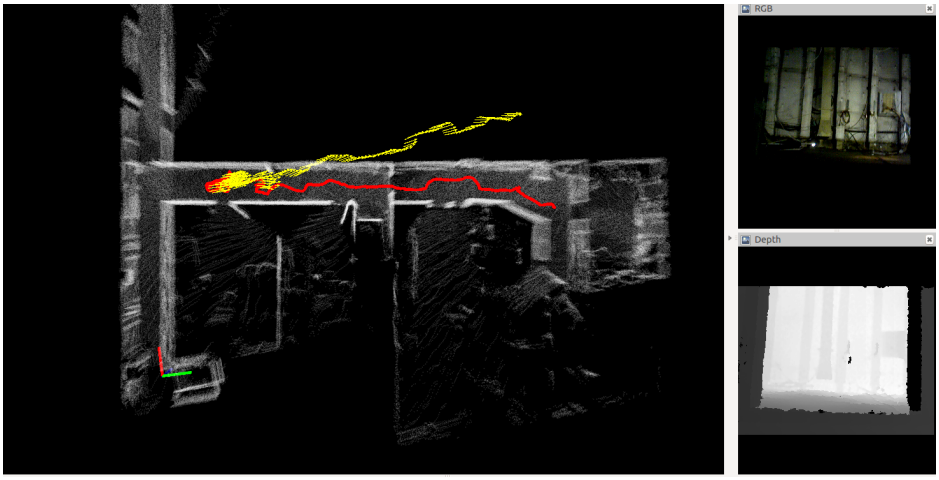


Figure 24: Visual odometry and localization results in dark environment during autonomous flights (Yellow arrows are visual odometry and red trajectory is localization result)

3) Test3: With smoke In this test, we want to test the ability of our navigation system in smoky environments. We did the test in the same area but filled with wood fire smoke. The smoke in this test is not very dense, we found that even the performance of the depth camera is reduced dramatically, our system can still work. As you can see from Fig.25, the drift of the visual odometry is much bigger than previous tests(especially the drift in Z direction). However, the localization system still worked very well (As you can see the robot went back to the start point). However, when the smoke is very dense, then the depth camera almost dosen't return any useful depth information, which will make the whole system fail. Therefore, navigation in dense smoky environments is still an open problem.

Discussion Some images of the flight are shown in Fig. 26. The result of 20 runs is shown in Table 7. Failure cases are usually due to quadrotors being slightly rotated and stuck in the narrow door. It is difficult to cross the door in smoky environment because the depth image is corrupted by smoke making difficult for state estimation, obstacle detection and trajectory tracking difficult. Experimental results show that our robot can work very well in all the conditions except when

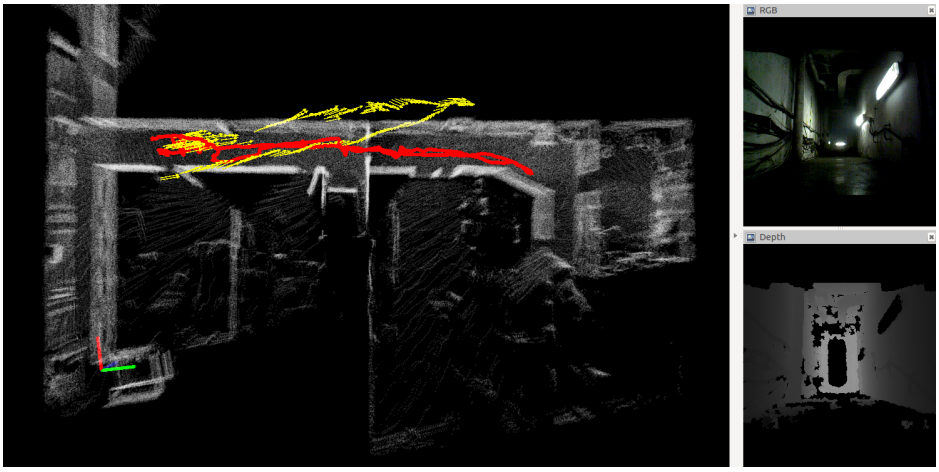


Figure 25: Visual odometry and localization results in smoky environment during autonomous flights (Yellow arrows are visual odometry and red trajectory is localization result)

the environment is filled with very dense smoke.

(a) (b)

Figure 26: Experiment demonstration. Left: turning. Right: crossing door.

Runtime performance is also very important for MAVs since the onboard computation abilities are limited. We record the performance including CPU usages of some key algorithms on the Odroid system shown in Table. 8. We use 300 particles for MCL localization. When all the system modules are running, the total CPU usage is between $60 \sim 65\%$. The experiment result shows our navigation system can run in real-time by only using the onboard computation resources.

The video of a field experiment with fire detection at Shadwell in Nov 2014 can be found at https://www.youtube.com/watch?v=g3dWQCECwlY.

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

Table 7: Results of experiments.

Table 8: Runtime Performance on the Embedded Computer

Name	Algorithm Runtime			
	Mean	Min	Max	StdDev
Odometry	30.3ms	5ms	110ms	20.2ms
Localization	65.8ms	45.8ms	97ms	16.5ms
Local Planning	29.2 ms	15.2 ms	37.8 ms	6.7 ms

8.4.3 Lessons Learned

During the experiments, we did several modifications to make the system more robust and accurate. We found that if we didn't make those modifications the whole system didn't work very well.

1) Depth Camera Calibration The depth value from the Xtion Pro Live depth camera is very noisy. And the noise increases with the measurement distance. For example, when the measurement distance is about 1 3 meters, the noise is just around 2 3cm. But when the measurement distance is about 5 meters, then the noise is around 7 cm. Another problem is that the depth camera always underestimates the actual distance. If we didnt do the depth calibration, then the close points from the point cloud are correct, but the far points are severely underestimated. That means, far objects in the point cloud cannot be aligned to the global map correctly, which often made the whole localization system fail.

2) Adding UKF Filtering Our visual odometry outputs the transform at 15Hz, and this estimation is fed into FCUs EKF filter to fuse with other data for pose control. We found that if we just use the visual odometry, it seems that it is not fast enough for EKF filtering. The latency sometime makes 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 then before. The possible reasons are: first, by fusing IMU and visual odometry, we can get a smoother odometry estimation;

second, the latency is dramatically reduced when using the UKF output since our UKF runs at 50Hz.

9 Conclusion

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 of any additional infrastructure on the ship reducing the installation and maintenance costs of the system. We achieved autonomous flight with fully online and onboard control and state estimation in complete dark through 1m wide passages while crossing doorways with only 8cm clearance. We demonstrated 10 consecutive runs where the vehicle crossed a lit, completely dark, smoky passageway respectively and ended by detecting wood and diesel fires.

The next challenges are to increase to robustness and safety of the vehicle while increasing flight time. This will involve improvements in both software and hardware. The current size of vehicle is a little large, resulting in a very tight fit through the ship doorways. In 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 which extend the range of environments 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.

References

[1] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. A fully autonomous indoor quadrotor. *Robotics, IEEE Transactions on*, 28(1):90–100, 2012.

[2] Ivan Dryanovski, Roberto G. Valenti, and Jizhong Xiao. An open-source navigation system for micro aerial vehicles. *Autonomous Robots*, 34(3):177–188, March 2013.

[3] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous multi-floor indoor navigation with a computationally constrained mav. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 20–25. IEEE, 2011.

[4] Konstantin Schauwecker and Andreas Zell. On-board dual-stereo-vision for the navigation of an autonomous MAV. *J. Intell. Robot. Syst. Theory Appl.*, 74:1–16, 2014.

[5] Friedrich Fraundorfer, Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor MAV. In *IEEE Int. Conf. Intell. Robot. Syst.*, pages 4557–4564, 2012.

[6] Allen D. Wu, Eric N. Johnson, Michael Kaess, Frank Dellaert, and Girish Chowdhary. Autonomous Flight in GPS-Denied Environments Using Monocular Vision and Inertial Sensors. *J. Aerosp. Inf. Syst.*, 10:172–186, 2013.

[7] D Scaramuzza, M Achtelik, L Doitsidis, F Fraundorfer, E Kosmatopoulos, A Martinelli, et al. Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in gps-denied environments. pages 26–40, 2014.

[8] Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments. *Journal of Field Robotics*, 28(6):854–874, 2011.

[9] Gerardo Flores, Shuting Zhou, Rogelio Lozano, and Pedro Castillo. A vision and gps-based real-time trajectory planning for a mav in unknown and

low-sunlight environments. *Journal of Intelligent & Robotic Systems*, 74(1-2):59–67, 2014.

[10] AS Huang and Abraham Bachrach. Visual odometry and mapping for autonomous flight using an RGB-D camera. *Int. Symp. Robot. Res.*, pages 1–16, 2011.

[11] Roberto G Valenti, Ivan Dryanovski, Carlos Jaramillo, Daniel Perea Strom, and Jizhong Xiao. Autonomous quadrotor flight using onboard rgb-d visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 5233–5238. IEEE, 2014.

[12] Sebastian Scherer, Joern Rehder, Supreeth Achar, Hugh Cover, Andrew Chambers, Stephen Nuske, and Sanjiv Singh. River mapping from a flying robot: state estimation, river detection, and obstacle mapping. *Autonomous Robots*, 33(1-2):189–214, 2012.

[13] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2520–2525. IEEE, 2011.

[14] P.J. Besl and H.D. McKay. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[15] T. Stoyanov, M. Magnusson, H. Andreasson, and A. J. Lilienthal. Fast and accurate scan registration through minimization of the distance between compact 3D NDT representations. *Int. J. Rob. Res.*, 31(12):1377–1393, September 2012.

[16] Kaustubh Pathak, Andreas Birk, Narunas Vaskevicius, and Jann Poppinga. Fast Registration Based on Noisy Planes With Unknown Correspondences for 3-D Mapping. *IEEE Trans. Robot.*, 26(3):424–441, June 2010.

[17] Berthold K.P. Horn and John G. Harris. Rigid body motion from range image sequences. *CVGIP Image Underst.*, 53(1):1–13, January 1991.

[18] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *Int. J. Rob. Res.*, 31(5):647–663, February 2012.

[19] Graeme Jones. Accurate and Computationally-inexpensive Recovery of Ego-Motion using Optical Flow and Range Flow with Extended Temporal Support. In *Procedings of the British Machine Vision Conference 2013*, pages 75.1–75.11. British Machine Vision Association, 2013.

[20] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Robust odometry estimation for RGB-D cameras. In *2013 IEEE Int. Conf. Robot. Autom.*, pages 3748–3754. IEEE, May 2013.

[21] Zheng Fang and Sebastian Scherer. Experimental Study of Odometry Estimation Methods using RGB-D Cameras. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2014.

[22] Shuji Oishi, Yongjin Jeong, Ryo Kurazume, Yumi Iwashita, and Tsutomu Hasegawa. ND voxel localization using large-scale 3D environmental map and RGB-D camera. In *2013 IEEE Int. Conf. Robot. Biomimetics*, number December, pages 538–545. IEEE, December 2013.

[23] Daniel Maier, Armin Hornung, and Maren Bennewitz. Real-time navigation in 3D environments based on depth camera data. In *IEEE-RAS International Conference on Humanoid Robots*, pages 692–697, 2012.

[24] Maurice F. Fallon, Hordur Johannsson, and John J. Leonard. Efficient scene simulation for robust monte carlo localization using an RGB-D camera. In *Proc. - IEEE Int. Conf. Robot. Autom.*, pages 1663–1670, 2012.

[25] Sebastian Thrun, D Fox, and W Burgard. Monte carlo localization with mixture proposal distribution. *AAAI/IAAI*, 2000.

[26] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[27] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Proc. Third Int. Conf. 3-D Digit. Imaging Model.*, pages 145–152. IEEE Comput. Soc, 2001.

[28] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[29] Nathan Ratliff, Matthew Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

[30] D. Dey, K. Shaurya Shankar, and et al Zeng, S. Vision and learning for deliberative monocular cluttered flight. *arXiv preprint arXiv:1411.6326*, 2014.

[31] Gene H Golub, Per Christian Hansen, and Dianne P O'Leary. Tikhonov regularization and total least squares. *SIAM Journal on Matrix Analysis and Applications*, 21(1):185–194, 1999.

[32] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[33] Ji Zhang and Sanjiv Singh. LOAM : Lidar Odometry and Mapping in Real-time. *Robotics: Science and Systems Conference (RSS)*, 2014.

[34] Joydeep Biswas and Manuela Veloso. Depth camera based indoor mobile robot localization and navigation. In *IEEE Int. Conf. Robot. Autom.*, pages 1697–1702, 2012.

[35] Sebastian Scherer, Sanjiv Singh, Lyle Chamberlain, and Mike Elgersma. Flying fast and low among obstacles: Methodology and experiments. *The International Journal of Robotics Research*, 27(5):549–574, 2008.

[36] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *arXiv preprint arXiv:1005.0416*, 2010.