Real-time Vision-based Autonomous Navigation of MAV in Dynamic Environments

Jiahao Lin

Master of Science Thesis

TU Delft Delft University of Technology

Cognitive Robotics

Real-time Vision-based Autonomous Navigation of MAV in Dynamic Environments

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Jiahao Lin

October 17, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Safe navigation in unknown environments is a challenging task for autonomous Micro Aerial Vehicle (MAV) systems. Previous works generally avoid obstacles by assuming that the environment is static. The purpose of this thesis work is to develop a MAV system that can navigate autonomously and safely in dynamic environments. We present an onboard vision-based approach for the avoidance of moving obstacles in dynamic environments. This approach uses a state-of-art visual odometry algorithm to estimate the pose of MAV and an efficient obstacle sensing method based on stereo image pairs to estimate the center position, velocity, and size of the obstacles. Considering the uncertainties of the estimations, a chance-constrained Model Predictive Controller (MPC) is applied to achieve robust collision avoidance. The method takes into account the MAV's dynamics, state estimation and the obstacle sensing results ensuring that the collision probability between the MAV and each obstacle is below a specified threshold. The proposed approach is implemented on a designed experimental platform that consists of a quadrotor, a depth camera, and a single-board computer, and is successfully tested in a variety of environments, showing effective online collision avoidance of moving obstacles.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my sincere gratitude to my supervisors Dr. Javier Alonso-Mora and Dr. Guido de Croon for offering me this opportunity to investigate my topic of interest as well as for their patience and support during the research.

Besides, I would like to thank my mentor Hai Zhu for his excellent assistance throughout the research.

Finally, I also want to thank my parents and all my friends for accompanying and supporting me all the time.

Delft, University of Technology Jiahao Lin
October 17, 2019

Chapter 1

Introduction

1-1 Background

Over the last decade, Micro Aerial Vehicles (MAVs) have captured the attention of both academia and industry because of their mechanical and control simplicity, high maneuverability and low cost of entry point [7]. These features of MAVs make them popular aerial robots in a large number of robotics applications such as package delivery, cinematography, and surveillance.

Figure 1-1: A pilot flies a MAV over a field of strawberries to study how reflectance data may help detect outbreaks of spider mite [1].

However, these aerial vehicles used to be operated by professional pilots in order to accomplish different tasks (an example is shown in Figure 1-1), since it is challenging for these aerial vehicles to navigate safely and autonomously in clutter environments. Especially in real-world scenarios, aerial vehicles suffer from imperfect onboard sensors and power-limited computers, fast and aggressive flights and dynamic environments. Recent researches on attempts to narrow the gap between piloted and pilotless aerial vehicles have demonstrated amazing progress. Figure 1-2 shows several autonomous MAV applications including drone delivery, cinematography and agriculture.

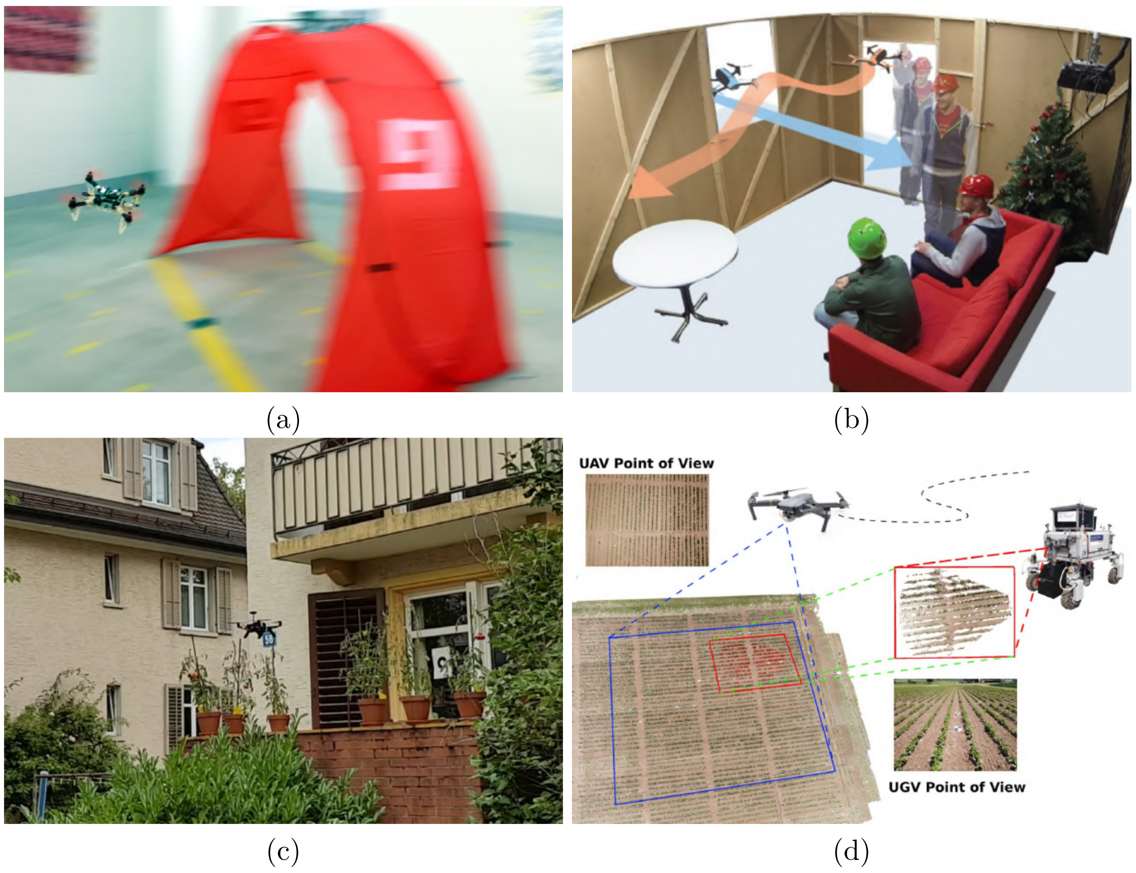


Figure 1-2: Several autonomous MAV applications. (a) An autonomous racing drone is passing through a checkpoint [2]. (b) A snapshot of automated drone videography in a dynamic scene [3]. (c) A delivery drone is approaching the drop-off point on a balcony [4]. (d) Aerial-ground collaborative 3D mapping for precision farming with an Unmanned Aerial Vehicle (UAV) and an Unmanned Ground Vehicle (UGV) [5].

One example is the autonomous drone racing [8], a study of Kaufmann et al. [9] shows that their learning-based autonomous racing system can complete the racing track with speeds of up to 3.5 m/s and achieve a higher success rate than a professional pilot. Another example is the automated aerial videography, which can reproduce camera's motions from recorded footage more precisely than a human pilot. Nevertheless, unlike autonomous drone racing and cinematography, whose coarse map may be given as a priori information, the autonomous navigation in unknown environments is more demanding and is a longed-for solution for various real-world applications, such as searching and rescuing in disaster scenes and package delivery.

In recent years, different vision-based autonomous MAV systems have been developed with the aim to automate safe flights in unknown environments [10, 11, 12, 13, 14]. The main challenge of vision-based autonomous navigation is that MAVs are required to track a set of waypoints and avoid collisions with only the onboard sensing and computing during flights. To achieve this, most of the previous approaches follow a similar pipeline as shown in Figure 1-3. For the waypoint tracking, the aforementioned approaches generally take the MAV's odometry, estimated with the onboard visual and inertial information, as the feedback of waypoint tracking. And as for the environment perception, most of them use visual or ranging data (such as depth image, Light Detection And Ranging of Laser Imaging Detection and Ranging (LIDAR) measurements) to precept MAVs' local environments. And in order to avoid collisions with obstacles, both the estimated states of MAVs and obstacles' information are then used to generate collision-free trajectories with different planning algorithms (for example graph search is used in [10], minimum-jerk is used in [12] and motion primitive is used in [13, 14]).

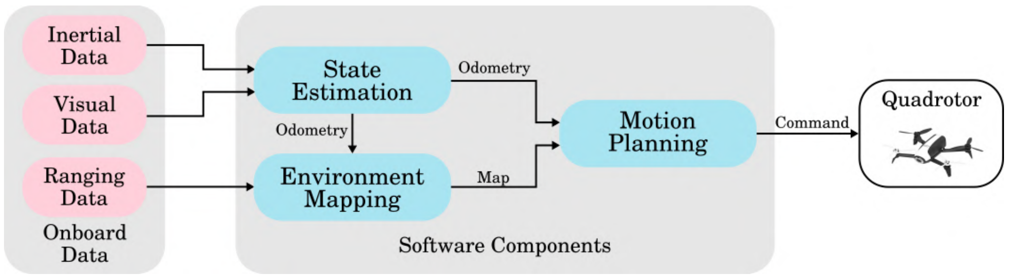


Figure 1-3: A general pipeline for autonomous navigation of MAV.

In addition, MAVs are required to navigate within dynamic environments in most of the indoor scenarios. For example, an automated drone videography system may need to fly in human environments and is required to track or avoid pedestrians. Take Autonomous Drone Race (ADR) as another example, in IROS ADR 2018, the racing track was featured with a rotating obstacle. The presentation of moving objects may be a crucial problem when MAVs are navigating in unknown environments. Because if a MAV navigates by simply following the trajectory planned at the very first beginning, it may collide with the changing environments.

These two aforementioned challenges together raise a question: is it possible to build a vision-based MAV system to navigate autonomously and safely in highly dynamic environments with onboard sensing and computing?

1-2 Relative Works

In some early approaches, collision-free navigation in unknown environments has been successfully demonstrated with different reactive heuristic methods. For example, in [15], frontal obstacles are detected by extracting expanding-size Speed-Up Robust Feature (SURF) points[16] in a sequence of monocular images. And in order to avoid collisions, once the MAV detects a frontal obstacle, it overtakes the obstacle from its side-way by simply controlling the MAV's velocity in the y-direction (while its velocity in the x-direction is constant). In another example [11], extended objects are used as the obstacles' representation instead of sparse feature points. It uses disparity images to detect vertical obstacles in camera's Field of View (FOV) and models them as two-dimensional ellipses in the x-y plane. It then avoids the extracted ellipses by selecting a set of feasible waypoints and planning the shortest path along the ellipse's edge.

Some other approaches avoid obstacles by using the motion primitive [17], which was first introduced for efficient autonomous ground vehicle navigation. The main idea of the motion primitive is evaluating a fixed number of constant-curvature arc motions and selecting the best trajectory from the pre-generated motion primitive library. The evaluation of the arc motions considers the quality of obstacle avoidance, the traveling distance toward the goal point, etc. For example, later in [18], motion primitive was adapted and used for MAV's three-dimensional navigation. This approach uses Gaussian Mixture Model (GMM) to group the three-dimensional point cloud, generated by a depth camera, into ellipsoids and selects a collision-free trajectory by calculating the collision probability of each motion.

Mellinger *et al.* [19] firstly showed that a quadrotor's dynamics are differentially flat, which means its states and control commands can be written in terms of flat outputs and their derivatives. With this property, some approaches plan smooth trajectories by finding a polynomial equation that passes through all the waypoints. For example, in [20], a sequence of waypoints is obtained by using Rapidly-exploring Random Trees (RRT)* algorithm [21] and the optimal collision-free trajectory is then generated by finding a minimum-snap polynomial trajectory passing through all the obtained waypoints. Later in [22], the author extracts a convex region within the free space and assigns interval waypoints within the convex decomposition. The optimal collision-free trajectory is then generated by fitting a Bézier curve to those extracted interval waypoints.

However, the aforementioned algorithms' optimal collision-free trajectories are optimal only under the assumption that the environment is static. When a moving obstacle is presented in a MAV's local neighborhood, the aforementioned algorithms do not take the obstacle's future positions into account.

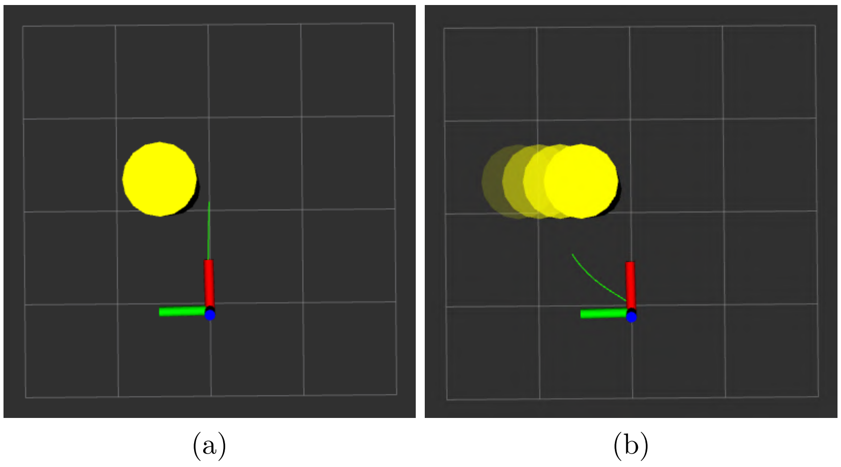

(a) (b)

Figure 1-4: A comparison between the optimal collision-free trajectory (green) generated by (a) a planner considers only obstacle's current position and (b) a planner considers both obstacle's current and future positions. The blurred obstacle is the previous position that used in the planner in order to estimate its velocity.

An example is shown in Figure 1-4(a), a point (represented by axes) is designed to navigate from the bottom to the top, while an obstacle (yellow) is moving from the left to the right. Without future re-planning, the aerial vehicle will collide with the obstacle by following the generated trajectory (green line). While in Figure 1-4(b), the vehicle is planned to overtake the obstacle from its left side without any collisions.

To tackle this problem, some researchers have moved their attention to Model Predictive Controller (MPC) since it has not only the ability of handling constraints but also the preview capability. These features of MPC make it an ideal local planner for waypoint tracking and moving obstacle avoidance. In the past, the process of MPC has to be performed offboard [23, 24], because optimal control problems are too computationally expensive to be computed with the limited onboard computation capability. But thanks to the Moore's Law [25] and the recent development of nonlinear MPC solvers (such as ACADO Toolkit [26] and FORCES Pro [27]), solving nonlinear MPC problems with an limited computational unit in real-time has become realistic and widely used in various autonomous MAV applications [3, 28, 29].

In order to generate optimal collision-free trajectories with MPC, multiple objectives and constraints of the optimal control problem are expressed mathematically, which can be categorized into two main tasks: waypoint tracking and obstacle avoidance. For the task of waypoint tracking, the planner punishes the distance between the next waypoint and the aerial vehicle over the prediction horizon. And as for the task of obstacle avoidance, the planner performs collision checking for not only the current time step but also the future predictions.

In the previous works, autonomous and safe navigation have been successfully demonstrated in static environments [30] or controlled dynamic environments with an overhead motion capture system [3, 31], while real-time onboard vision-based MAV avoidance of moving obstacles haven't been demonstrated. Thus, in this thesis work, multiple objectives and constraints are formulated with the onboard sensing data to achieve safe navigation in dynamic environments. In order to obtain the aerial vehicle's states (including the center position, velocity, and orientation), a state-of-art Visual-Inertial Odometry (VIO) algorithm is used to estimate the states of MAV using the onboard visual and inertial measurements. To gather obstacles' kinematics, an effective obstacle sensing method is implemented to extract obstacles' information (including their center positions, sizes, and their uncertainties) and estimate obstacles' linear velocities. And in order to improve the flight safety, the planner then performs collision checking by ensuring the collision probability, which is computed with the measurement and estimation uncertainties, between the MAV and each obstacle is below a specified threshold.

1-3 Objectives and Contributions

To the best of the writer's knowledge, real-time onboard vision-based MAV's avoidance of moving obstacles haven't been demonstrated. This leads to the main objective of this thesis work:

> *To design and construct an autonomous Micro Aerial Vehicle (MAV) system that can navigate autonomously and safely in dynamic environments with only onboard sensing and computing.*

The more detailed sub-objectives are listed as follows, which also reflect the contributions of this thesis work:

- Integrate an autonomous MAV system for real-time vision-based collision avoidance of moving obstacles in dynamic environments.
- Extend a fast obstacle detection method to detect, track, and predict obstacles in three-dimensional space and evaluate its performance in both static and dynamic environments.
- Adapt an optimal control problem for robust obstacle avoidance in real-world scenarios. And validate it in a variety of environments to show its effectiveness.

1-4 Thesis Outline

The structure of the following report is five-fold: in Chapter 2, the main findings and scientific contributions of this thesis work are presented in a scientific paper format. In Chapter 3, the report introduces the experimental platform used in this thesis work which includes hardware design and software architecture. In Chapter 4, state-of-art vision-based state estimation algorithms are introduced, evaluated and compared in order to select the most suitable state estimation method for the designed experimental platform. Chapter 5 introduces the obstacle sensing algorithm used in this thesis work including obstacle detection, tracking, and prediction. In Chapter 7, the proposed MAV autonomous navigation system is tested in a variety of environments. Finally, Chapter 8 concludes this report with a short summary and suggestions for future works.

Chapter 2

Scientific Paper

This chapter contains a conference paper which has been submitted to International Conference on Robotics and Automation (ICRA) 2020[1] on September 15th 2019. And a submitted video as the supplemental material of the conference paper can be accessed via https://youtu.be/nvxn7niSBCM.

[1]https://icra2020.org/

Robust Vision-based Obstacle Avoidance for Micro Aerial Vehicles in Dynamic Environments

Jiahao Lin*, Hai Zhu* and Javier Alonso-Mora

Abstract— **In this paper, we present an on-board vision-based approach for avoidance of moving obstacles in dynamic environments. Our approach relies on an efficient obstacle detection and tracking algorithm based on stereo image pairs, which provides the estimated position, velocity and size of the obstacles. Robust collision avoidance is achieved by formulating a chance-constrained model predictive controller (CC-MPC) to ensure that the collision probability between the micro aerial vehicle (MAV) and each moving obstacle is below a specified threshold. The method takes into account MAV dynamics, state estimation and obstacle sensing uncertainties. The proposed approach is implemented on a quadrotor equipped with a stereo camera and is tested in a variety of environments, showing effective on-line collision avoidance of moving obstacles.**

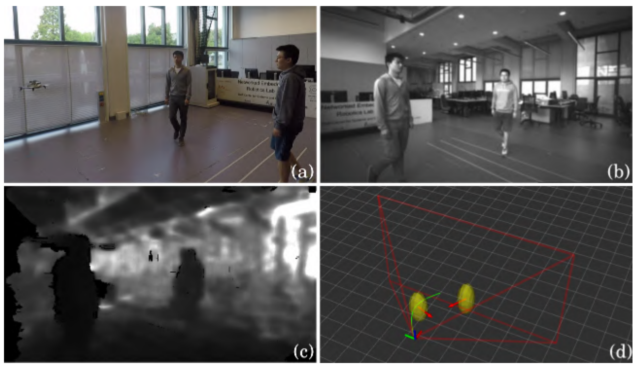


Fig. 1: Experimental results of vision-based collision avoidance in dynamic environments with two moving humans. The MAV is equipped with a stereo camera both for visual odometry and obstacle detection. (a) A snapshot of the experiment. (b) On-board grayscale image. (c) The depth image. (d) Visualization of detected obstacles and planned collision-free trajectory.

I. INTRODUCTION

Micro Aerial Vehicles (MAVs) are being deployed in a variety of application domains [1], such as search and rescue, industrial inspection, and cinematography. In particular, these applications require MAVs to safely navigate and avoid obstacles in the environments. Successful autonomous navigation of MAVs using on-board sensors has been demonstrated in static environments [2], [3] or in controlled dynamic environments with an overhead motion capture system [4], [5].

The presence of moving obstacles requires a fast and efficient obstacle detection and tracking strategy to perform obstacle avoidance in real time. Moreover, obstacle sensing and MAV state estimation uncertainties should be accounted for to achieve robust collision avoidance. In this paper, we present an on-board vision-based approach for robust navigation of MAVs in dynamic environments. Our approach builds upon, and extends, a vision-based obstacle detection and tracking algorithm [6] and a model predictive controller (MPC) [5] to generate feasible and probabilistically safe trajectories for the MAV.

A. Related Work

There has been a large amount of work in vision-based autonomous navigation for MAVs in unknown environments [7]. Some early works demonstrate autonomous navigation by abstracting simple obstacle representations from camera images and adopting reactive heuristic collision avoidance techniques. [8] maps obstacles as cylinders by detecting and tracking features of interest from images, then computes a reactive acceleration for collision avoidance. Similarly, [9] uses a monocular camera to avoid frontal obstacles by detecting relative size changes of image features and computing an avoidance velocity. However, image processing in those approaches is computationally heavy, which requires an off-board ground computer. In [10], the authors filter 3D point clouds from depth images into planes that are then used to compute the optimal collision avoidance direction with maximal open path length. In [6], the authors segment obstacles, modelled as ellipses, from a disparity map and plan a set of waypoints along the edge of obstacles using a heuristic collision checking algorithm. While the two approaches are shown to be fast, robot dynamics are not taken into account in the planner.

Recent works mainly rely on a similar pipeline where an environment map is built from image data and then used to plan collision-free motions for MAVs. In [11], the authors incrementally build a 3D global occupancy map on-board the MAV and use the VHF+ algorithm [12] for collision avoidance. Recent advances have lead to more efficient map representations than the occupancy map [13], including the ESDF map [14], the k-d tree [15], the NanoMap [16], the FIESTA map [17], etc. After building those maps, two main categories of methods are developed to plan collision-free motions. One is to use a library of pre-computed motion primitives [15] or funnels [18] and choose the best one

*J. Lin and H. Zhu contributed equally.

This work is supported by the Netherlands Organisation for Scientific Research (NWO), domain Applied Sciences, the Amsterdam Institute for Advanced Metropolitan Solutions and ONRG-NICOP-grant N62909-19-1-2027. The authors are with the Department of Cognitive Robotics, Delft University of Technology, 2628 CD, Delft, The Netherlands {h.zhu; j.alonsomora}@tudelft.nl

from the library via collision checking. The other is to construct a collision-free flight corridor based a planned path obtained from a discrete planner such as A* and JPS [19], followed by trajectory optimization to generate dynamically feasible trajectories for the MAV [20], [21], [22]. While those approaches have shown successful navigation of a MAV in a variety of environments, a common limitation of them is that they all assume the environments to be static without moving obstacles. Moreover, obstacle sensing uncertainty and MAV state estimation uncertainty are generally neglected.

B. *Contribution*

The main contribution of this paper is an integrated system for collision avoidance of moving obstacles in dynamic environments. Obstacles are detected and their position, velocity and size are estimated from stereo images and the generated U-depth maps (Section III). Chance-constraints are then formulated to account for the measured MAV state estimation and obstacle sensing uncertainty. These chance constraints are integrated in a model predictive controller to generate dynamically feasible and robust trajectories that keep the probability of collision below a specified threshold (Section IV). Finally, we demonstrate the system in real-world experiments to show its effectiveness (Section V).

II. System Overview

Given a goal point, the MAV is required to plan and execute safe collision-free trajectories to navigate towards the goal while avoiding moving obstacles in the environment, based on its sensed stereo camera images.

Fig. 2 illustrates the proposed system for solving the problem, which consists of three main components: MAV state estimation, obstacle sensing and collision-free trajectory planning. In this paper, we focus on the last two components and achieve robust collision avoidance of moving obstacles. For state estimation, we rely on a visual-inertial odometry (VIO) method [23] to obtain the MAV pose and associated uncertainty. For obstacle sensing, we model obstacles as three-dimensional ellipsoids and adopt an efficient obstacle detection and tracking approach based on depth images, to obtain the obstacle size, position, velocity and associated uncertainty. For collision-free trajectory planning, we formulate a chance-constrained model predictive controller (CC-MPC) [5], taking into account the MAV state estimation and obstacle sensing uncertainty. The CC-MPC ensures that the collision probability between the MAV and each obstacle is below a specified threshold.

III. Obstacle Detection and Tracking

In this section we describe our obstacle detection and tracking algorithm using depth images, as shown in Fig. 3. The algorithm is built on [6] where planar static obstacles are considered. We extend it to three-dimensional scenarios with moving obstacles.

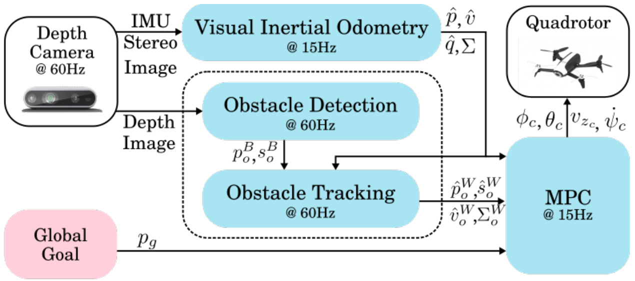


Fig. 2: The proposed system for robust vision-based obstacle avoidance in dynamic environments.

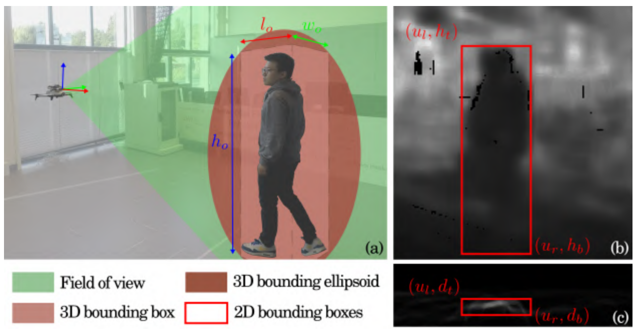


Fig. 3: Obstacle detection based on depth image and the U-depth map. (a) Schematic of the camera facing a human obstacle, which is represented as a cube with unknown constant size. (b) Obstacle height and vertical (z) position detection from the depth image. (c) Obstacle length, width and horizontal (x, y) position detection from the U-depth map.

A. *Obstacle Detection using Stereo Images*

For obstacle detection we model each obstacle as a three-dimensional cube and detect its position and size (length, width and height) based on the camera depth image. The length and width and horizontal position are firstly derived from the U-depth map (Fig. 3c), and then the height and vertical position of the obstacle is derived from the depth image directly (Fig. 3b).

1) U-depth map: An U-depth map is computed with the column depth value histograms of the original depth image. Fig. 3b shows an onboard depth image when the MAV is facing a human obstacle in a lab space and Fig. 3c is the corresponding generated U-depth map. When an obstacle is in front of the depth camera, the size of the corresponding bin in the U-depth map becomes larger. Based on this property, a bin of the histogram is considered as a point of interest if its value is larger than a threshold T_{POI}, which is defined as:

$$T_{POI} = \frac{f T_{h_o}}{d_{bin}},$$

where f is the focal length, T_{h_o} is a predefined threshold for obstacle's height in the space and d_{bin} is the corresponding depth of a bin in the histogram.

Those points of interest are then grouped with other candidate points in their neighborhood so that a bounding box can be extracted from the U-depth map, as shown in Fig. 3c.

2) Obstacle detection: We represent obstacles as three-dimensional cubes with unknown yet constant sizes. Let $\mathbf{p}_o^W = (x_o^W, y_o^W, z_o^W)^T$ be the position of the center of a cube obstacle, in which the super index W indicates the position expressed in the world frame (while B indicates in the MAV body (camera) frame) and let $\mathbf{s}_o^W = (l_o^W, w_o^W, h_o^W)^T$ be the size (length, width and height) of the cube.

Based on the bounding box found in the U-depth map [6] (see Fig. 3c), which is represented by its top-left (u_l, d_t) and bottom-right corners (u_r, d_b), we can obtain the obstacle's horizontal (x and y) position and size (length and width) in the body frame:

$$\begin{aligned} x_o^B &= d_b, & y_o^B &= \frac{(u_l + u_r)\, d_b}{2f}, \\ l_o^B &= 2\,(d_b - d_t)\,, & w_o^B &= \frac{(u_r - u_l)\, d_b}{2f}. \end{aligned} \tag{1}$$

Then we can further find a corresponding bounding box of the obstacle in the original depth image by grouping depth image points whose horizontal index are within $[u_l, u_r]$ and depth values are within $[d_t, d_t + l_o]$. Let (u_l, h_t) and (u_r, h_b) be the top-left and bottom-right corners of the bounding box. We can derive the obstacle's vertical (z) position and height in the body frame:

$$z_o^B = \frac{(h_t + h_b)\, d_b}{2f}, \quad h_o^B = \frac{(h_t - h_b)\, d_b}{f}. \tag{2}$$

For a stereo camera, the range measurements error generally increases quadratically with the measured depth [24]. In this paper, we adopt an empirically determined detection uncertainty covariance Σ_o^B for the obstacle position and $\Sigma_{o,s}^B$ for the size. Then the detected obstacle position, size and corresponding uncertainty covariance are transformed into the world frame by considering the MAV's real-time pose:

$$\begin{aligned} \mathbf{p}_o^W &= R_B^W \mathbf{p}_o^B + \mathbf{p}^W, & \Sigma_o^W &= R_B^{W^T} \Sigma_o^B R_B^W + \Sigma^W, \\ \mathbf{s}_o^W &= R_B^W \mathbf{s}_o^B, & \Sigma_{o,s}^W &= R_B^{W^T} \Sigma_{o,s}^B R_B^W, \end{aligned} \tag{3}$$

where R_B^W is the rotation matrix of the MAV's pose, $\mathbf{p}^W$ and Σ^W denote the MAV's position and uncertainty covariance respectively.

B. Obstacle Tracking and Prediction

To predict obstacle future positions within the prediction horizon, the detected obstacles in sequential frames are firstly associated by computing the probability of their means and covariances coming from the same Gaussian distribution:

$$p = p_G(\mathbf{x}_o^m \mid \hat{\mathbf{x}}_o^{m|m-1}, P_o^{m|m-1}),$$

where $p_G(\cdot)$ is the probability density function of the multivariate Gaussian distribution, $\mathbf{x}_o = (\mathbf{p}_o^W, \mathbf{s}_o^B)^T$ and $P_o = \operatorname{diag}(\Sigma_o^W, \Sigma_{o,s}^B)$ are the obstacle state (position and size) and corresponding uncertainty covariance, the super index $\cdot^m$ indicates the current frame, $\hat{\mathbf{x}}_o^{m|m-1}$ and $P_o^{m|m-1}$ are the predicted state and covariance matrix based on previous detection. If the probability p is larger than a threshold, the two objects detected are determined to be the same moving obstacle whose information is then fed to a Kalman filter.

The Kalman filter estimates the obstacle's position and its velocity and size. Denote by $\hat{\mathbf{p}}_o^k$, $\hat{\mathbf{v}}_o^k$ and $\hat{\mathbf{s}}_o^k$ the estimated obstacle position, velocity and size with uncertainty covariance Σ_o^k, $\Sigma_{o,v}^k$ and $\Sigma_{o,s}^k$ at time k. Here we omit the super index W for simplicity since in the remaining of this paper, all variables are expressed in the world frame.

For collision avoidance of moving obstacles, we predict their future positions and uncertainty covariances using a constant velocity model for obstacle movement. Hence, we have

$$\begin{aligned} \hat{\mathbf{p}}_o^{k+1} &= \hat{\mathbf{p}}_o^k + \hat{\mathbf{v}}_o^k \Delta t, & \hat{\mathbf{v}}_o^{k+1} &= \hat{\mathbf{v}}_o^k, \\ \Sigma_o^{k+1} &= \Sigma_o^k + \Sigma_{o,v}^k \Delta t^2, & \Sigma_o^{k+1} &= \Sigma_o^k. \end{aligned} \tag{4}$$

We assume the size of the obstacle is constant, i.e. $\hat{\mathbf{s}}_o^{k+1} = \hat{\mathbf{s}}_o^k$, and its uncertainty is not considered in collision avoidance.

Since polygonal obstacles are ill-posed for online constrained optimization, where smooth shapes are preferred to avoid local minima, we enlarge the detected obstacle cube using a bounding ellipsoid with semi-major axes proportional to the cube dimensions, i.e.

$$(a_o^k, b_o^k, c_o^k) = \frac{\sqrt{3}}{2}(l_o^k, w_o^k, h_o^k), \tag{5}$$

and a rotation matrix R_o^k indicating the obstacle orientation (yaw) in the world frame.

IV. Robust Collision Avoidance

In this section, we present the robust obstacle avoidance method using chance constrained model predictive control (CC-MPC). The method is based on [5] which is used for collision avoidance in a controlled environment with an overhead motion capture system. We extend it to an on-board vision based system, by furthering considering the camera's limited filed of view constraints and yaw control of the MAV.

A. Model Predictive Controller

To formulate the MPC, we first consider the MAV's dynamics model, described by a stochastic nonlinear discrete-time equation,

$$\mathbf{x}^{k+1} = \mathbf{f}(\mathbf{x}^k, \mathbf{u}^k) + \omega^k, \quad \mathbf{x}^0 \sim \mathcal{N}(\hat{\mathbf{x}}^0, \Gamma^0), \tag{6}$$

where $\mathbf{x}^k = [\mathbf{p}^k, \mathbf{v}^k, \phi^k, \theta^k, \psi^k]^T \in \mathcal{X}$ denotes the state of the MAV (position, velocity and orienting) and $\mathbf{u}^k \in \mathcal{U}$ the control input at time step k. $\mathcal{X}$ and $\mathcal{U}$ are the admissible state and control space respectively. The initial state $\mathbf{x}^0$ is obtained from a state estimator with mean $\hat{\mathbf{x}}^0$ and covariance Γ^0. $\mathbf{f}$ denotes the nonlinear dynamics. We consider the MAV's motion disturbances as Gaussian process noise $\omega^k \sim \mathcal{N}(0, W^k)$. See [5] for details of the dynamics model.

At every time step, for obstacle avoidance, we formulate and solve online a receding horizon constrained optimization problem with N time steps and planning horizon $\tau = N\Delta t$, where Δt is the sampling time, as follows,

$$\begin{aligned}
\min_{\hat{\mathbf{x}}^{1:N},\mathbf{u}^{0:N-1}} \quad & \sum_{k=0}^{N-1} J^k(\hat{\mathbf{x}}^k, \mathbf{u}^k) + J^N(\hat{\mathbf{x}}^N) && \text{(7a)}\\
\text{s.t.} \quad & \hat{\mathbf{x}}^0 = \hat{\mathbf{x}}(t_0), && \text{(7b)}\\
& \hat{\mathbf{x}}^k = \mathbf{f}(\hat{\mathbf{x}}^{k-1}, \mathbf{u}^{k-1}), && \text{(7c)}\\
& \mathbf{G}(\hat{\mathbf{x}}^k, \Gamma^k) \leq 0, && \text{(7d)}\\
& \mathbf{u}^{k-1} \in \mathcal{U}, && \text{(7e)}\\
& \hat{\mathbf{x}}^k \in \mathcal{X}, && \text{(7f)}\\
& \forall k \in \{1, \ldots, N\},
\end{aligned}$$

where J^k denotes the cost term at time k and J^N denotes the terminal cost, $\mathbf{G}$ is a function representing the state constraints, which are described in detail in the following. Γ^k is the MAV state uncertainty covariance at time k. We further denote by Σ^k the 3×3 covariance matrix of the MAV position $\mathbf{p}^k$, extracted from Γ^k.

B. Cost Function

We now describe the components of the cost function presented in Eq. (7a).

1) Goal navigation: Let $\mathbf{p}^g$ be the given goal position of the MAV. We minimize the displacement between its terminal position in the planning horizon and its goal. To this end, we define the terminal cost term,

$$J^N(\hat{\mathbf{x}}^N) = \left\|\hat{\mathbf{p}}^N - \mathbf{p}^g\right\|_{\mathbf{Q}_g}, \tag{8}$$

where $\mathbf{Q}_g$ is a tuning weight coefficient.

2) Control input cost: The second cost term is to minimize the MAV control inputs, designed as a stage cost,

$$J_u^k(\mathbf{u}^k) = \left\|\mathbf{u}^k\right\|_{\mathbf{Q}_u}, \tag{9}$$

where $\mathbf{Q}_u$ is a tuning weight coefficient.

3) Collision cost: To improve flight safety, we also introduce an obstacle potential field cost based on the sigmoid function. Denote by $d_o^k = \left\|\hat{\mathbf{p}}^k - \hat{\mathbf{p}}_o^k\right\|$ the nominal distance between the MAV and obstacle o. Then at time stage k, the potential field cost corresponding to obstacle o is

$$J_o^k(\hat{\mathbf{p}}^k) = \frac{\mathrm{Q}_o}{1 + \exp\left(\lambda_o(d_o^k - r_o)\right)}, \tag{10}$$

where Q_o is a tuning weight coefficient, λ is a parameter defining the smoothness of the cost function and r_o is a tuning threshold distance between the MAV and the obstacle where the collision cost is $Q_o/2$. The reason to use a sigmoid function is to achieve a smooth and bounded collision cost function.

4) MAV yaw control: Since the MAV has a limited field of view, it is generally desirable to make the camera axis, hence the yaw orientation aligned with the direction of motion. Instead of employing a velocity tracking yaw control method as in [25] which may generate infeasible yaw trajectories, we design a cost function to minimize the deviation between the MAV's yaw and motion direction,

$$J_\psi^k(\psi^k) = Q_\psi(\psi^k - \bar{\psi}^k)^2, \tag{11}$$

where Q_ψ is a tuning weight coefficient, $\bar{\psi}^k = \arctan\frac{\hat{v}_y^k}{\hat{v}_x^k}$ indicates the MAV's motion direction angle. To reduce computation time, we compute $\bar{\psi}^k$ based on the MAV's last-loop planned trajectory.

Finally, the overall stage cost of the formulated MPC is

$$J^k(\hat{\mathbf{p}}^k, \mathbf{u}^k) = J_u^k(\mathbf{u}^k) + J_o^k(\hat{\mathbf{p}}^k) + J_\psi^k(\psi^k). \tag{12}$$

C. Constraints

1) Collision chance constraints: For the obstacle, modelled as an ellipsoid, at position $\mathbf{p}_o^k$ with semi-principal axes (a_o^k, b_o^k, c_o^k), the MAV at position $\mathbf{p}^k$ with radius r is considered to be in collision with it if

$$C_o^k : (\mathbf{p}^k - \mathbf{p}_o^k)^T \Omega_o^k (\mathbf{p}^k - \mathbf{p}_o^k) \leq 1$$

where $\Omega_o^k = R_o^{k,T} \operatorname{diag}(\frac{1}{(a_o^k+r)^2}, \frac{1}{(b_o^k+r)^2}, \frac{1}{(c_o^k+r)^2}) R_o^k$.

In this paper, we take into account the MAV state estimation uncertainty and obstacle sensing uncertainty. Hence, the collision avoidance constraints would be satisfied in a probabilistic manner, which are formulated as chance constraints:

$$\Pr(C_o^k) \leq \delta,\ \forall k = 1, \ldots, N, \tag{13}$$

where δ is the probability threshold for robot-obstacle collision.

By assuming $\mathbf{p}^k$ and $\mathbf{p}_o^k$ are according to Gaussian distributions (obtained from our estimators), i.e. $\mathbf{p}^k \sim \mathcal{N}(\hat{\mathbf{p}}^k, \Sigma^k)$ and $\mathbf{p}_o^k \sim \mathcal{N}(\hat{\mathbf{p}}_o^k, \Sigma_o^k)$, the chance constraint in Eq. (13) can be transformed into a deterministic constraint with their mean and covariance as follows [5]

$$\begin{aligned}
\mathbf{n}_o^{k^T} \Omega_o^{k^{\frac{1}{2}}} (\hat{\mathbf{p}}^k - \hat{\mathbf{p}}_o^k) - 1 \geq \operatorname{erf}^{-1}(1 - 2\delta)& \\
\cdot \sqrt{2\mathbf{n}_o^{k^T} \Omega_o^{k^{\frac{1}{2}}} (\Sigma^k + \Sigma_o^k) \Omega_o^{k^{\frac{1}{2}}} \mathbf{n}_o^k}&
\end{aligned} \tag{14}$$

where $\mathbf{n}_o^k = (\hat{\mathbf{p}}^k - \hat{\mathbf{p}}_o^k/) \left\|\hat{\mathbf{p}}^k - \hat{\mathbf{p}}_o^k\right\|$, $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ is the standard error function for normal distribution.

2) FOV Constraints: To ensure flight safety, the MAV planned trajectory should be within its current limited field of view (FOV) and limited depth sensing range. Given the MAV's current pose, its three-dimensional FOV with limited depth sensing range can be described by an intersection of five half-spaces,

$$FOV^k := \{\mathbf{p} \mid \mathbf{n}_j^k \mathbf{p} \leq m_j^k\},\ j = 1, \ldots, 5, \tag{15}$$

where $\mathbf{n}_j^k$ and m_j^k are parameters of the half-spaces. Hence, the FOV constraints are formulated as

$$\mathbf{p}^k \in FOV^k,\ \forall k = 1, \ldots, N. \tag{16}$$

D. MAV State Uncertainty Propagation

Evaluating the collision chance constraints in Eq. (14) requires calculating the MAV state, in particular, position uncertainty covariance at each time step. High-precision uncertainty propagation for nonlinear systems, as in Eq. (6) could be very computationally intensive [26]. In this paper, to achieve fast real-time uncertainty propagation, we approximately propagate the MAV state uncertainty using an Extended Kalman Filter (EKF) based update, i.e.

$$\Gamma^{k+1} = F^k \Gamma^k F^{k^T} + W^k, \tag{17}$$

where W^k is the process noise accounting for motion disturbances, $F^k = \frac{\partial \mathbf{f}^k}{\partial \mathbf{x}}|_{\hat{\mathbf{x}}^k, \mathbf{u}^k}$ is the state transition matrix of the MAV. Then the position uncertainty covariance Σ^k can be extracted from Γ^k. Note that in the above equation, the computation of F^k correlates the robot state and control inputs, which will introduce additional variables in the optimization problem Eq. (7) and increases the computation time greatly. To this end, we propagate the MAV state uncertainty based on its last-loop trajectory and control inputs before solving this-loop optimization problem.

V. Results

In this section, we describe our implementation of the proposed approach and evaluate it in real-world experiments. A video showing the flight test results accompanies this paper.

A. Implementation and Hardware

Our experimental platform is the Parrot Bebop 2 quadrotor[1] mounted with an NVIDIA Jetson TX2 Module[2] and an Intel RealSense Depth Camera D435i[3], as shown in Fig. 4. The Parrot Bebop 2 allows for executing control commands sent via ROS[4]. The D435i camera is dually used for visual-inertial odometry and depth image sensing, which has a $87° \times 58°$ FOV and 5 m depth sensing range. The TX2 is used to perform all on-board computation and is connected with the Bebop 2 via WiFi.

We use a filter-based stereo visual-inertial odometry algorithm, the S-MSCKF [23], for state estimation of the MAV, which runs at 15 Hz. The camera depth images are received at 60 Hz and the obstacle detection and tracking is running at frame rate. We rely on the ACADO toolkit [27] to generate a fast C solver for our MPC, in which a sampling time of 60 ms is used and the prediction horizon is set to 1.5 s. The radius of the MAV is set as 0.4m. The two closest detected obstacles are fed to the MPC for collision avoidance. The collision probability threshold is set as $\delta = 0.03$. In order to obtain some quantitative results, in the lab scenarios we use an external motion capture system (OptiTrack) to measure the position of the MAV and moving obstacles, which is only used as ground true data.

B. Obstacle Detection and Tracking Performance

We first evaluate the obstacle detection and tracking performance for moving obstacles. Fig. 1(a) shows a lab scene with two walking human obstacles. We put the camera at the origin of the world frame and recorded a dataset of which the two humans were walking around at a speed of approximately 1.2 m/s. Position and velocity of the two human obstacles are obtained using our obstacle detection and tracking algorithm. Table I shows the average position and velocity estimation errors of the two moving obstacles comparing with ground truth measurements. It can be observed that the average position estimation error is around 0.3 m and that of velocity can be up to 0.5 m/s, which indicates the obstacle sensing uncertainty should be taken into account when planning robust collision-free trajectories for the MAV. In practice, the obstacle's velocity estimation may be very noisy and has a very large uncertainty covariance. In this case we bound the $\Sigma^k_{o,v}$ in Eq. (4) when predicting the obstacle's future positions and corresponding uncertainty covariances.

[1]https://www.parrot.com/us/drones/parrot-bebop-2

[2]https://developer.nvidia.com/embedded/jetson-tx2

[3]https://www.intelrealsense.com/depth-camera-d435i/

[4]https://bebop-autonomy.readthedocs.io

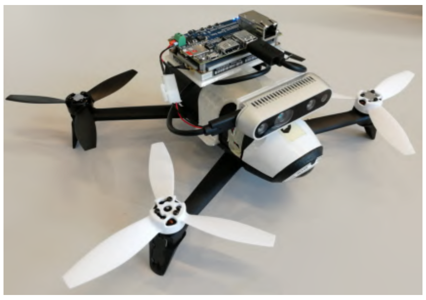


Fig. 4: MAV used in the experiments. It is equipped with an NVIDIA Jetson TX2 Module for all on-board computation, an Intel RealSense Depth Camera D435i dually for visual-inertial odometry and depth image sensing.

TABLE I: Detection and tracking errors of moving obstacles.

Moving obstacle	**Average estimation error**	
	Position (m)	**Velocity (m/s)**
No. 1	0.28	0.47
No. 2	0.25	0.41

C. Obstacle Avoidance in Dynamic Environments

We tested the system in a variety of flight tests. The results of two typical scenarios are particularly presented here.

1) Scenario 1 (Flying in a confined lab space): The MAV is required to navigate from a start point to an end point while avoiding two walking humans. Fig. 5 shows a series of snapshots and the MAV on-board camera grayscale images taken during the experiment. In this scenario, we performed the experiment five times. Fig. 6a shows the measured distance between the MAV and the two moving human obstacles over time in the five experiments. The distance is computed, based on ground truth measurements, as the closest distance between the MAV's position and the obstacle ellipsoid's surface (with semi-major axis $(0.4, 0.4, 0.9)m$). In Fig. 6b, we cumulate all the distance data. It can be observed that in all instances a minimum safe separation of 0.4 m was achieved and therefore collision with the humans were avoided. A maximal speed of around 1.6 m/s of the MAV was observed in this experiment.

The boxplots of the on-board runtimes in this scenario is shown in Fig. 6c. For the runtimes of the obstacle detection and tracking, the 75th percentile is always below 8 ms, which is fast enough to be run at frame rate (60 Hz). For the

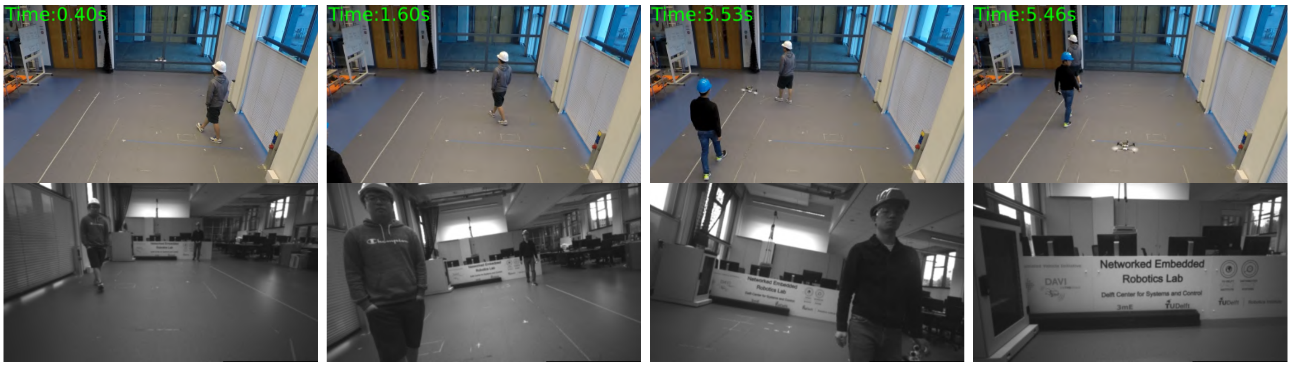
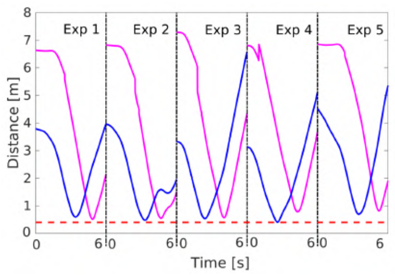
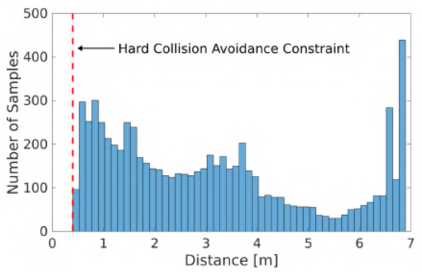


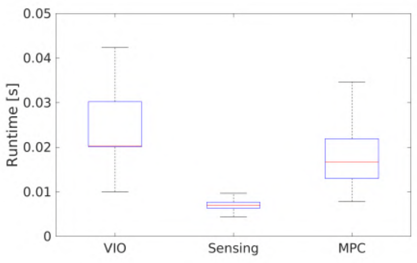
Fig. 5: A sequence of images during the experiment Scenario 1. The MAV is required to fly from a start point to an end goal while avoiding two walking humans. Top: Snapshots of the experiment. Bottom: On-board camera grayscale images.



(a) Distance to obstacles over time



(b) Histogram of distance



(c) MAV on-board runtimes

Fig. 6: Quantitative results of the experiment Scenario 1. (a) Distance between the MAV and the two moving obstacles (magenta and blue) over time during 5 experiments. (b) Histogram of all the distance data. (c) On-board runtimes of the MAV state estimation (VIO), obstacle detection and tracking, and collision-free trajectory optimization (MPC).

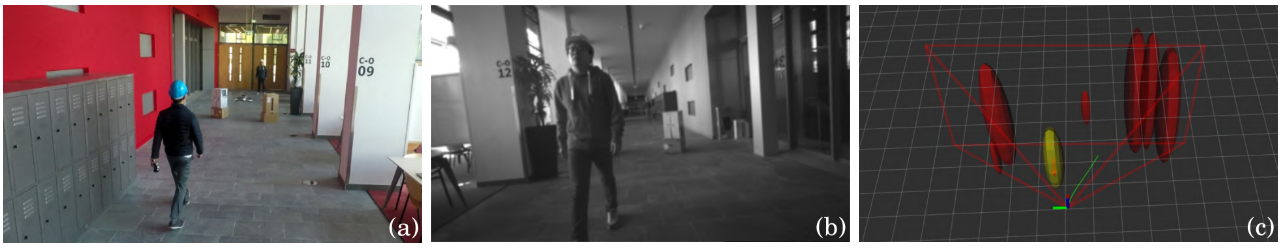


Fig. 7: Results of the experiment Scenario 2. The MAV is flying in a corridor while avoiding static and moving obstacles. (a) A snapshot during the experiment. (b) An on-board grayscale image captured in the experiment and (c) visualization of the corresponding obstacle detection and trajectory planning results.

runtimes of the MPC framework, the 75th percentile is always below 22 ms, indicating the framework can be run efficiently in real time.

2) Scenario 2 (Flying in a long corridor): The MAV is flying in a long narrow corridor where there are both static and moving obstacles. Fig. 7 shows a snapshot taken during the experiment. A maximum speed of around 2.4 m/s was achieved by the MAV in the experiment. Detailed results of the experiment can be found in the video accompanying this paper.

VI. Conclusion

We presented an on-board vision-based obstacle avoidance approach for MAVs navigating in dynamic environments. Flight test results in a variety of environments with moving obstacles demonstrated the effectiveness of the proposed approach. We adopted a fast three-dimensional obstacle detection and tracking algorithm based on stereo depth images which can run at a frame rate of 60 Hz. We took into account the obstacle sensing uncertainty by using a chance constrained model predictive controller (CC-MPC) to generate robust local collision-free trajectories for the MAV. We implemented the approach on a computational power-limited quadrotor platform, where the obstacle detection and tracking has a mean computation time of 8 ms and that of the MPC is 16 ms. In real-world indoor experiments, the MAV is shown to be able to avoid walking human obstacles at a maximum speed of 2.4 m/s.

References

[1] S.-J. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar, "A survey on aerial swarm robotics," *IEEE Transactions on Robotics*, vol. 34, pp. 837–855, 2018.

[2] H. Oleynikova, Z. Taylor, A. Millane, R. Siegwart, and J. Nieto, "A complete system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments," 2018.

[3] L. Campos-Macías, R. Aldana-López, R. de la Guardia, J. I. Parra-Vilchis, and D. Gómez-Gutiérrez, "Autonomous navigation of mavs in unknown cluttered environments," 2019.

[4] T. Nageli, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges, "Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization," *IEEE Robotics and Automation Letters*, vol. 2, pp. 1696–1703, 2017.

[5] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for mavs in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[6] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for MAV flight," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 50–56.

[7] S. Tang and V. Kumar, "Autonomous flight," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 29–52, 2018.

[8] S. Ahrens, D. Levine, G. Andrews, and J. How, "Vision-based guidance and control of a hovering vehicle in unknown, GPS-denied environments," in *2009 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2009, pp. 2643–2648.

[9] T. Mori and S. Scherer, "First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 53. IEEE, 2013, pp. 1750–1757.

[10] J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 1697–1702.

[11] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-based autonomous mapping and exploration using a quadrotor MAV," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 4557–4564.

[12] I. Ulrich and J. Borenstein, "VFH+: reliable obstacle avoidance for fast mobile robots," in *1998 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1998, pp. 1572–1577.

[13] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, pp. 189–206, 2013.

[14] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: incremental 3d euclidean signed distance fields for on-board mav planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1366–1373.

[15] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5759–5765.

[16] P. R. Florence, J. Carter, J. Ware, and R. Tedrake, "NanoMap: fast, uncertainty-aware proximity queries with lazy search over local 3d data," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7631–7638.

[17] L. Han, F. Gao, B. Zhou, and S. Shen, "FIESTA: fast incremental euclidean distance fields for online motion planning of aerial robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[18] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *International Journal of Robotics Research*, vol. 36, pp. 947–982, 2017.

[19] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, pp. 1688–1695, 2017.

[20] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*, 2016, vol. 114, pp. 649–666.

[21] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 344–351.

[22] J. Tordesillas, B. T. Lopez, and J. P. How, "FaSTraP: fast and safe trajectory planner for flights in unknown environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[23] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, pp. 965–972, 2017.

[24] K. Schauwecker and A. Zell, "Robust and efficient volumetric occupancy mapping with an application to stereo vision," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6102–6107.

[25] T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2135–2142.

[26] Y. Luo and Z. Yang, "A review of uncertainty propagation in orbital mechanics," *Progress in Aerospace Sciences*, vol. 89, pp. 23–39, 2017.

[27] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit-An open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, pp. 298–312, 2011.

Chapter 3

System Design

In this chapter, the design of the experimental platform is presented. Specifically, considerations of the hardware selection and the software architecture design are discussed.

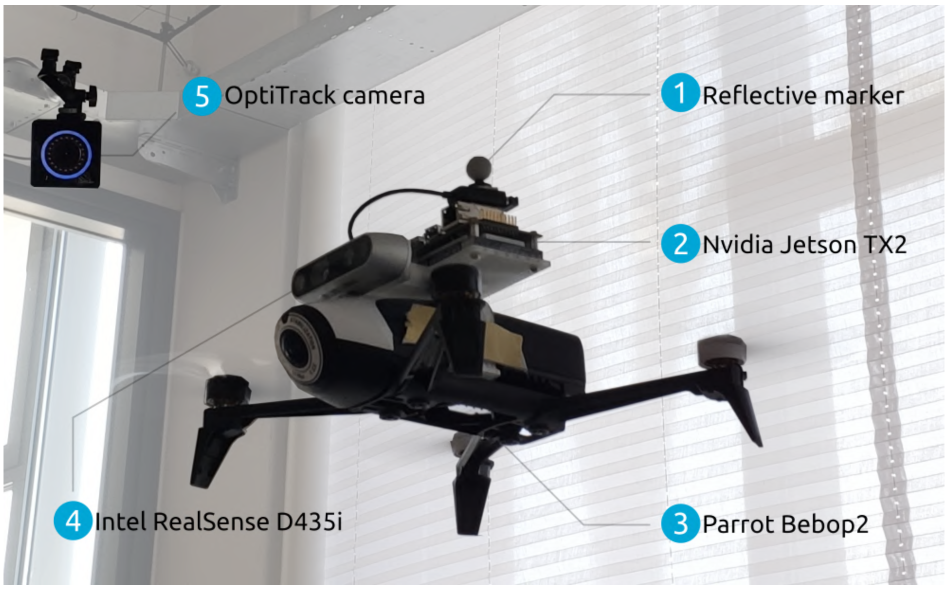


Figure 3-1: A Parrot Bebop2[1] is equipped with a forward-looking Intel RealSense depth camera D435i[2] and an onboard computational unit Nvidia Jetson TX2[3]. The OptiTrack camera[4] and the equipped reflective marker are used for ground truth data collection.

[1]https://www.parrot.com/us/drones/parrot-bebop-2
[2]https://www.intelrealsense.com/depth-camera-d435i/
[3]https://developer.nvidia.com/embedded/jetson-tx2
[4]https://optitrack.com/

3-1 Platform Design

The aim of the system design is to construct a small-size and light-weight micro aerial vehicle which can navigate autonomously with only onboard sensing and computing. To accomplish this, the autonomous aerial vehicle is desired to be equipped with onboard sensors whose range is not less than 5 m. Besides, the system should carry a powerful computational unit, which is able to perform all the software components including obstacle sensing, state estimation, and trajectory planning in real-time. These considerations imply that the search for such an aerial vehicle system is a weight-power trade-off, which is discussed in the following.

3-1-1 Sensing Module

In order to precept obstacles in the aerial vehicle's local neighborhood, the onboard sensor is desired to have the capability to measure the distance. In some approaches [7, 32], LIDAR is equipped on the aerial vehicles for localization, mapping and obstacle detection. Although the LIDAR measurements are accurate and available on long-distance, 3D LIDAR sensors are too heavy (ranging from 590 g[1] to 1300 g[2]) to be carried by a micro aerial vehicle. Instead, 2D LIDAR are used in the aforementioned approaches. The problem of planar LIDAR is that it only provides 2D data and it is not suitable for the navigation in 3D space. Alternatively, some other approaches [10, 11, 13] use stereo cameras (or depth cameras), which is particularly useful for providing rich 3D information. Besides, stereo cameras can not only measure the distance of obstacles but also provide fast and accurate state estimation [33, 6, 34, 35, 36]. These considerations determine the choice that depth camera is a suitable sensor for this platform. And in order to figure out the most suitable onboard depth camera for the autonomous aerial vehicle, a list of different commercially available depth (or stereo) cameras are shown in Table 3-1.

Table 3-1: Comparison between various commercially available depth (or stereo) cameras.

Camera	Weight (g)	Power (W)	Max Range (m)	Depth Image Processor	IMU
DUO MLX[3]	12.5	2.5	-	✗	✓
ZED Mini[4]	62.9	1.9	12.0	✗	✓
Intel RealSense D435i	72.0	3.5	10.0	✓	✓
MYNT Depth[5]	184.0	3.5	18.0	✓	✓
Orbbec Astra[6]	300.0	2.4	8.0	✓	✗

Note that DUO MLX is a stereo camera, the max range of depth measurement is not provided officially. Nevertheless, the length of its baseline is only 30 mm, which suggests that the depth measurement at long distance will be inaccurate.

[1] https://velodynelidar.com/vlp-16-lite.html
[2] https://velodynelidar.com/hdl-32e.html
[3] https://duo3d.com/product/duo-minilx-lv1
[4] https://www.stereolabs.com/zed-mini/
[5] http://www.myntai.com/en/mynteye/depth
[6] https://orbbec3d.com/product-astra-pro/

The consideration of the camera selection includes the camera's weight and power consumption, these two factors will affect the stability of flight maneuvers and flight endurance. Thus, a camera with lighter weight and lower power consumption is more preferable. In addition, the embedded depth image processor should be also highlighted. With such a processor, the stereo algorithm can be performed on the depth camera instead of the onboard computational unit, which will save a large amount of memory and computational efforts. Also, the integrated IMU sensor is a valuable plus, as by fusing the inertial measurement with the visual information, the accuracy of state estimation will be improved without doubts.

From the above survey of the commercially available cameras, Intel RealSense D435i is considered the most suitable choice among the proposed camera since it is relative light-weight and it integrates an embedded depth image processor and an IMU. Figure 3-2 shows the available visual information from Intel RealSense D435i. The visual information is captured at a rate of up to 90 Hz, which consists of a pair of global-shutter, monochrome images, a rolling-shutter RGB image ,and a depth image. And the angular rates and acceleration from the integrated IMU are measured at a rate of 250 Hz.

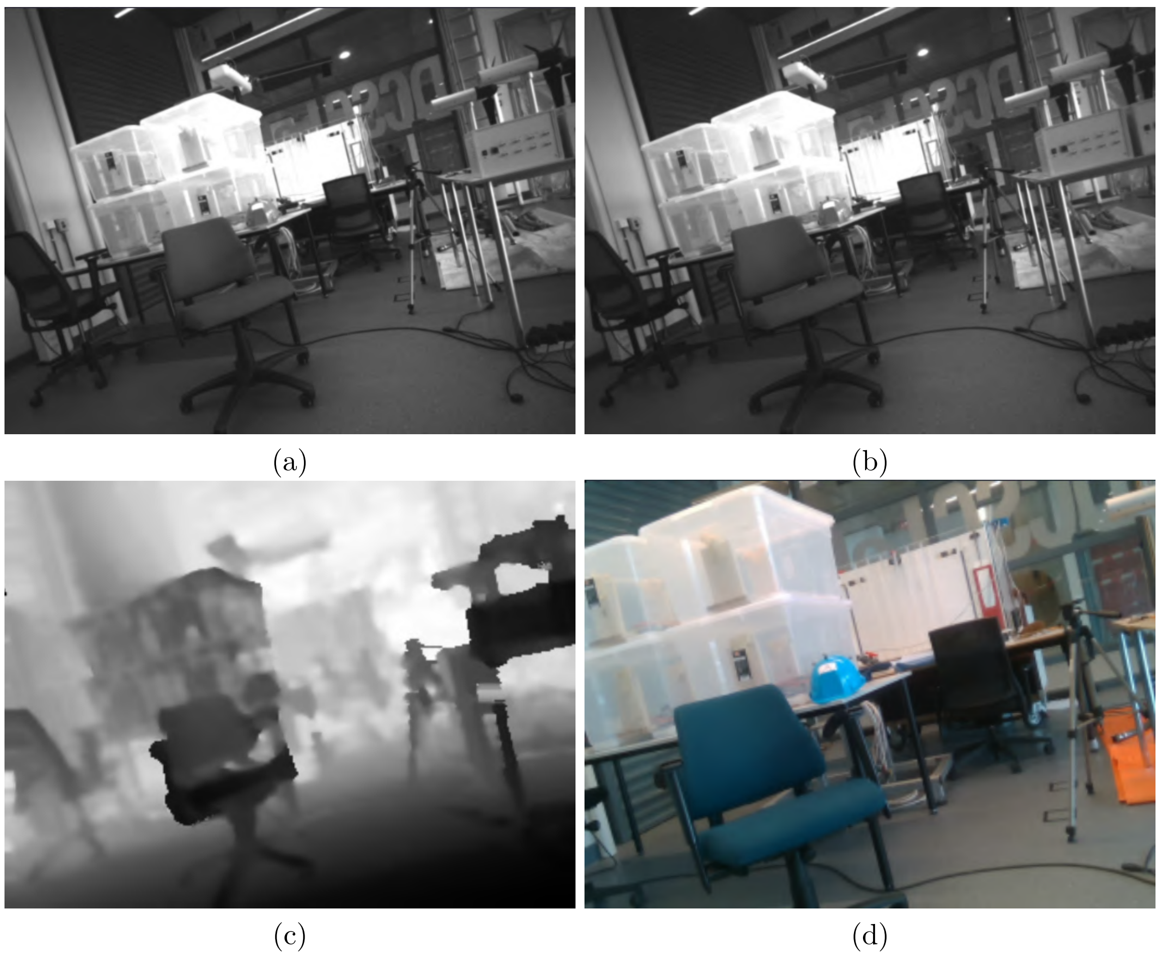


Figure 3-2: The onboard visual information provided by Intel RealSense D435i: (a) left and (b) right image of the stereo camera, (c) depth image generated by the embedded depth image processor and (d) RGB image.

3-1-2 Computing Module

In order to perform all the computational components onboard, Nvidia Jetson TX2 is selected as the onboard computational unit. Since this single board computer is credit card sized and its weight is 126 g, which is possible to be carried by a small size aerial vehicle. Besides, compared with other single board computers, it is computationally powerful and has a larger memory (as can be seen in Table 3-2). Especially, its GPU outperforms the others', from which the onboard image process will benefit. With these features, it is sufficiently powerful to perform optimal control, vision-based state estimation and obstacle sensing in real-time. This will certainly improve the aerial vehicle's performance with aggressive maneuvers in dynamic environments.

Table 3-2: Comparison between various commercially available single board computers.

Computer	Weight (g)	Power (W)	CPU	RAM (GB)	GPU Performance (GFLOPS)
RaspberryPi 3B[1]	45.0	12.5	4×ARM Cortex-A53, quad-core, 64-bit.	1	6.2
ODROID XU4[2]	60.0	20.0	4×ARM Cortex-A15, 4×ARM Cortex-A7, octa-core, 32-bit.	2	122
UP Board[3]	80.0	20.0	Intel Atom x5-Z8350, quad-core, 64-bit.	4	294
Jetson TX2	126.0	20.0	4×ARM Cortex-A57, NVIDIA Denver2, hexa-core, 64-bit.	8	750
Jetson Nano[4]	136.0	20.0	4×ARM Cortex-A57, quad-core, 64-bit.	4	256

Note that the weight of Jetson TX2 consists of the weight of a Jetson TX2 core module (85 g) and an Orbitty Carrier[5] (41 g).

In addition to the computationally capacity and memory, Nvidia Jetson TX2 also integrates a wireless module, which supports dual-band WIFI. By connecting it with an antenna (an extremely light-weight WIFI antenna[6] is used), it can then provide a stable wireless communication between the ground control station and itself. This is a valuable plus because having a ground control station enables the operator to monitor, visualize and control the aerial vehicle during development and experiment. Also, the USB 3.0 port on the Nvidia Jetson TX2 ensures the reception of massive visual information generated by the depth camera at a high frequency.

[1] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[2] https://wiki.odroid.com/odroid-xu4/odroid-xu4
[3] https://up-board.org/up/specifications/
[4] https://developer.nvidia.com/embedded/jetson-nano-developer-kit
[5] http://connecttech.com/product/orbitty-carrier-for-nvidia-jetson-tx2-tx1/
[6] https://www.delock.com/produkte/G_88984/merkmale.html

3-1-3 Drone Kit

Parrot Bebop2 is selected as the drone kit, as it is widely used among academia [3, 8, 29, 37, 38] and also because the various existing Software Development Kit (SDK)s (such as bebop_autonomy[1] and Paparazzi UAV[2]) which are well supported by their corresponding mature development communities. With these SDKs, the quadrotor can be easily controlled by giving it high-level control commands (for example pitch, roll, yaw rate, and vertical velocity). In this work, bebop_autonomy is used as the driver since it provides a Robot Operating System (ROS) wrapper which can be immediately used for the prototype construction.

From the aspect of Parrot Bebop2's power supply[3], assuming the power-to-weight ratio is constant, the flight time T'_{BAT} can be approximated by the following calculations.

Known the battery's rated capacity C_{BAT} is 3350 mAh, rated voltage V_{BAT} is 11.4 V and the claimed fly time without payload T_{BAT} is up to 30 minutes. The power-to-weight ratio ϕ_{BAT} is:

$$\begin{aligned}\phi_{BAT} &= \frac{C_{BAT}V_{BAT}}{T_{BAT}W_{drone}} \\ &= \frac{3.35\,Ah \times 11.4\,V}{0.5\,h \times 0.5\,kg} \\ &= 152.76\,W/kg\end{aligned} \tag{3-1}$$

where W_{drone} is the weight of Parrot Bebop2. After adding $W_{load} = 300\,g$ extra payload to the quadrotor, the total current draw will be:

$$\begin{aligned}I_{total} &= \frac{\phi_{BAT}(W_{drone} + W_{load})}{V_{BAT}} + I_{RealSense} + I_{Jetson} \\ &= \frac{152.76\,W/kg \times (0.5\,kg + 0.3\,kg)}{11.4\,V} + 0.7\,A + 2\,A \\ &= 13.42\,A\end{aligned} \tag{3-2}$$

The flight time of the experimental platform can then be calculated as:

$$T'_{BAT} = \frac{C_{BAT}}{I_{total}} \approx 0.25\,h \tag{3-3}$$

As for Parrot Bebop2's payload capacity, the selected drone kit is small-size (frame size is 290 mm) and light-weight (500 g) while its payload-capacity is sufficiently large to carry the extra payload. Although its payload capacity is not officially published, this is reflected by the fact that the 13.42 A estimated current draw is lower than the 16.75 A rated discharge current.

In addition to the above estimation, the flight time and the payload capacity are validated with several flight tests, which illustrates that Parrot Bebop2 is sufficiently powerful to carry all the onboard sensing and computing modules with a flight time about 12 minutes.

[1] https://bebop-autonomy.readthedocs.io
[2] https://wiki.paparazziuav.org
[3] https://www.parrot.com/us/spareparts/drones/power-battery-bebop-2-power

3-1-4 Mount

To attach all the selected sensing and computing components to the drone kit, a mount is designed and manufactured. The design has three main considerations. Firstly, the weight center of the extra payload should be well horizontally aligned with the weight center of the drone kit for the stable maneuver. Secondly, the propeller is not in the camera's Field of View (FOV) in order to achieve full visibility. Finally, the mount should be firmly attached to the drone kit. Figure 3-3 shows the mount design and gives assembly instructions of the experimental platform.

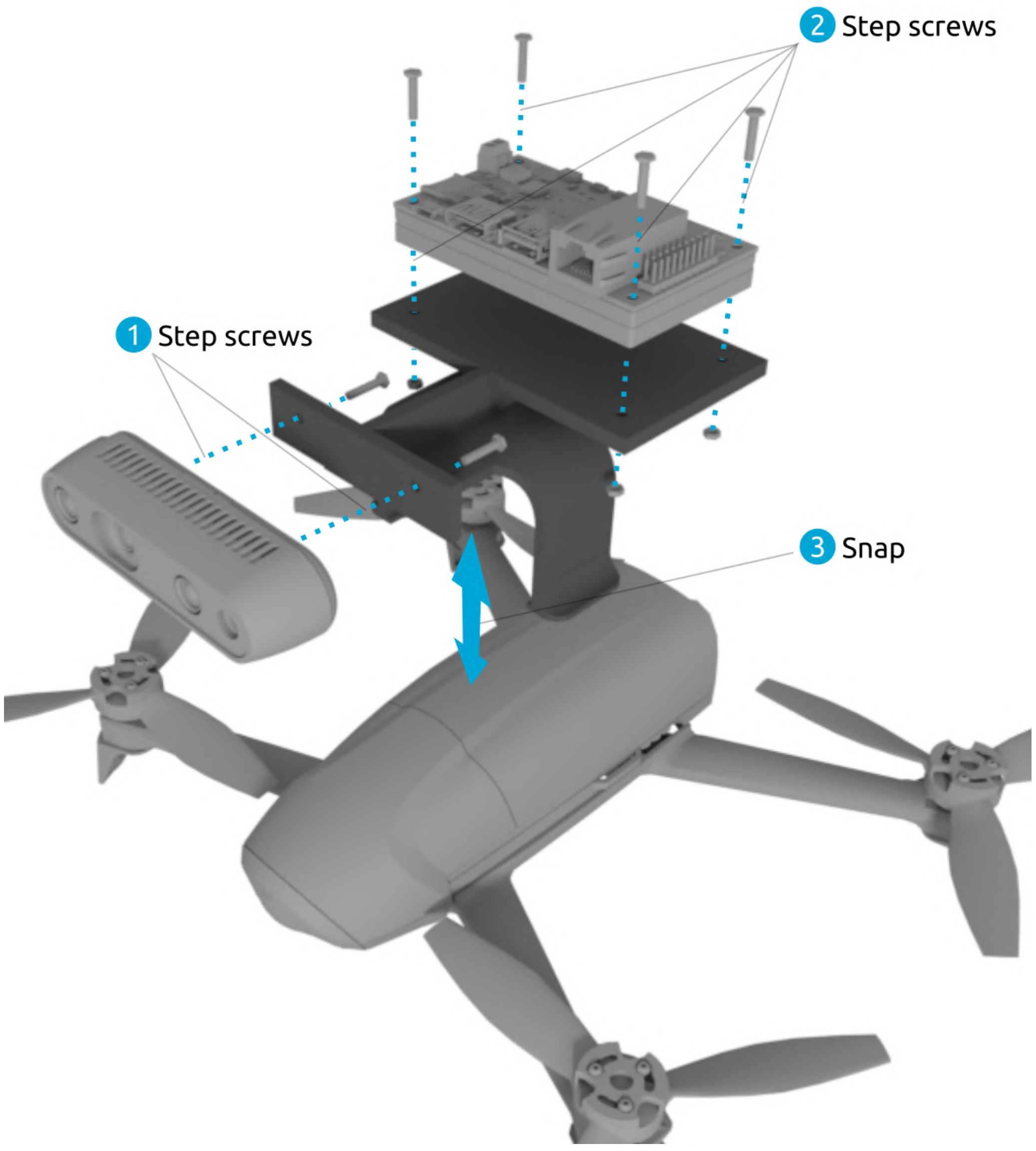


Figure 3-3: Assembly instructions of the experimental platform.

3-2 Software Architecture

As introduced in section 3-1, the autonomous MAV system consists of a depth camera (which provides both the visual and inertial information), a computational unit, and a drone kit. Besides, a ground control station is used to monitor, visualize and operate the aerial vehicle during the flight. In order to run the big system properly, these modules should be connected with each other. Figure 3-4 shows the block diagram of the system, which also demonstrates the communications among different hardware modules and software components.

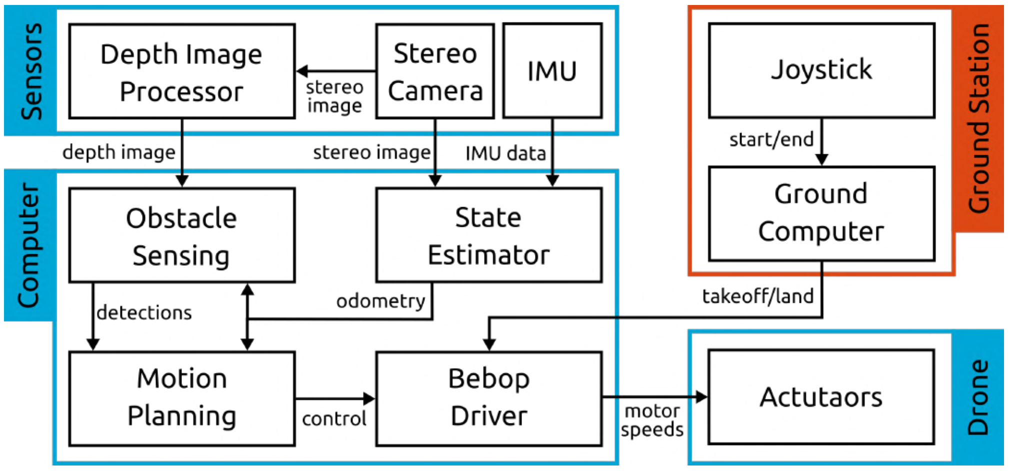


Figure 3-4: A block diagram of the whole system. The blocks are marked by different colors for the discrimination of onboard (blue) and offboard (orange) modules.

As can be seen from the block diagram, the overall system has three basic software components, namely state estimation, obstacle sensing, and trajectory planning. And in order to integrate these individual components into a unified system, ROS is used as the framework of the software architecture. With ROS, each software component can be run individually as nodes. And each node can receive and transmit data with other nodes in the same environment. This property is certainly developer-friendly since it allows distributed building, testing and debugging. Besides, ROS is popular and widely used in academia and industry. Under a dozen years of development, there are a huge amount of libraries and tools are available in ROS ecosystem and can be easily adapted for a new project. For example in Figure 3-4, several software components are adapted from developed tools, such as the ROS wrapper of Intel RealSense D435i[1], joystick driver[2], and bebop driver (bebop_autonomy). Some messages passed among these components are available as well, for example images[3], IMU[4], detections[5], odometry[6], and control[7]. These features make ROS an ideal framework for the development of this autonomous MAV system.

[1]http://wiki.ros.org/RealSense
[2]http://wiki.ros.org/joy
[3]http://docs.ros.org/melodic/api/sensor_msgs/html/msg/Image.html
[4]http://docs.ros.org/melodic/api/sensor_msgs/html/msg/Imu.html
[5]http://docs.ros.org/melodic/api/vision_msgs/html/msg/Detection3D.html
[6]http://docs.ros.org/melodic/api/nav_msgs/html/msg/Odometry.html
[7]http://docs.ros.org/api/geometry_msgs/html/msg/Twist.html

Chapter 4

MAV State Estimation

Several previous works [39, 36] have already compared different vision-based state estimation algorithms using a well-known dataset [40]. However, since there are differences exist between our experimental platform and that of [40] (for example the IMU quality and sensor synchronization), a set of data is collected with our setup to evaluate their performances and select the most suitable one for this project. In this chapter, several state-of-art algorithms are introduced and compared based on our dataset.

4-1 Visual Odometry

The research of visual odometer has been undertaken for decades, the main idea of the visual odometer is tracking features in sequential views and estimating the camera's ego-motion. There are vast of approaches based on different image sensors (e.g. monocular [41, 42, 43], stereo [44, 45] and RGBD camera [46]). For the approaches with the monocular camera, specific initialization is necessary for the estimation of the landmarks' depth. Since only the bearing information of the extracted features can be extracted from the monocular image, the scale factor needs to be estimated by direct measurements or fusing with other sensors (e.g. IMU, range finder) [47]. Compared with monocular-based approaches, using a stereo camera can avoid the uncertainty of the scale factor. Known the baseline b of the stereo camera, landmarks' depth can be obtained via stereo triangulation. RGBD cameras also provide the direct measurement of the depth, however, it is too computationally heavy to perform the real-time dense image processing on nowadays single-board computer. For these aforementioned reasons, monocular-inertial or stereo-inertial are preferable for on-board MAV state estimation. Indeed, there are several state-of-art vision-inertial odometers, which have already shown their robustness and accuracy with power-limited computers (for example, OKVIS [6], ROVIO [34] and S-MSCKF [36]).

The architecture of a visual odometer consists of front-end and back-end. The front-end estimates camera's ego-motion by extracting and matching correspondences, and the back-end refines the estimated camera poses as well as the landmarks' positions. Typically, previous approaches can be divided into two categories: optimization-based and filtering-based.

Optimization-Based The optimization-based approaches find the maximum likelihood solution by assuming the measurement noise is distributed normally. The optimal camera poses $\mathbf{C}_j$ and the landmarks' positions in space $\mathbf{L}_i$ are obtained by minimizing the reproject error of n landmarks in m camera views:

$$\min_{\mathbf{C}_j,\mathbf{L}_i} \sum_{j=1}^{m}\sum_{i=1}^{n} d(\mathbf{l}_{ji}, \mathbf{C}_j(\mathbf{L}_i))^2, \tag{4-1}$$

where $\mathbf{l}_{ji}$ is $\mathbf{L}_i$'s corresponding feature in the camera view $\mathbf{C}_j$, and $d(\mathbf{a}, \mathbf{b})$ is the Euclidean distance between $\mathbf{a}$ and $\mathbf{b}$.

The problem of optimization-based approaches is that the computation of global bundle adjustment is extremely costly because it iteratively computes the optimal $3n+6m$ parameters, which will grow unbounded with the increase of m and n. Sparse bundle adjustment is one of the potential solutions to alleviate the computation load, because when solving Eq. (4-1) with Levenberg-Marquardt algorithm, the normal equation matrix has a certain sparse block structure [48]. Besides, to achieve real-time performance, computation complexity should not grow without bound. A common solution to reduce the computation complexity is only optimizing the keyframe (some keyframe selection methods are proposed in [49, 6]) within the sliding window.

Filtering-Based Compared with the optimization-based approaches, the filtering-based approaches are more computationally efficient but less accurate [6]. The filtering-base approaches take the camera extrinsic parameters $\mathbf{C}$ as well as the landmarks' positions $\mathbf{l}_i$ as the filter's states:

$$\hat{\mathbf{x}}_{filtering} = \begin{bmatrix} \hat{\mathbf{C}} \\ \hat{\mathbf{l}}_1 \\ \vdots \\ \hat{\mathbf{l}}_n \end{bmatrix}, \quad \mathbf{P}_{filtering} = \begin{bmatrix} \mathbf{P}_{\mathbf{CC}} & \mathbf{P}_{\mathbf{Cl}_1} & \cdots & \mathbf{P}_{\mathbf{Cl}_n} \\ \mathbf{P}_{\mathbf{l}_1\mathbf{C}} & \mathbf{Pl}_1\mathbf{l}_1 & \cdots & \mathbf{P}_{\mathbf{l}_1\mathbf{l}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{\mathbf{l}_n\mathbf{C}} & \mathbf{P}_{\mathbf{l}_n\mathbf{l}_1} & \cdots & \mathbf{P}_{\mathbf{l}_n\mathbf{l}_n} \end{bmatrix}. \tag{4-2}$$

In the prediction step, the robot's state and its uncertainty are calculated by using a predefined motion model. Once a new measurement is observed, the estimation and the uncertainty are updated with a measurement model [50].

Besides its inaccuracy, the uncertainty will grow unbounded with increasing traveled distance because of the linearization error in both the filter's propagation and update steps, which is detrimental to the state estimation performance. To alleviate this, some techniques are proposed (such as the robocentric local approach in [51] and anchor-centric parameterization in [52]).

Combining the visual measurement with the inertial measurement can always lead to a better estimation of the odometer because these two sensors have complementary characteristics. According to [6], visual-inertial fusion approaches can be separated into two main categories: loosely-coupled and tightly-coupled.

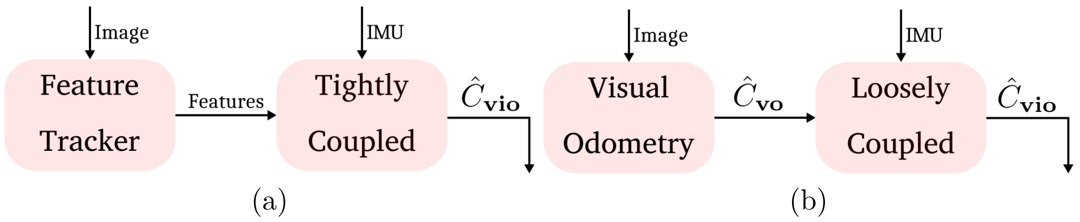


Figure 4-1: The block diagram of (a) loosely-coupled approaches and (b) tightly-coupled approaches.

Loosely-coupled Loosely-coupled is the simplest way to fuse the inertial measurement with the visual measurement, it treats the inertial measurement as an independent module to aid the vision-only estimation (which can be gathered from structure-from-motion) by using an Extended Kalman Filter (EKF).

Tightly-coupled Tightly-coupled jointly optimizes both the camera and IMU measurements from the raw measurement level. Compared with the loosely-coupled approaches, it is more accurate but computational heavier. An example given by [6] shows that the tightly-coupled approach outperform the loosely-coupled as well as the vision-only approaches (as shown in Figure 4-2).

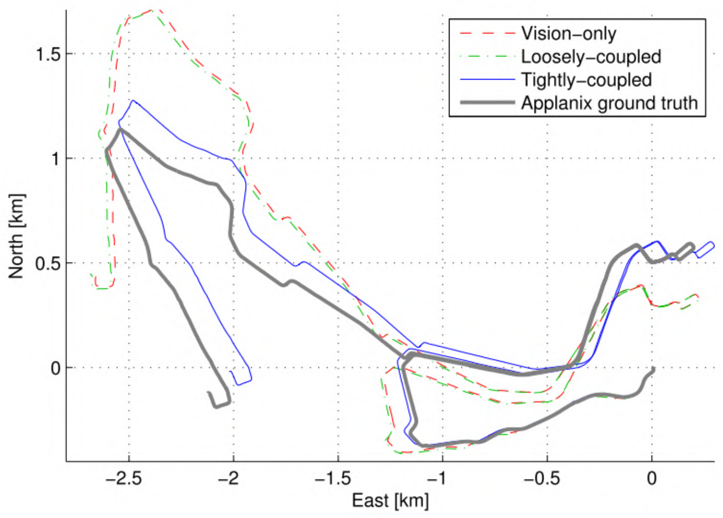


Figure 4-2: An example of visual-(inertial) fusion comparison [6].

4-1-1 Comparison of VO Algorithms

In order to figure out the most suitable visual-inertial algorithm for the designed experimental platform (as introduced in Chapter 3), three state-of-art approaches are investigated, namely Robust Visual Inertial Odometry (ROVIO) [34], Visual-Inertial System (VINS) [35] and Stereo Multi-State Constraint Kalman Filter (S-MSCKF) [36].

ROVIO ROVIO[1] is a filtering-based approach, which use monocular image and inertial measurements to estimate the odometry. However, [36] concludes that it may not perform well when the scene is far from the camera. Besides, its performance is not comparable with the other approaches during aggressive flights (around 17 m/s).

VINS VINS is an optimization-based approach, which performs non-linear optimization on a sliding window to obtain highly accurate visual-inertial odometry by fusing pre-integrated IMU and observed landmarks. The algorithm is originally designed for a minimum visual-inertial system (a monocular camera and a low-cost inertial measurement unit), and later it is extended as a multi-sensor state estimator[2]. In the following comparisons and evaluations, VINS with two different multi-sensor combinations are presented: stereo camera only, stereo camera with an IMU unit.

S-MSCKF S-MSCKF[3] is a filtering-based approach, which uses FAST detector [53] and Kanade-Lucas-Tomasi (KLT) tracker [54] for efficiency. It maintains sequential camera poses in the state vector and uses the visual measurements of the same landmark across multiple views to form a multi-constraint update. According to the author, S-MSCKF's accuracy is comparable with VINS in the fast flight test, while the computational load is much lower than the one of VINS.

Experiment Setup

The goal of this experiment is to test the performance of state-of-art visual-inertial odometry algorithms including ROVIO, S-MSCKF and VINS on a low-cost aerial vehicle platform. The selected low-cost hardware setup used for the data collection in this experiment consisting of a Parrot Bebop2[4], a forward-looking Intel RealSense Depth Camera D435i [5] visual-inertial sensor and a NVIDIA Jetson TX2[6] (as introduced in Chapter 3).

In addition to the the experimental platform, Optitrack[7] motion capture system was also used for the data collection (as can be seen from Figure 4-3). The onboard data was logged from the visual-inertial sensor, which includes 15 Hz stereo infra images with resolution 424×240 and 250 Hz IMU data. For the ground truth data, the motion capture system measured the aerial vehicle's position and orientation by tracking the reflective markers adhered to it. The external measurement was highly accurate and logged at 150 Hz.

Experiment Result

In order to perform a quantitative analysis, the trajectories estimated by the aforementioned visual odometry algorithms were firstly aligned with the ground truth trajectory measured

[1]https://github.com/ethz-asl/rovio
[2]https://github.com/HKUST-Aerial-Robotics/VINS-Fusion
[3]https://github.com/KumarRobotics/msckf_vio
[4]https://www.parrot.com/us/drones/parrot-bebop-2#lightweight
[5]https://www.intelrealsense.com/depth-camera-d435i/
[6]https://developer.nvidia.com/embedded/buy/jetson-tx2
[7]https://optitrack.com/

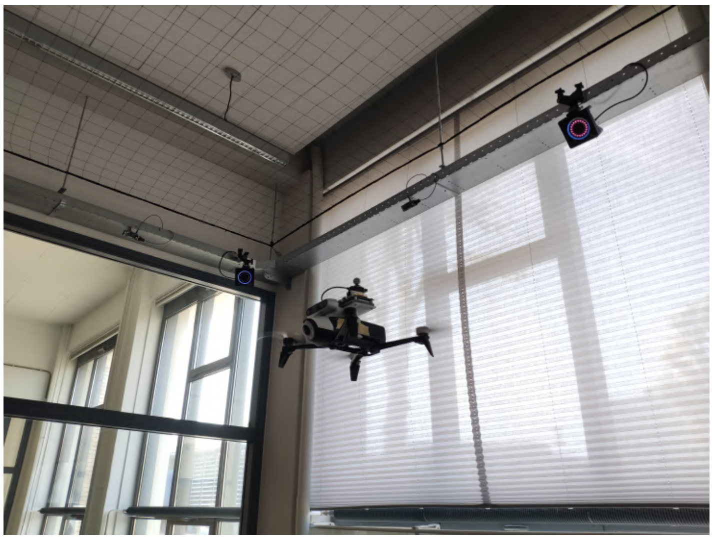


Figure 4-3: Parrot Bebop2 with a forward-looking visual-inertial sensor and an onboard computational unit. The reflective markers are equipped for collecting ground truth.

by the motion capture system. The trajectory's alignment transformation matrix T was calculated with only the first $N = 100$ logged estimated positions $\{\hat{\mathbf{p}}_{raw_i}\}_{i=0}^{N-1}$ and their corresponding ground truth position $\{\mathbf{p}_i\}_{i=0}^{N-1}$. By solving the minimization problem on the distance between the $\hat{\mathbf{p}}_{raw_i}$ and $\mathbf{p}_i$, the aligned estimation $\{\hat{\mathbf{p}}_i\}_{i=0}^{N-1}$ can be computed.

$$\begin{aligned} T &= \arg\min_{T} \sum_{i=0}^{N-1} \|\mathbf{p}_i - \mathbf{T}\hat{\mathbf{p}}_{raw_i}\|^2, \\ \hat{\mathbf{p}}_i &= \mathbf{T}\hat{\mathbf{p}}_{\mathrm{raw_i}} \qquad \mathrm{i} = 0, 1, 2, \dots, \mathrm{N} - 1. \end{aligned} \tag{4-3}$$

Figure 4-4 gives an intuitive comparison result of trajectories estimated by the aforementioned visual odometry algorithms. In order to firstly evaluate the scaling accuracy of different approaches, the estimated trajectories and their corresponding ground truth were firstly divided into $K = 10$ sub-trajectories $\{\hat{Traj_i}\}_{i=0}^{K-1}$ and $\{Traj_i\}_{i=0}^{K-1}$. And the scale error was then computed by calculating the ratio between the traveled distance of the estimated sub-trajectories $dist(\hat{Traj_i})$ and their corresponding ground truth $dist(Traj_i)$.

$$\Delta s_i = \frac{1}{K} \sum_{i=0}^{K-1} |\frac{dist(\hat{Traj_i})}{dist(Traj_i)} - 1| \tag{4-4}$$

An overview of the aforementioned visual odometry algorithms' scaling errors is shown in the Table 4-1. As can be seen, the scale error of ROVIO is much higher than the others'. This is mainly because ROVIO is a monocular-inertial odometer, whose scale factor is recovered with the low-quality IMU measurements, and the noisy inertial information led to the scaling ambiguity. While S-MSCKF and VINS are stereo-inertial odometry, which does not have the scale ambiguity once the stereo rig is well-calibrated.

To evaluate the proposed algorithm's local accuracy, the Relative Pose Error (RPE) of each estimated position is calculated with the equation given by [55]. The estimated trajectory

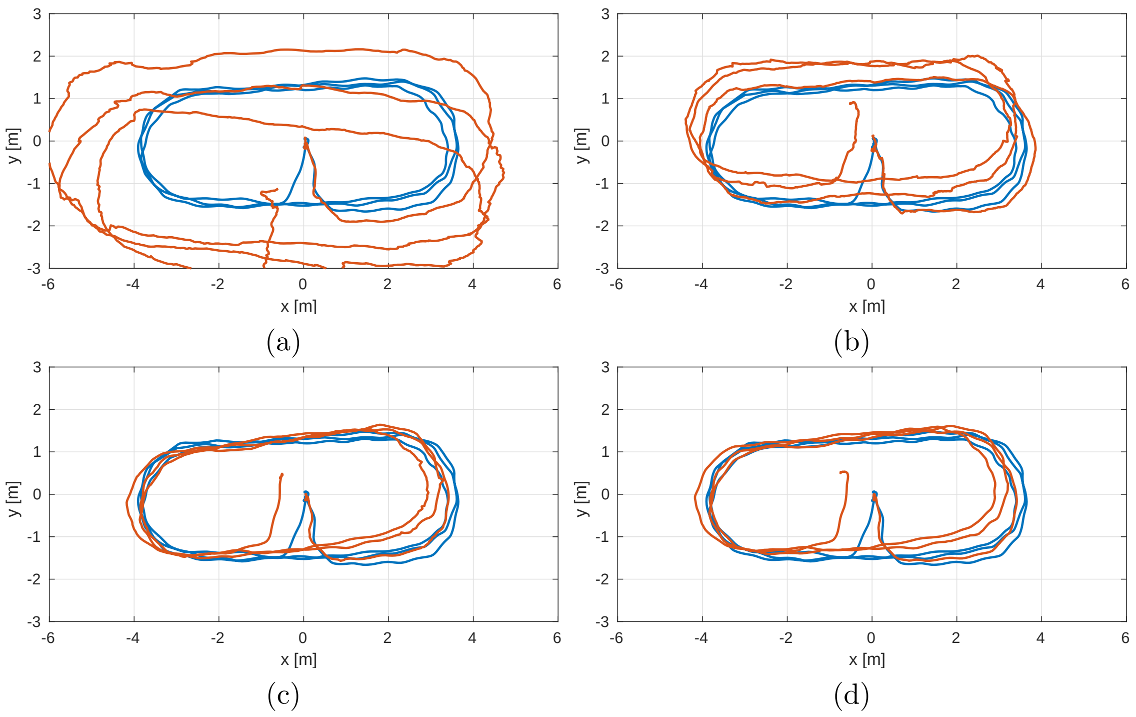


Figure 4-4: Red trajectories are estimated by (a) ROVIO, (b) S-MSCKF, (c) VINS (Stereo), (d) VINS (Stereo+IMU) respectively. The estimated trajectories (red) are aligned with the ground truth (blue).

Table 4-1: The scale error of proposed visual (-inertial) odometry algorithms.

VO	**Scale Error Variance**	**Scale Error Mean**
ROVIO	0.02650	0.31032
S-MSCKF	0.00110	0.05059
VINS (Stereo)	0.00055	0.04433
VINS (Stereo+IMU)	0.00050	0.04291

was firstly divided into M sub-trajectories with constant time interval $\Delta t = 0.5s$, and the Relative Pose Error was extracted by

$$RPE = \sum_{i=0}^{M-1} trans\left(\left(\mathbf{Q}_{\mathrm{i}}^{-1}\mathbf{Q}_{\mathrm{i+\Delta t}}\right)^{-1}\left(\mathbf{P}_{\mathrm{i}}^{-1}\mathbf{P}_{\mathrm{i+\Delta t}}\right)\right) \tag{4-5}$$

where $\mathbf{P}_i \in SE(3)$ and $\mathbf{P}_{i+\Delta t} \in SE(3)$ represent the starting and ending pose of the aligned estimated trajectory, $\mathbf{Q}_i \in SE(3)$ and $\mathbf{Q}_{i+\Delta t} \in SE(3)$ are their corresponding ground truth and $trans(\bullet)$ is the function extracting the translational error.

The average translational components of the Relative Pose Error were then extracted and used as the representation of the local accuracy. As can be seen from Table 4-2, optimization-based approaches (VINS (Stereo) and VINS (Stereo+IMU)) are more accurate than filtering-based approaches (ROVIO and S-MSCKF). Note that the high translational error of ROVIO mainly due to the scale ambiguity.

Table 4-2: The RPE of proposed visual odometry algorithms on our dataset.

VO	**Translational RPE (m)**
ROVIO	0.14701
S-MSCKF	0.08425
VINS (Stereo)	0.07647
VINS (Stereo+IMU)	0.07481

In addition to the accuracy, the computational loads of the aforementioned algorithms are another critical concern. The average computational load and the processing time of each aforementioned visual odometry with an Nvidia Jetson TX2 were then counted and listed in Table 4-3.

Table 4-3: The average computational load and processing time of proposed visual odometry algorithms.

VO	**CPU (%)**	**Processing Time (ms)**
ROVIO	10.40	41.3
S-MSCKF	15.06	26.9
VINS (Stereo)	17.51	134.7
VINS (Stereo+IMU)	19.27	138.3

The result is identical with expectation since VINS is an optimization-based method, it should be computationally heavier than filtering-based methods (ROVIO and S-MSCKF). And ROVIO is the computationally lightest approach because it is monocular-inertial, the image processing of monocular images is much faster than those of stereo images. From the result of the processing time, VINS is not real-time as its processing time is longer than the sampling time of the stereo camera.

Summary

To summarize, the selection of a suitable state-of-art visual odometry for a low-cost aerial vehicle platform (for example the aerial vehicle platform introduced in section 4-1-1) requires the considerations of their accuracy and computational load. In this section, the comparison of four different algorithms' accuracy shows that the monocular approach can not provide a reliable estimation since the low-cost IMU provides noisy inertial measurements, which introduces scale ambiguity. Additionally, Table 4-2 shows that the optimization-based method is more accurate than the filtering-based method. Nevertheless, in terms of the computational load and the processing time, the optimization-based methods are much computationally heavier than the filtering-based methods (as shown in Table 4-3). S-MSCKF outperforms among the proposed visual odometry algorithms since it is computationally light. It occupies about 15.06% of the CPU and takes 26.9ms for each pair of stereo images which illustrates that it can be run in real-time. Besides, its transnational Relative Pose Error is approximately 0.07780 m, which is comparable with optimization-based algorithms (such as VINS). These facts leads to the conclusion that S-MSCKF is the most suitable visual odometry for the specified aerial vehicle platform.

4-2 Vision-based Simultaneous Localization and Mapping

Different from odometry, which always suffers from long term drifting, Vision-based Simultaneous Localization and Mapping (VSLAM) solves this problem by not only tracking landmarks across multiple camera views but also simultaneously constructing and updating the environment. Actually, in some modern VIO approaches, this process is already integrated (for example the loop-closure module in [35]). And to maintain the environment, VSLAM algorithms need to undergo several stages which include map initialization, place recognition and optimization. In this section, two state-of-art VSLAM algorithms are compared and evaluated: namely ORB-SLAM [45] and VINS (with loop-closure).

4-2-1 Comparison of VSLAM Algorithms

In this section, state-of-art VSLAM algorithms are introduced, compared and evaluated.

VINS (loop-closure) As introduced in Section 4-1, VINS estimates the relative camera's transformation by solving a non-linear optimization problem within the sliding window. In this section, the loop-closure module is also integrated so that a full VSLAM system is evaluated. This module uses DBoW2 [56], a state-of-art bag-of-word place recognition approach, to perform place recognition. Once a loop is detected, the current sliding window and the past poses are aligned by jointly optimizing all the visual-inertial measurements and retrieved features.

ORB-SLAM (online) ORB-SLAM is another feature-based VSLAM algorithm, it extracts sparse Oriented FAST and Rotated BRIEF (ORB) features from the camera images and estimates the camera poses as well as to perform a sparse reconstruction of the environment. When there is no loop detected, it performs local bundle adjustment to estimate the camera poses; while a loop is detected, it then corrects the loop key-frame as well as all the map points seen by the loop key-frame.

ORB-SLAM (offline) Nevertheless, VSLAM algorithms are so computationally heavy that they can not be performed with a low-cost computational unit at a high frame rate. To tackle this problem, ORB-SLAM is run in offline mode, which only performs place recognition: a sparse feature map is constructed with stereo camera in advanced, and monocular images are then used to extract the landmarks and estimate the camera poses.

Experiment Setup

In order to evaluate different VSLAM methods' real-time performances, two scenarios were created, in the first scenario, all the aforementioned algorithms were fully online including both the localization and mapping; in the second scenario, only the re-localization with monocular camera was run online while the mapping of the static indoor environment was built offline. To achieve this, another dataset was recorded with the same platform and in the same environment for map building and the dataset captured in section 4-1-1 was then used for latter comparison between VO and VSLAM algorithms.

Experiment Result

Different from the relative estimation obtained from visual odometry algorithms, in the case that the aerial vehicle is flying in a known scene, the estimated pose of VSLAM comes from the direct measurement. For this reason, instead of the first N logged data, the estimated positions and their corresponding ground truth were used in this experiment with Equation 4-3. The alignment results of the aforementioned methods are shown in Figure 4-5.

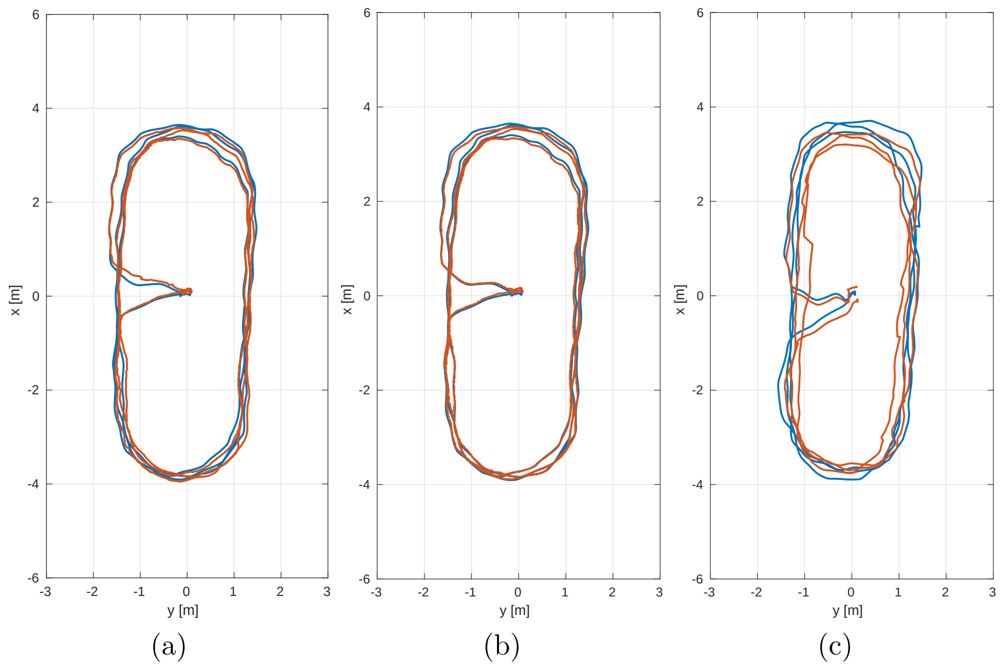


Figure 4-5: Red trajectories are estimated by (a) ORB-SLAM (online), (b) ORB-SLAM (offline), (c) VINS (Stereo+IMU+loop closure) respectively. The estimated trajectories are aligned with the ground truth(blue).

For the evaluation of their accuracy, the Absolute Trajectory Error (ATE) was used in this experiment since we now more interest in their global consistency instead of their local accuracy. By using the Equation 4-6, the Absolute Trajectory Error is calculated and listed in Table 4-4.

$$ATE = \sum_{i=0}^{N-1} \left(\mathbf{Q}_{\mathrm{i}}^{-1} \mathbf{P}_{\mathrm{i}} \right) \tag{4-6}$$

Table 4-4: The ATE of proposed VSLAM algorithms on our dataset.

VSLAM	**Translational ATE (m)**
ORB-SLAM (online)	0.13765
ORB-SLAM (offline)	0.09239
VINS (Stereo+IMU+loop closure)	0.20171

From the ATE result, ORB-SLAM (offline) has the most accurate result, this is mainly because with a pre-generated map, it always directly measures its position by re-localization. While the online method estimates relative pose when it is in an unknown environment, and this will certainly keep introducing accumulated error. VINS has the highest ATE among the three proposed algorithms since ORB [57] is a more distinctive feature than Shi-Tomasi [58], which illustrates that it is easier for ORB-SLAM to re-localize the camera.

As for the computational load and the processing time of the aforementioned VSLAM algorithms, a statistical result is given in Table 4-5. The ORB-SLAM (offline) is the most computationally light among the three and it is the only real-time approach since it only uses the monocular camera. Besides, unlike ORB-SLAM (online) and VINS (Stereo+IMU+loop closure), ORB-SLAM offline only has the re-localization process, while the other two run the localization and mapping simultaneously.

Table 4-5: The computational load and processing time of proposed VSLAM algorithms.

VSLAM	**CPU (%)**	**Processing Time (ms)**
ORB-SLAM (online)	46.18	285.4
ORB-SLAM (offline)	28.39	70.5
VINS (Stereo+IMU+loop closure)	25.94	139.6

Summary

ORB-SLAM (offline) is the only one that compatible with the low-cost aerial vehicle among the three proposed VSLAM algorithms since it runs at the rate approximately the same as the 15Hz camera. Besides of the computational load, the offline approach provides the most accurate estimation among the three because the map is generated offline, which ensures the accurate re-localization result.

4-3 Concluding Remarks

In this chapter, different state-of-art visual odometry and vision-based simultaneous localization and mapping algorithms are evaluated with the proposed low-cost aerial vehicle platform (as introduced in Chapter 3). Their accuracy and their computational load are compared in order to find out the most suitable algorithm for MAV state estimation. From the comparison among VOs, S-MSCKF provides the most efficient performance among the proposed methods, because it only consumes 15.06% CPU while its RPE is comparable with optimization-based approaches. And from the comparison among VSLAMs, ORB-SLAM (offline) is the only approach can be run in real-time. An overview of the comparison of S-MSCKF and ORB-SLAM (offline) is shown in Table 4-6.

Table 4-6: Comparison between S-MSCKF and ORB-SLAM (offline).

Methods	**RPE (m)**	**ATE (m)** (m)	**CPU (%)**	**Processing Time (ms)**
S-MSCKF	0.08425	0.64968	15.06	26.9
ORB-SLAM (offline)	0.06545	0.09239	28.39	70.5

It is obvious that VSLAM is more accurate than VO as it localizes the camera with not only the recent visual information but the global map which is updated at run-time. On the other side, VO is computationally light because it only estimates camera poses with only the local information. The performance of ORB-SLAM (offline) is very sensitive to the environments, which suggests that the localization performance will be largely affected if the environment is different from the one from which the map is generated. Thus, a future selection between the recommended two approaches (S-MSCKF and ORB-SLAM (offline)) requires further consideration of other functionalities of the applications. For indoor and long traveling distance navigation, we should opt to use ORB-SLAM (offline) as it provides drift-free state estimation which is expressed in map coordination. While in the scenario that the aerial vehicle navigates in an unknown environment, S-MSCKF should be used since ORB-SLAM (offline) requires a pre-generated map, which is impossible for this scenario. In addition, for outdoor navigation, because of the change of the environment (such as daylight, moving objects, etc.), S-MSCKF will be preferable. Thus, S-MSCKF is a better choice for MAV autonomous navigation in highly dynamic environments.

Chapter 5

Obstacle Sensing

To navigate safely, the MAV is required to precept the obstacles in its local environment. Concerning the avoidance of moving obstacles, the obstacle sensing system must be able to extract essential information of them, which includes their positions, sizes, and motions. Besides, due to the imperfect measurement, error exists in the extracted information, which should be also considered to perform safe navigation. In this chapter, some state-of-art obstacle sensing methods are introduced, and an obstacle sensing method is proposed and evaluated.

5-1 Obstacle Detection

For an autonomous MAV navigation system, the obstacle detection method has to be robust and computationally light so that a real-time obstacle detection can be performed with limited computational power at a high frame rate. In the past decades, a variety of obstacle detection algorithms have been done in allowing autonomous robots to precept obstacles around them. Some of them use sophisticated information which requires heavy computation while some others aim to extract coarse obstacle's information at high-frequency.

5-1-1 Obstacle Representation

The first consideration of the detection algorithm is the representation of the obstacles. Typically, the obstacles' representations can be categorized into two classes: occupied cell map and extended objects.

Occupied Cell Map The occupied cell map samples the three-dimensional space with unit cells. This representation is not computationally efficient to be processed as it always contains massive data. For example in Figure 5-1, a typical 3D point cloud generated by the RealSense Depth Camera D435i contains 101760 points (the resolution is set as 424×240). In addition, these data always contain measurement noise, which is detrimental to the detection of each single unit cell. A simple solution to this is down-sampling the data. For example in [10], the ranging data from a stereo camera is firstly mapped to a 3D spherical grid. Each point in space is then treated as a virtual ray extended from the stereo camera, and the median ray in each spherical cell is then extracted. With this method, the computation load is decreased and the measurement noise is filtered out by the median filter. After that, a binary occupancy map is then constructed based on the virtual scan result. Other than using a binary occupancy map, OctoMap [59] uses a probabilistic occupancy estimation to model the environment, it efficiently stores the map by using octrees and reduces the sensor noise with probabilistic modeling and updating method. With this kind of obstacle representation, the MAV can extract a free space by allocating a convex region and ensuring its current and future positions lie within it. For example in [22], the MAV have successfully navigated in static environments with velocity up to 3.65 m/s.

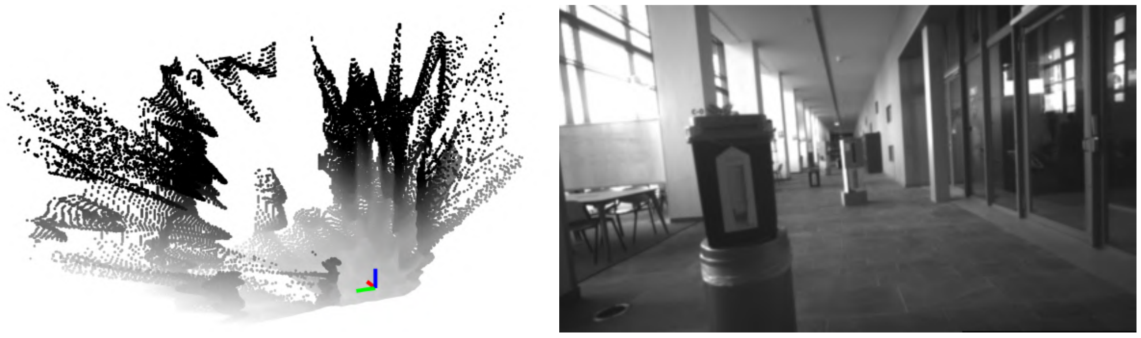

(a) (b)

Figure 5-1: (a) A three-dimensional point cloud generated by RealSense Depth Camera D435i and (b) its corresponding grayscale image. Points are marked by colors according to their heights.

Extended Objects Nevertheless, the obstacle's kinematic (which including positions, velocities, and orientations, etc.) can not be extracted with the aforementioned representations, which is necessary information for moving obstacle avoidance. Instead of using the occupied cell map, some other approaches use the extended objects to represent the detection obstacle. For example, cubes are used in [60, 61] and ellipsoids are used in [3, 29]. Compared with the demanding computation complexity and memory used for the occupied cell map, the extended objects group the sensing data into bounding shapes. This will certainly decrease the computation of collision checking and the memory required to store the obstacle. An example is shown in Figure 5-2, two bounding ellipsoids are extracted from the depth image so that the collision checking only needs to be performed for the two detections instead of all the 3D point in space. Besides, extended objects are easier to be used for data association (determine whether detection in two different camera views belongs to the same object), this property can be used later for the estimation of the obstacles' kinematics (as can be seen from Figure 5-2(d)). These properties lead to the conclusion that the extended object representation is preferable for moving obstacle detection with limited computational power.

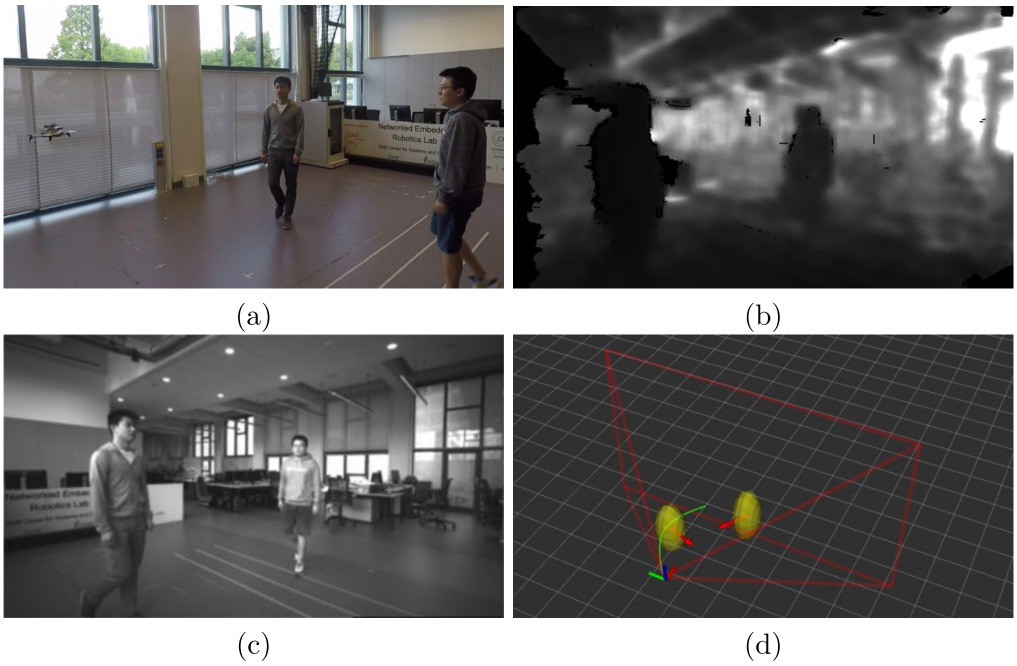


Figure 5-2: An example of the extended object detection. (a) A snapshot of the scene. (b) The onboard depth image used for extended object extraction. (c) The onboard gray-scale image corresponding to the depth image. (d) Visualization of the detection result, two ellipsoids (yellow) are extracted and used to plan a collision-free trajectory (green).

5-1-2 Detection Methods with Range Data

To extract the three-dimensional objects' information, range data is used in a vast number of approaches. Some of them are so computational heavy that they can not be performed on the onboard computational unit. For example 3D convolutional network, such as 3D Region Proposal Network [60] and VoxelNet [61], takes the range data as input and output 3D bounding boxes (expressed by objects' positions, sizes and orientations). However, these methods require powerful GPUs to perform the 3D convolutional network, their heavy computational cost makes them difficult to be implemented with limited computational power for real-time applications. In addition, these learning-based methods can only detect objects which are presented in the training set (for example pedestrians, vehicles and cyclists, etc). Thus, they are not suitable for object detection in unknown environments. Some others are originally designed for ground vehicles, which will not work in the applications of aerial vehicles. For example, stixel world [62] uses the disparity map to detect the vertical obstacles in the camera's field of view, assuming the camera is parallel to the horizon and all the obstacle touch the ground plane. However, for aerial vehicles, the camera's optical axis is not necessary to be parallel to the ground plane because of the pitch and roll rotations. Even though the raw image can be rotated to overcome this problem, stixel world will also not work once the ground plane is not in the field of view. Besides, this method can not detect floating objects (for example, another aerial vehicle) because they are not on the ground plane. Other than those aforementioned approaches, U-V map is used to detect obstacles in [11, 63], which can be used to extract the bounding ellipses with light computational power. According to the author of [11], the average computation time of their detection method is 5.49 ms on a low-cost computer. Thus, in this thesis work, a computationally light 3D obstacle detection method is developed based on the idea of U-V map.

5-1-3 3D Obstacle Detection with U-depth Map

In this section, the 3D obstacle detection method is described and evaluated. This detection method models each obstacle as a three-dimensional cube and detect its position $\mathbf{p}_{\text{obs}}$ and size $\mathbf{s}_{\mathbf{obs}} = (\mathbf{l_o}, \mathbf{w_o}, \mathbf{h_o})$ based on the camera depth image. The length, width and horizontal position (x-y plane) are firstly derived from the U-depth map, and the height and the vertical position (z-direction) of the obstacle is then extracted directly from the depth image.

U-depth Map

The first step of the detection method is to extract the U-depth map from the depth image. The U-depth map is computed with the column depth value histograms of the depth image generated by depth camera. Figure 5-3 shows an example of the extraction of U map.

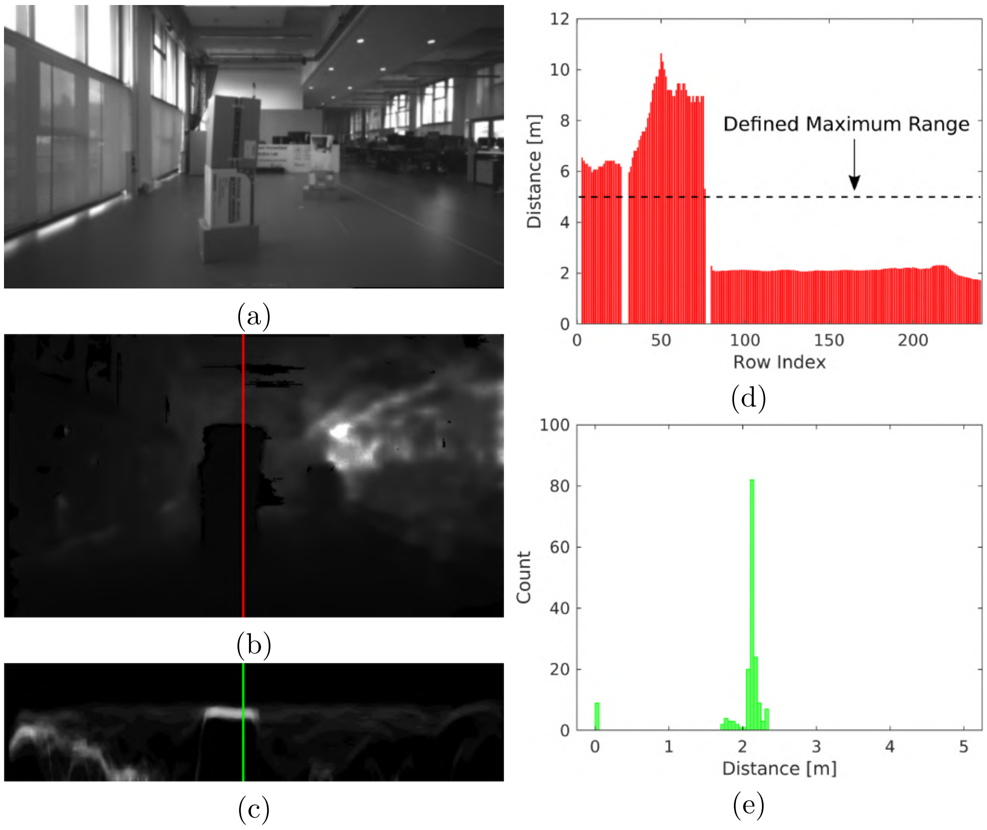


Figure 5-3: The extraction of U map. (a) The onboard gray scale image. (b) The depth image and the extracted U map. (c) The U map. (d)The corresponding data to the red column in the depth image, and (d) is the histogram computed from (d), which is corresponding to the green column in (c).

Note that the maximum range is set as 5 meters in order to generated histograms with a fixed range (from 0 to 5 meters) with a high resolution (the bin's width is set as 5 centimeters). Another reason is that the average depth measurement error increase with the distance, the measurement farther than 5 meters is not reliable.

As can be seen from Figure 5-3(a), there is an obstacle presented around 2 meters in front of the camera. And in one of its corresponding columns in the depth image (marked by red), most of the pixels contain the value around 2 m (as shown in Figure 5-3(d)). This then results in the large bin size in the histogram Figure 5-3(e), which is corresponding to the green column in the U map. Based on this property, a bin of the histogram is considered as a point of interest if its value is larger than a threshold T_{POI}, which is defined as:

$$T_{POI} = \frac{fT_{h_{obs}}}{d_{bin}}, \tag{5-1}$$

where f is the camera's focal length, $T_{h_{obs}}$ is a predefined threshold for obstacle's height in the space and d_{bin} is the corresponding mean depth of a bin in the histogram.

Then these points of interest are grouped with other candidates located in their neighborhood so that a two-dimensional bounding box can be extracted from the U map (as demonstrated in Figure 5-4).

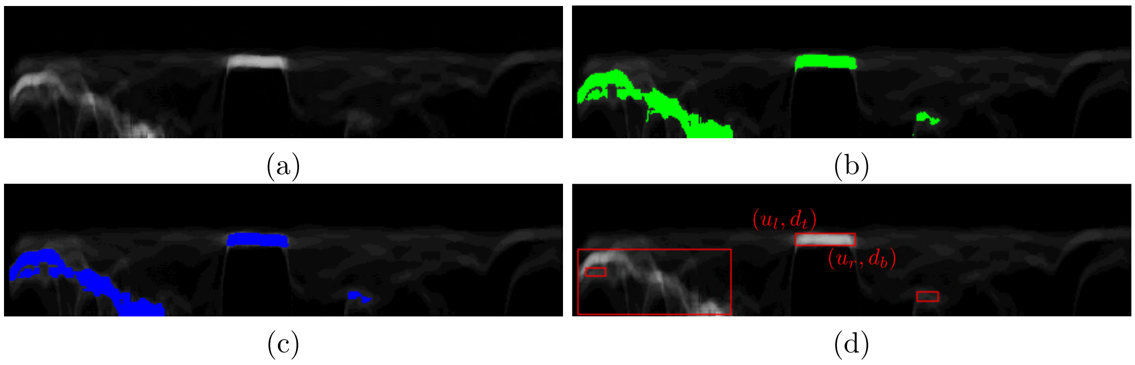


Figure 5-4: The extraction of bounding boxes from the U map. (a) The computed U map. (b) The point of interest. (c) Points are connected into line segments. (d) The extracted bounding boxes.

The contiguous lines (blue lines in Figure 5-4(c)) are generated by horizontally connecting the point of interest, and only the line whose length is longer than a predefined threshold T_{LINE} are considered later in the bounding box extraction, which is defined as:

$$T_{LINE} = \frac{fT_{w_{obs}}}{d_{bin}}, \tag{5-2}$$

where $T_{w_{obs}}$ is a predefined threshold for obstacle's width in the space.

Then the bounding boxes (red boxes in Figure 5-4(d)) are extracted by finding the top-left (u_l, d_t) and bottom-right (u_r, d_b) corners of each line cluster. And the obstacle's length l_o and width w_o as well as horizontal position (x_o, y_o) can be computed from the bounding box. It is expressed as:

$$x_o^B = d_b, \tag{5-3}$$

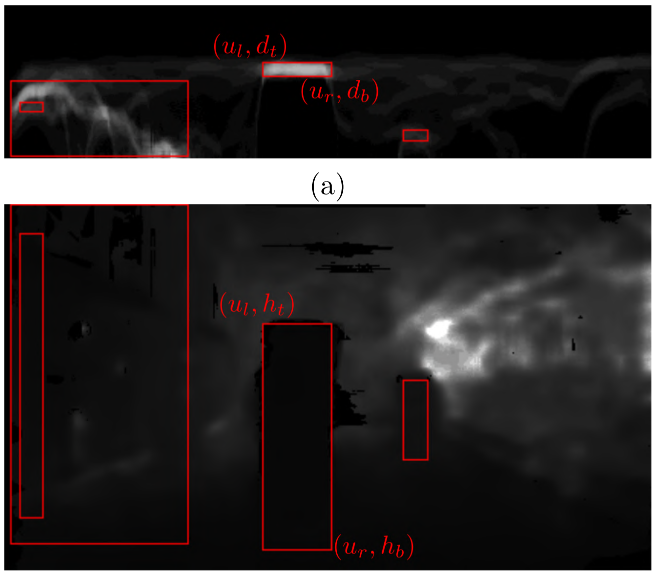
$$y_o^B = \frac{(u_l + u_r)\, d_b}{2f}, \tag{5-4}$$

$$l_o^B = 2\,(d_b - d_t)\,, \tag{5-5}$$

$$w_o^B = \frac{(u_r - u_l)\, d_b}{2f}, \tag{5-6}$$

where the super index B indicates the position expressed in the MAV body frame (while W indicates the world frame).

Then a corresponding bounding box of the obstacle in the original depth image can be extracted by grouping depth image points whose horizontal index are within $[u_l, u_r]$ and depth values are within $[d_t, d_t + l_o]$. An example is shown in Figure 5-5, let (u_l, h_t) and (u_r, h_b) be the top-left and bottom-right corners of the bounding box in the original depth image.



(b)

Figure 5-5: The extraction of bounding boxes from the U-depth map. (a) The bounding box extracted from the U map. (b) The bounding box extracted from the depth image.

The height of the obstacle h_o and its vertical position z_o can be computed by

$$z_o^B = \frac{(h_t + h_b)\, d_b}{2f}, \tag{5-7}$$

$$h_o^B = \frac{(h_t - h_b)\, d_b}{f}. \tag{5-8}$$

And the uncertainty of the obstacle detection is formulated based on the depth measurement error model in [64], which consists of the obstacle's position uncertainty covariance $\mathbf{\Sigma}_o^B$ and the obstacle's size uncertainty covariance $\mathbf{\Sigma}_{o,s}^B$. Then the detected obstacle and its covariance matrix are expressed in the world frame by considering the MAV's real-time state estimation:

$$\mathbf{p}_o^W = R_B^W \mathbf{p}_o^B + \mathbf{p}^W, \tag{5-9}$$

$$\mathbf{s}_o^W = R_B^W s_o^B, \tag{5-10}$$

$$\mathbf{\Sigma}_o^W = R_B^{W^T} \mathbf{\Sigma}_o^B R_B^W + \mathbf{\Sigma}^W, \tag{5-11}$$

$$\mathbf{\Sigma}_{o,s}^W = R_B^{W^T} \mathbf{\Sigma}_{o,s}^B R_B^W, \tag{5-12}$$

where $\mathbf{p}^W$ and R_B^W are the estimated MAV position and orientation, and $\mathbf{\Sigma}^W$ is the uncertainty matrix of the estimated MAV position.

5-1-4 Detection Experiment

In order to evaluate the performance of the proposed obstacle detection method, the MAV hovers in front of two static obstacles in the lab space (as shown in Figure 5-6). For the ground truth data, the positions of the obstacles are measured by the motion capture system and the obstacles' sizes are measured manually. The actual camera pose is measured by the external measurement as well, however, the camera-MOCAP calibration must be performed in advance (which is introduced in Section A).

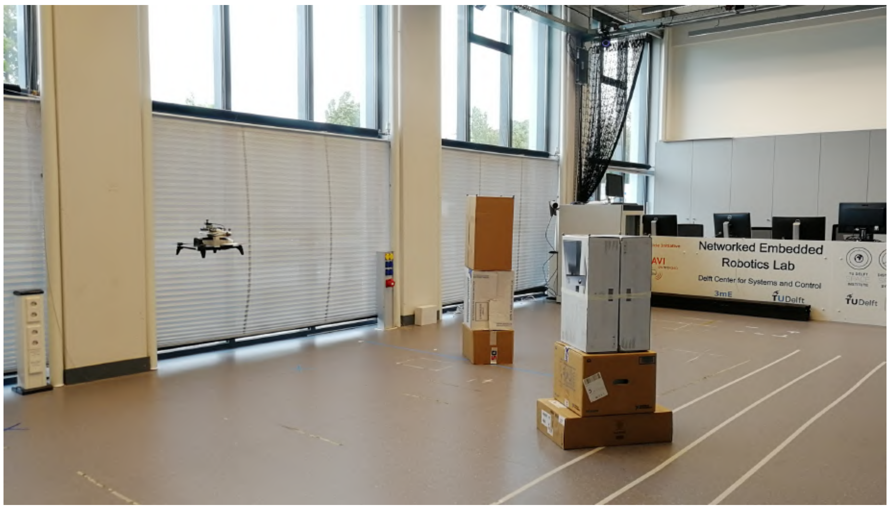


Figure 5-6: The snapshot of the experiment setup: MAV is hovering in front of two static obstacles.

The logged detection result is then compared with ground truth by simply compared the six detected facades with their actual positions. As can be seen from the result in Figure 5-7, for the top facade, the detected result is approximately the same as the ground truth, while for the bottom facade, the detected one is lower than the actual value because part of the ground plane is also included in the detection (its depth value lies within $[d_t, d_t + l_o]$). This also leads to a large variance in the obstacle's vertical position and height. As for the back facade, the difference between the detection and the actual value is larger than the others, since the back surface can not be seen by the camera, the detector assumes the obstacle is rounded and estimates the obstacle's length by multiplying the length of its front surface (see Equation 5-5).

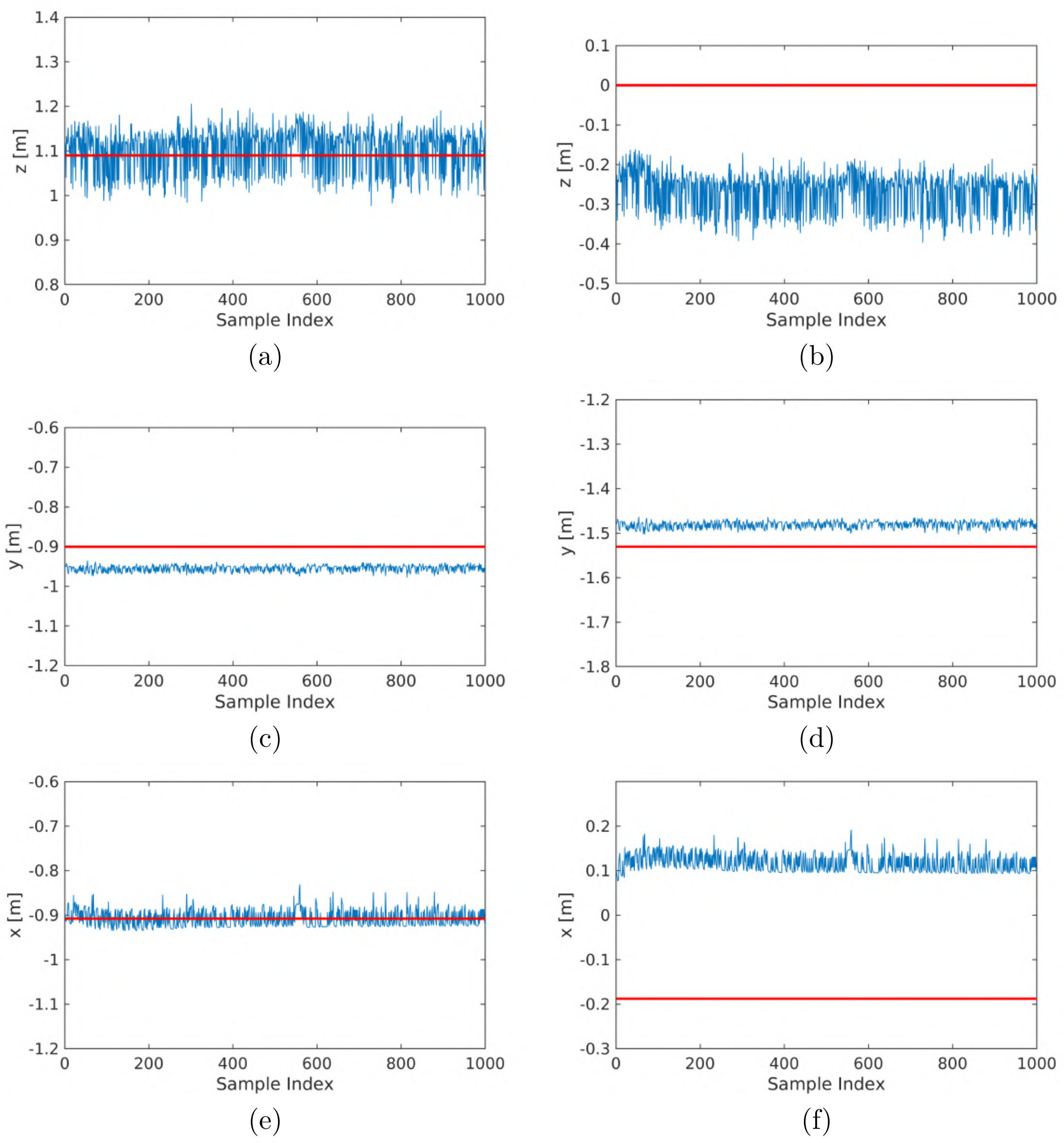


Figure 5-7: The comparison between an obstacle's six detected facades (blue) and their actual value (red). (a) The top facade. (b) The bottom facade. (c) The left facade. (d) The right facade. (e) The front facade. (f) The back facade.

5-2 Obstacle Tracking

In addition to the obstacle detection, the obstacle sensing module is also required to estimate obstacles' motions so that their future positions can be predicted. To tackle this problem, the sensing module store each obstacle's states, which includes position, size, and velocity. And to acquire obstacles' velocities, obstacle tracking module must be implemented to associate detection at different timestamps and estimate their motions. In this chapter, the proposed method is introduced and evaluated.

5-2-1 Data Association

To perform data association between two sequential frames, the probability, which determine if the two detections belong to the same object, is computed with the Gaussian probability density function:

$$p = p_G(\mathbf{x}_o^m \mid \hat{\mathbf{x}}_o^{m|m-1}, P_o^{m|m-1}), \tag{5-13}$$

where $p_G(\bullet)$ is the probability density function of the multivariate Gaussian distribution, $\mathbf{x}_o = (\mathbf{p}_o^W, \mathbf{s}_o^B)^T$ and $P_o = diag(\mathbf{\Sigma}_o^W, \mathbf{\Sigma}_{o,s}^B)$ are the obstacle state (position and size) and their corresponding uncertainty covariance. The super index m indicates the current frame, $\hat{\mathbf{x}}_o^{m|m-1}$ and $P_o^{m|m-1}$ are the predicted states and covariance matrix based on the previous detection. If the probability p is larger than a threshold, the two detections are then recognized as the same moving obstacle. If a new detected obstacle is not associated with any previous detection, its position, size and covariance matrix are then set as the detection result (as introduced in Section 5-1), while its velocity is initialized as zeros. Nevertheless, in some cases, a previous obstacle may not be associated with any new detection, its information will be stored in the memory, and its position will be kept updating by assuming a constant linear velocity until it is too far away from the MAV. The purpose is that when an obstacle is unseen by the camera due to occlusion, the sensing module can still estimate its position and velocity until it appears again in the view.

5-2-2 Estimation and Prediction

Since the detector only extracts obstacles' positions and sizes, a Kalman filter is then used to acquire obstacles' velocities and refine the obstacles' positions and sizes. Let $\hat{\mathbf{p}}_o^m$, $\hat{\mathbf{v}}_o^m$ and $\hat{\mathbf{s}}_o^m$ be an estimated obstacle's position, velocity and size at step m, and let $\mathbf{\Sigma}_o^m$, $\mathbf{\Sigma}_{o,v}^m$ and $\mathbf{\Sigma}_{o,s}^m$ be their corresponding covariance matrices. The prediction model used in the Kalman filter can then be written as:

$$\begin{bmatrix} \mathbf{p}_o^{m+1} \\ \mathbf{v}_o^{m+1} \\ \mathbf{s}_o^{m+1} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & diag(\Delta t) & 0 \\ 0 & \mathbf{I} & 0 \\ 0 & 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{p}_o^m \\ \mathbf{v}_o^m \\ \mathbf{s}_o^m \end{bmatrix}, \tag{5-14}$$

where Δt is the interval time between step $m+1$ and m.

And since only the obstacles' positions and sizes are extracted, the innovation matrix H is written as:

$$H = \begin{bmatrix} \mathbf{I} & 0 & 0 \\ 0 & 0 & \mathbf{I} \end{bmatrix}. \tag{5-15}$$

The constant velocity model is used to predict obstacles' future positions and uncertainty covariance within a short-time horizon. The prediction function is written as:

$$\mathbf{p}_o^{m+1} = \mathbf{p}_o^m + \mathbf{v}_o^m \Delta t, \tag{5-16}$$

$$\mathbf{v}_o^{m+1} = \mathbf{v}_o^m, \tag{5-17}$$

$$\mathbf{\Sigma}_o^{m+1} = \mathbf{\Sigma}_o^m + \mathbf{\Sigma}_{o,v}^m \Delta t^2, \tag{5-18}$$

$$\mathbf{\Sigma}_o^{m+1} = \mathbf{\Sigma}_o^m. \tag{5-19}$$

Note that obstacles' sizes are assumed to be constant and their uncertainties are not considered.

5-2-3 Tracking Experiment

The purpose of this experiment is to evaluate the performance of the proposed obstacle tracking method. As shown in Figure 5-8, the MAV hovers in front of two pedestrians in the lab space. The estimated pedestrians' positions and velocities are compared with the logged ground truth data measured by the motion capture system.

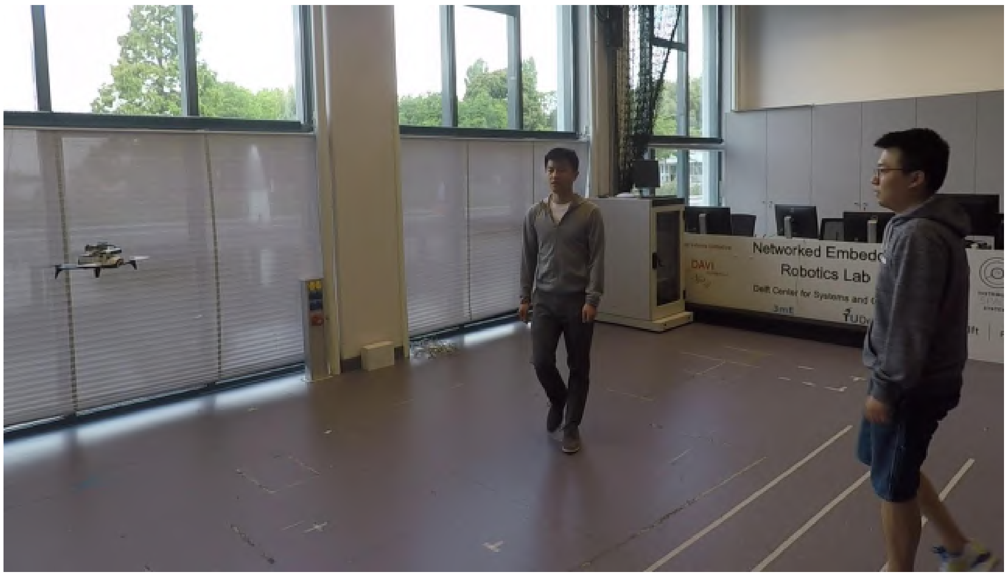


Figure 5-8: The snapshot of the experiment setup: MAV is hovering in front of two pedestrians.

A comparison is shown in Figure 5-9, as can be seen, when the obstacle is close to the camera, the estimation result is more accurate than the estimation at farther distances. This is because the depth measurement error generally increases quadratically with the depth [64]. Another finding is that, when an obstacle is just presented in the camera's view, the first few estimations are unreliable. Take the estimation starts at position $(-2, -1)$ as an example, in the beginning, the estimated velocity is pointing to the bottom-right while the actual velocity is pointing top-right. This is mainly caused by the changing size of the obstacle, which later results in an inaccurate estimation of the obstacle's center position.

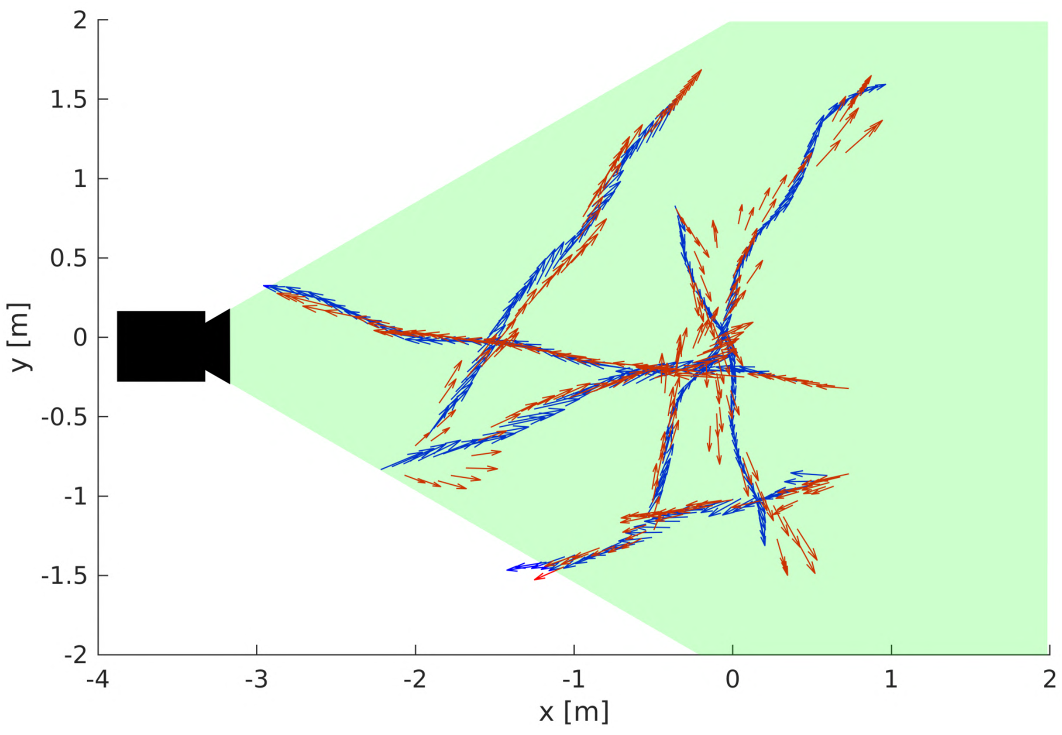


Figure 5-9: The comparison between the detected obstacles and their corresponding ground truth in top view. The red arrow represent the estimated velocity at the estimated position while the the blue arrow is the actual value measured by the motion capture system.

Table 5-1 shows the average position and velocity estimation error of the two pedestrians experiment, which calculated with the estimation and the ground truth. It can be observed that the average position error is around 0.26 m and that of the velocity is around 0.43 m/s, which indicates the obstacle sensing uncertainty should be taken into account when MAV is avoiding moving obstacles. However, the velocity estimation may be very noisy and has a very large uncertainty covariance. To this end, the uncertainty should be bounded when predicting the obstacles' future positions and corresponding uncertainty covariance matrices (Equation 5-18).

Table 5-1: The estimation error of the two pedestrians experiment.

Average Position Error (m)	**Average Velocity Error (m/s)**
0.26	0.43

5-3 Concluding Remarks

This chapter introduced the proposed obstacle sensing module which includes obstacle detection, estimation, and prediction. The proposed sensing module is computationally light and able to extract obstaclesâĂŸ information (such as their center positions, sizes) and estimate their motions assuming their linear velocities are constant. These properties are preferable for a real-time vision-based MAV system to avoid moving obstacles. Nevertheless, the estimation of obstacles' linear velocities suffers from uncertainties and this leads to our conclusion that the uncertainty covariance of the obstacles' positions and velocities should be taken into account to perform robust moving obstacle avoiding.

Chapter 6

Robust Trajectory Optimization

Different from other trajectory generation methods (such as reactive avoiding, spline-based and motion primitives), MPC solves the optimal trajectory taking the MAV's dynamics into account. With this property, the generated trajectory's dynamically feasibility is promised. In addition, the proposed planning method considers the uncertainties of the MAV state estimation and obstacle sensing, which makes the real-time vision-based system avoid obstacles robustly. In this chapter, the optimal trajectory generation is expressed as an optimization problem, and the designed of the MPC is introduced and tested.

6-1 MPC Formulation

The trajectory generation method solves a receding horizon constrained optimization problem at each time step. The optimization problem considers multiple objectives and constraints (which is computed with MAV states $\mathbf{x}^k = [\mathbf{p}^k, \mathbf{v}^k, \phi^k, \theta^k, \psi^k]^T \in \mathcal{X}$ and the control command $\mathbf{u}^k \in \mathcal{U}$ at time step k), over $N\Delta t$ prediction horizon, where Δt is the sampling time, as follows:

$$
\begin{aligned}
\min_{\mathbf{x}^{0:N}, \hat{\mathbf{u}}^{1:N-1}} \quad & \sum_{k=0}^{N-1} J^k(\mathbf{x}^k, \mathbf{u}^k) + J^N(\mathbf{x}^N) && \text{(6-1)} \\
\text{s.t.} \quad & \hat{\mathbf{x}}^0 = \mathbf{x}(t_0), && \text{(6-2)} \\
& \hat{\mathbf{x}}^k = \mathbf{f}(\hat{\mathbf{x}}^{k-1}, \hat{\mathbf{x}}^{k-1}), && \text{(6-3)} \\
& \mathbf{G}(\hat{\mathbf{x}}^k, \Gamma^k), && \text{(6-4)} \\
& \mathbf{u}^{k-1} \in \mathcal{U}, && \text{(6-5)} \\
& \mathbf{x}^k \in \mathcal{X}, && \text{(6-6)} \\
& \forall k \in \{1, \ldots, N\},
\end{aligned}
$$

where J^k denotes the cost term at time k and J^N denotes the terminal cost. $\mathbf{f}(\bullet)$ is the nonlinear dynamic model of the MAV, which is introduced and identified in Section A-4.

$\mathbf{G}(\bullet)$ is a function representing the state constraints, which are described in detail in the following. Γ^k is the MAV state uncertainty covariance at time k. We further denote Σ^k by the 3×3 covariance matrix of the MAV's position $\mathbf{p}^k$, extracted from Γ^k.

6-1-1 Cost Functions

The cost function of the optimization problem (Equation 6-1) consists of multiple objectives, including waypoint tracking, power save, collision avoidance, etc. In this section, the designed cost functions are introduced and simulated.

Waypoint Tracking Let $\mathbf{p}^g$ be the defined waypoint of the Micro Aerial Vehicle (MAV). The terminal cost term is used to minimize the distance between the terminal position and the defined waypoint in the prediction horizon. This cost is defined as:

$$J^N(\mathbf{x}^N) = ||\hat{\mathbf{p}}^N - \mathbf{p}^g||_{\mathbf{Q}_g} \tag{6-7}$$

where $\mathbf{Q}_g$ is a tuning weight coefficient.

The waypoint tracking performance is shown in Figure 6-1, as can be seen, the MAV keeps accelerating until it is close to the waypoint. This will certainly be detrimental to the MAV's obstacle avoiding performance especially when the waypoint is far from the starting point because the MAV's velocity will then be so high that the MAV can not react to obstacles.

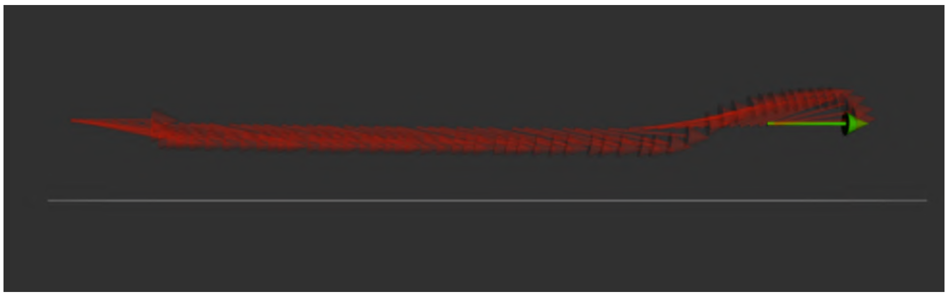


Figure 6-1: The simulation result of waypoint tracking performance (left view).

Velocity Cost To bound the MAV's velocity below a certain value, the sigmoid function, which can create a smooth step function at the defined maximum value v_{max}^k, is used instead of a simple box constraint. In addition to its smoothness, using the sigmoid function ensures the optimization problem is always feasible (using a simple box constraint may be infeasible due to the model inaccuracy, disturbances, estimation error, etc.) The velocity cost function is defined as:

$$J_v^k(\mathbf{x}^k) = \frac{\mathbf{Q}_v}{1 + exp(\frac{v_{max}^k - 1}{\lambda_v})}, \tag{6-8}$$

where $\mathbf{Q}_u$ is a tuning weight coefficient, λ_v is a smoothness factor.

Figure 6-2 shows the simulation result of the velocity control. Compared with the trajectory in Figure 6-1, the MAV will no longer fully pitch before it gets close to the waypoint. Note that the oscillation in the pitch angle can be eliminated by tuning $\mathbf{Q}_v$ and λ_v.

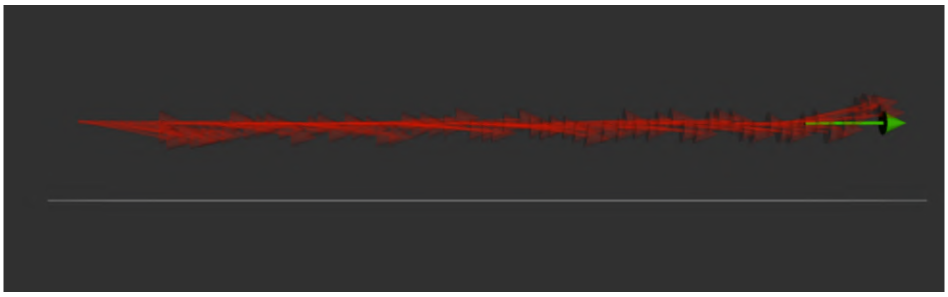


Figure 6-2: The simulation result of velocity control performance (left view).

Collision Cost To improve flight safety, a collision cost is introduced. Let d_o^k be the distance between the MAV and the obstacle o. And in order to have a smooth and bounded collision cost function, the sigmoid function is used in the collision cost term at each stage. The collision cost is expressed as:

$$J_o^k(\mathbf{x}^k) = \frac{\mathbf{Q}_o}{1 + exp(\lambda_o(d_o^k) - r_o)}, \tag{6-9}$$

where $\mathbf{Q}_o$ is a tuning weight coefficient, λ_o is the smoothness factor of the sigmoid function and r_o is a tuning threshold distance between the MAV and the obstacle where the collision cost is $\mathbf{Q}_o/2$.

Figure 6-3 shows the performance of the collision cost function in the simulation. The MAV is designed to navigate from left to the right, and an obstacle (marked by yellow) is placed on the halfway, it can be seen that the MAV successfully overtake the obstacle in the bottom side.

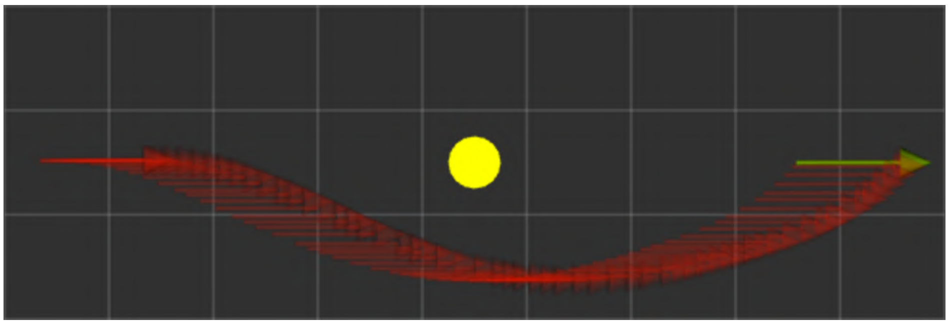


Figure 6-3: The simulation result of collision avoidance performance (top view).

MAV Yaw Control The purpose of this cost term is that, since the onboard camera has a limited field of view, the camera axis should be aligned with the direction of motion so that the free space along the direction of motion can be extracted. Instead of employing a velocity tracking yaw control method as in [65], which may generate infeasible yaw trajectories, a cost function that minimizes the deviation between the MAV's yaw angle and the motion direction is designed as:

$$J_\psi^k(\mathbf{x}^k) = \mathbf{Q}_\psi(\psi^k - \bar{\psi}^k)^2, \tag{6-10}$$

where $\mathbf{Q}_\psi$ is a tuning weight coefficient, $\bar{\psi}^k = arctan\frac{\hat{v}_y^k}{\hat{v}_x^k}$ indicates te MAV's motion direction angle. And in order to reduce the computational time, the desired yaw angle $\bar{\psi}^k$ is computed based on the last optimized trajectory.

A simulated trajectory is shown in Figure 6-4.

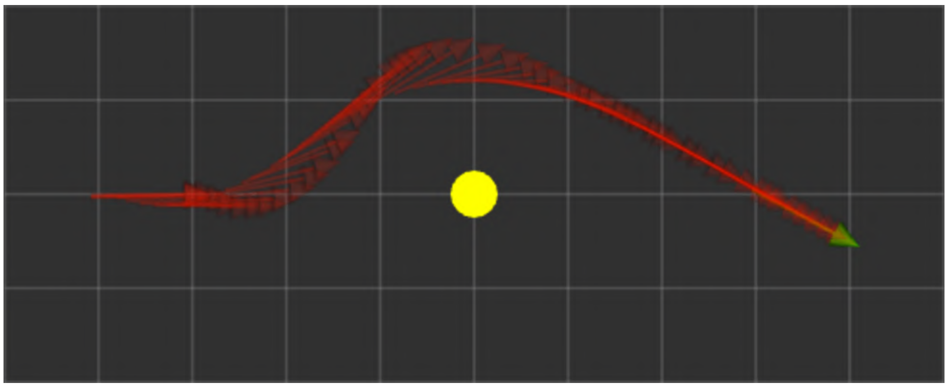


Figure 6-4: The simulation result of MAV yaw control performance (top view).

Control Input Cost In order to save the battery power and extend the flight time, the MAV control input is minimized at each stage with:

$$J_u^k(\mathbf{u}^k) = ||\mathbf{u}^k||_{\mathbf{Q}_u}, \tag{6-11}$$

where $\mathbf{Q}_u$ is a tuning weight coefficient.

Finally, the overall stage cost function of the formulated optimization is

$$J^k(\mathbf{x}^k, \mathbf{u}^k) = J^N(\mathbf{x}^N) + J_v^k(\mathbf{x}^k) + J_u^k(\mathbf{u}^k) + J_o^k(\mathbf{x}^k) + J_\psi^k(\mathbf{x}^k). \tag{6-12}$$

6-1-2 Constraints

Besides of the aforementioned cost functions, the constraints (Equation 6-4) below are introduced to improve flight safety.

Collision Chance Constraints As the detected obstacles are represented by bounding boxes in Section 5-1-3, which is not computationally efficient for collision checking when compared with using ellipsoids. The obstacles are modeled as ellipsoids with their positions $\mathbf{p}_o^k$, orientations R_o^k and semi-principal axes $(a_o^k, b_o^k, c_o^k) = \frac{\sqrt{3}}{2}(l_o, w_o, h_o)$. And the MAV is modeled as a sphere with position $\mathbf{p}^k$ and radius r. The collision chance constraints is then defined with the obstacle's information as well as the MAV's information to perform the collision checking.

$$C_o^k : (\mathbf{p}^k - \mathbf{p}_o^k)^T \mathbf{\Omega}_o^k (\mathbf{p}^k - \mathbf{p}_o^k) \leq 1, \tag{6-13}$$

where $\mathbf{\Omega}_o^k = R_o^{k,T} diag\left(\frac{1}{a_o^k+r}, \frac{1}{b_o^k+r} \frac{1}{c_o^k+r}\right) R_o^k$.

And as discussed in Chapter 4 and Chapter 5, the MAV state estimation and the obstacle sensing suffer from inaccurate estimation, their uncertainty need to be taken into account. Thus, the collision avoidance constraints would be satisfied in a probabilistic manner, which are formulated as chance constraints:

$$Pr(C_o^k) \leq \delta \tag{6-14}$$

where δ is the probability threshold for the collision.

With the assumption that the $\mathbf{p}^k \sim \mathcal{N}(\hat{\mathbf{p}}^k, \mathbf{\Sigma}^k)$ and $\mathbf{p}^k \sim \mathcal{N}(\hat{\mathbf{p}}^k, \mathbf{\Sigma}^k)$ are Gaussian distributions (obtained from the state estimator and the obstacle sensing module), Equation 6-14 can be transformed into a deterministic constraint based on [31]:

$$\mathbf{n}_o^{k^T} \mathbf{\Omega}_o^{k^{\frac{1}{2}}} (\mathbf{p}^k - \mathbf{p}_o^k) - 1 \geq \text{erf}^{-1}(1-2\delta) \cdot \sqrt{2\mathbf{n}_o^{k^T} \mathbf{\Omega}_o^{k^{\frac{1}{2}}} (\mathbf{\Sigma}^k + \mathbf{\Sigma}_o^k) \mathbf{\Omega}_o^{k^{\frac{1}{2}}} \mathbf{n}_o^k}, \tag{6-15}$$

where $\mathbf{n}_o^k = (\mathbf{p}^k - \mathbf{p}_o^k)/||\mathbf{p}^k - \mathbf{p}_o^k||$, and $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ is the standard error function for the normal distribution.

FOV Constraints In order to further improve flight safety, the generated trajectory should be within its current limited field of view. Known the MAV current pose, its three-dimensional FOV is modeled by an intersection of five half-spaces:

$$FOV^k := \{\mathbf{p} | \mathbf{n}_j^k \mathbf{p} \leq m_j^k\}, \quad j = 1, \cdots, 5, \tag{6-16}$$

where $\mathbf{n}_j^k$ and m_j^k are parameters of the half-spaces computed with camera intrinsic parameters. Then the FOV constraints are expressed as:

$$\mathbf{p}^k \in FOV^k \tag{6-17}$$

6-2 Uncertainty Propagation

In order to evaluate the collision chance constraints in Equation 6-15, the MAV's position and its corresponding uncertainty covariance matrix at each time step are required. However, using a high-precision uncertainty propagation for nonlinear systems (for example the nonlinear dynamic model used in Section A-4) is computationally heavy. And for the sake of the system's real-time performance, an approximate propagation based on Extended Kalman Filter (EKF) is used to update the MAV position uncertainty.

$$\mathbf{\Gamma}^{k+1} = \mathbf{F}^k \mathbf{\Gamma}^k \mathbf{F}^{k^T} + \mathbf{W}^k, \tag{6-18}$$

where $\mathbf{W}^k$ is the process noise accounting for motion disturbances, $\mathbf{F}^k = \frac{\partial \mathbf{f}^k}{\partial \mathbf{x}}|_{\hat{\mathbf{x}}^k, \mathbf{u}^k}$ is the state transition matrix of the MAV. The position uncertainty covariance $\mathbf{\Sigma}^k$ can be then extracted from $\mathbf{\Gamma}^k$.

Note that the computation of $\mathbf{F}^k$ correlates the MAV states and control commands, which will introduce additional variables in the optimization problem (Equation 6-1) and dramatically increase the computational time. To tackle this problem, the MAV state uncertainty covariance is propagated with the control inputs computed from the last-loop optimization.

6-3 Simulations

To evaluate the proposed chance-constrained MPC, the robust trajectory optimization method was tested in 12×12 m^2 random forests (the simulation environment is introduced in Chapter B). The cylindrical obstacles' positions $\mathbf{p}_o$ and their horizontal radius were defined randomly, while their velocities were kept as zero and their heights were set as infinity. For their uncertainties, both $\mathbf{\Sigma_o}$ and $\mathbf{\Sigma_{o,v}}$ were set as $diag(0.01)$. In order to perform real-time MPC, the ACADO toolkit [26] was used to generate a fast C solver for the trajectory optimization problem. And for the sake of computational efficiency, only the two closest obstacles in MAV's field of view were fed to the MPC in practice. As shown in the simulation result (Figure 6-5), the MAV can navigate safely within the random forest from the left to the right, which illustrates that the proposed trajectory planning method can generate collision-free trajectories in such cluttered environments.

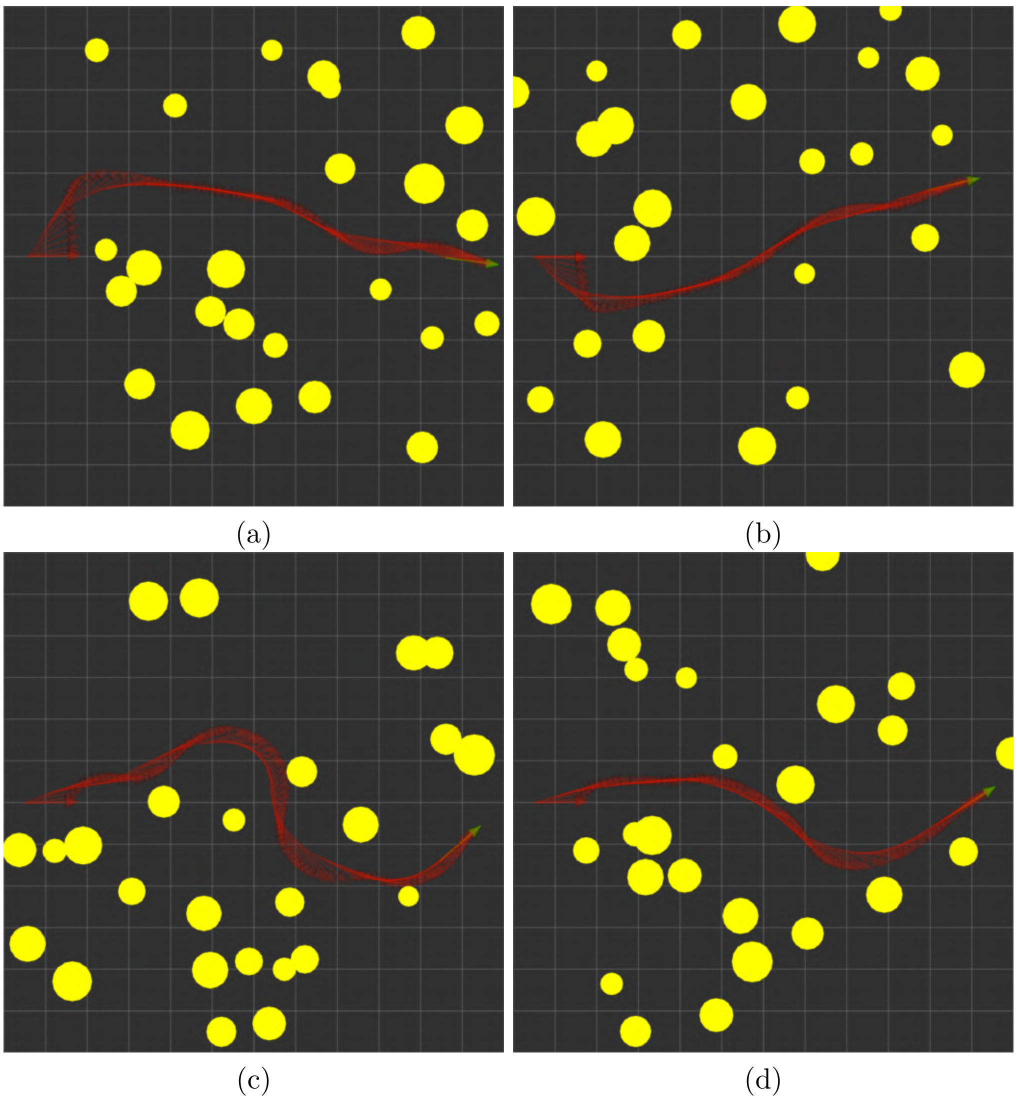


Figure 6-5: Four optimal collision trajectories (top view) planned in random forest simulation. Cylindrical obstacles are marked by yellow while the red arrow represent the past MAV pose.

6-4 Concluding Remarks

In this chapter, the robust trajectory optimization problem is formulated as a MPC whose cost functions and constraints are well designed to enable the MAV system to navigate autonomously and safely in dynamic environments with imprecise obstacle sensing result. From the obstacle avoiding performance in the simulation, the proposed method is computationally lightweight and successfully navigate within cluttered environments with desired behaviors (for example MAV yaw angle control, obstacle avoiding, etc.)

Chapter 7

Robust Obstacle Avoiding Experiments

In this chapter, the proposed MAV autonomous system was tested in two different typical scenarios. In the first scenario, the MAV was required to navigate from a defined starting point to the given waypoint within a lab space, where the motion capture system is installed. The obstacles presented in the lab space and the MAV were equipped with reflective markers so that their poses could be measured by the motion capture system as ground truth data. In the second scenario, the MAV was required to fly in a long narrow corridor to test the obstacle avoiding performance in long flights.

7-1 Collision Avoidance of Static Obstacles

The first experiment is to test the MAV's obstacle avoidance performance within static environments. In this experiment, multiple static obstacles are places in the experimental fields.

7-1-1 Scenario 1

In the first scenario, two static obstacles were placed between the defined starting point and the given waypoint. To obtain the ground truth data and evaluate the performance of the obstacle avoidance, the sizes of the two obstacles were measured manually, and their poses were measured by the motion capture system. Figure 7-1 shows a sequence of snapshots and the onboard grayscale images captured during the experiment. As can be seen, when the first static obstacle was presented, the MAV rolled to its right and then overtook it. When the MAV detected the second static obstacle, it then rolled to its left in order to accelerate in the y-direction and avoided the second obstacle. After it avoided the two static obstacles, it decelerates in the y-direction to track the waypoint. Finally, the MAV reached and hovered at the given waypoint. Figure 7-2 shows the measured distance between the MAV and the obstacles' bounding ellipsoids over time in the corresponding run of Figure 7-1, from which the conclusion can be drawn that the real-time vision-based autonomous MAV system can navigate safely in the first scenario.

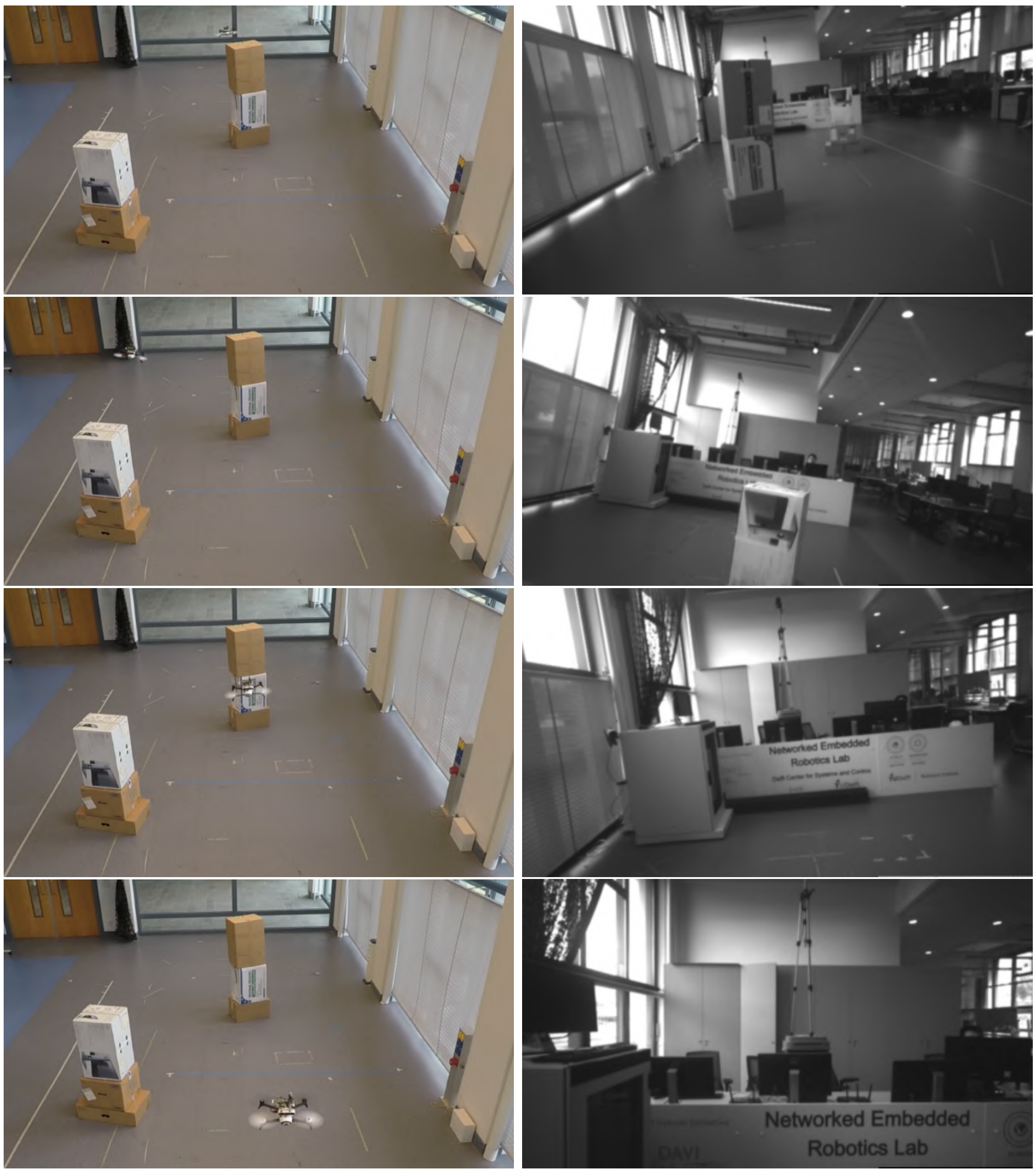


Figure 7-1: A sequence of images (from the top to the bottom) during the experiment in scenario 1 with static obstacles. The snapshot (left) is captured by the external camera, while the onboard grayscale image is taken by the MAV onboard camera.

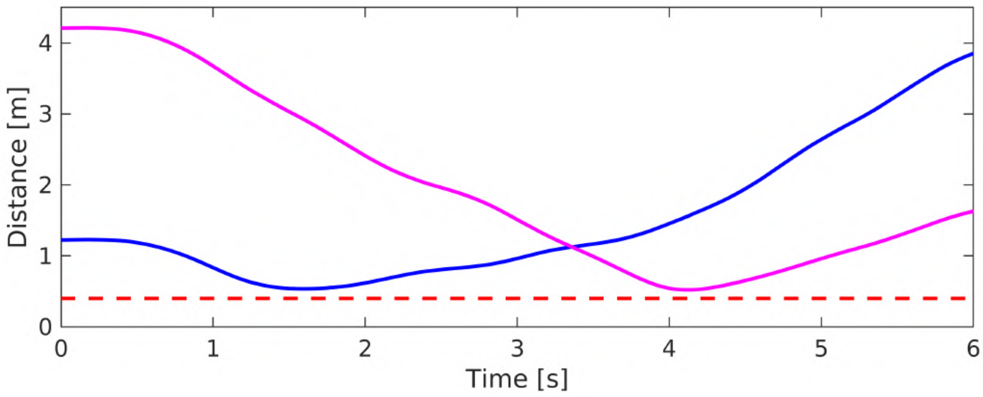
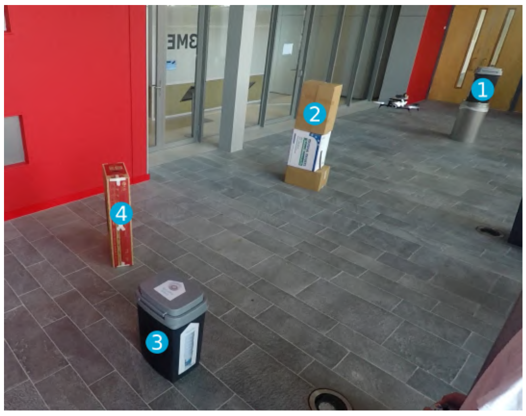


Figure 7-2: The distance between the MAV and the two static bounding ellipsoids (magenta and blue) overtime. The red dash line represent the radius of the MAV's bounding sphere.

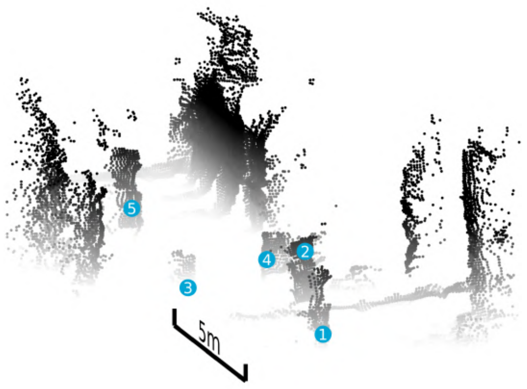
7-1-2 Scenario 2

In the second scenario, the MAV was required to navigate along a long narrow corridor, where five static obstacles were placed. Figure 7-3(a) shows a snapshot of the experiment in the long narrow corridor. In Figure 7-3(b), the reconstructed map of the experimental field is shown, the map is reconstructed from the depth images during the flight. And the result is shown in Figure 7-4, which shows that the MAV's trajectory as well as a series of the onboard grayscale images captured during the flight. As can be seen, the MAV successfully navigated from the defined starting point to the given waypoint without any collision.

Note that the MAV detected the third obstacle and avoided it by flying over its top.

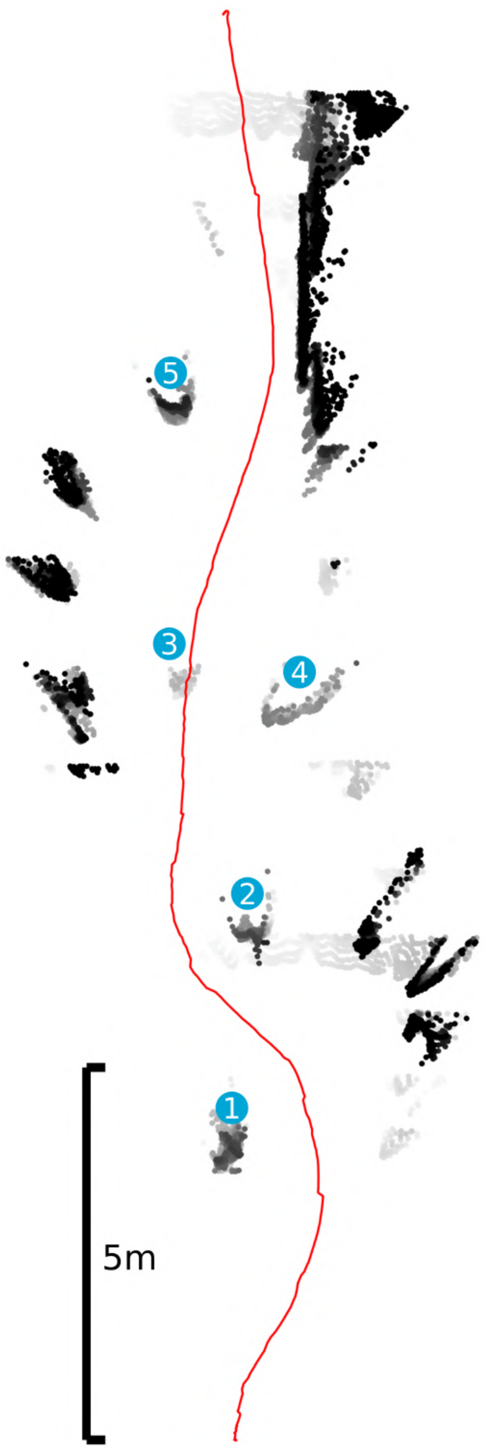


(a)



(b)

Figure 7-3: The MAV is required to navigate along a long narrow corridor in scenario 2. (a) A snapshot of the collision-avoidance of static obstacles in scenario 2. (b) A three-dimensional map of the experimental field reconstructed from the onboard depth measurement and the estimated odometry. Points are marked with colors according to their heights.



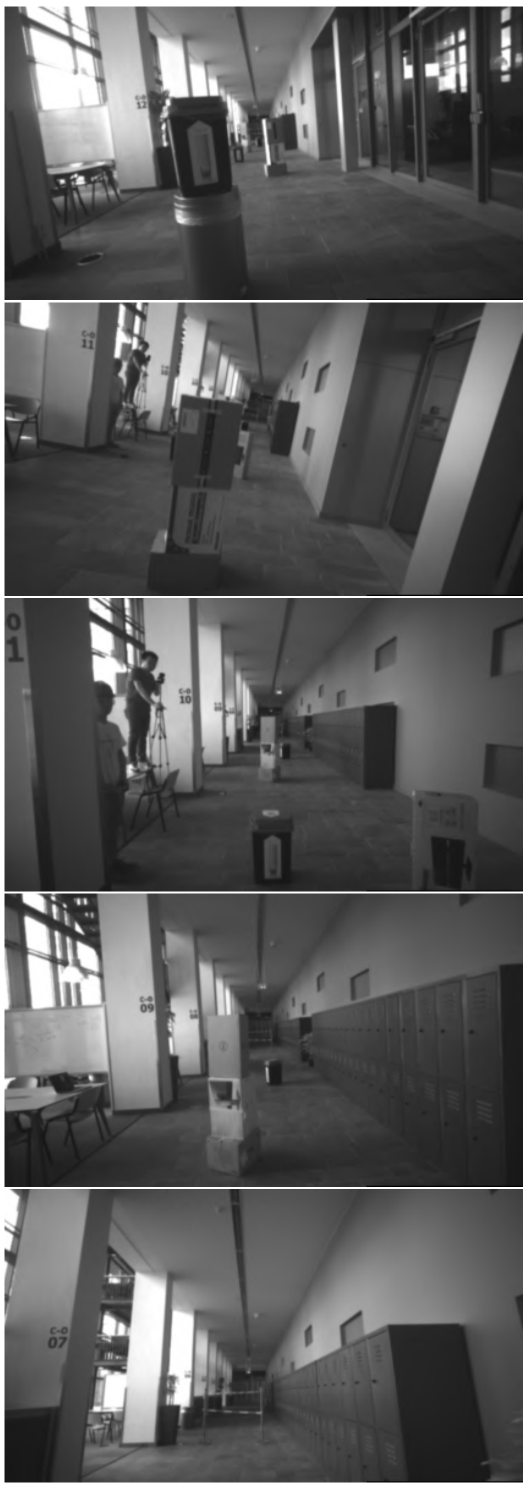


Figure 7-4: Left: the top view of the estimated MAV's trajectory during the experiment. Right: a sequence of the onboard grayscale images (from the top to the bottom) during the experiment in scenario 2 with static obstacles.

7-2 Collision Avoidance of Dynamic Obstacles

To test the autonomous MAV system's obstacle avoidance performance in dynamic environments, two pedestrians were presented in the experimental field.

7-2-1 Scenario 1

In the first scenario, the MAV was required to navigate from a starting point to an given waypoint while avoiding two walking humans. Motion capture system was used to measure the actual poses of the MAV and the two pedestrians. The pedestrians were modeled as ellipsoids whose semi-major axes were $(0.4, 0.4, 0.9)$ m, while the MAV was modeled as a sphere with radius $r = 0.4$ m. In Figure 7-5, the distance from the MAV's center to the two ellipsoids' surfaces over time, which was measured by the motion capture system, in 6 different runs are shown. It can be observed that in all instances a minimum safe separation of 0.4 m was achieved and therefore collision with the pedestrians were avoided.

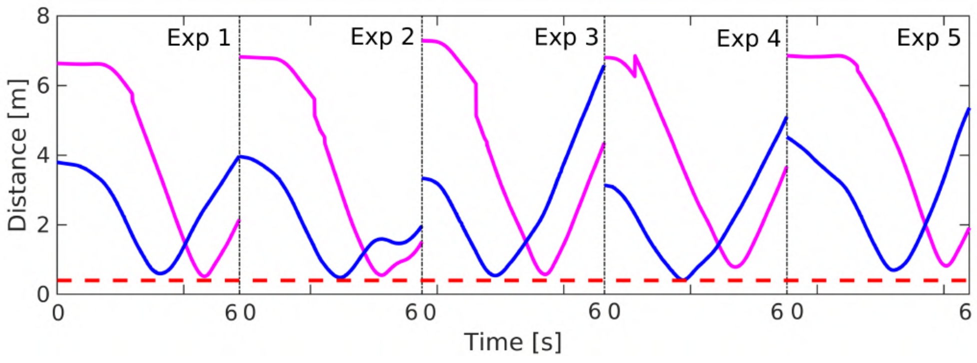


Figure 7-5: The distance between the MAV and the two static bounding ellipsoids (magenta and blue) overtime. The red dash line represent the radius of the MAV's bounding sphere.

And a histogram of the measured distance is shown in Figure 7-6.

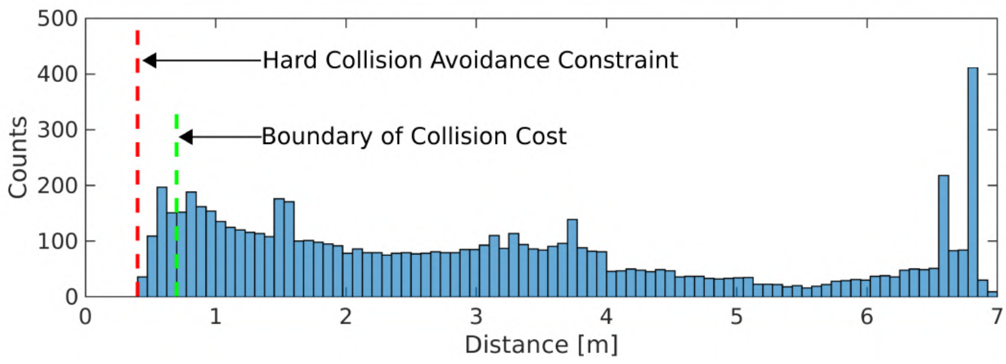


Figure 7-6: The accumulation of the measured distance. The red dash line represent the radius of the MAV's bounding sphere and the green dash line is the defined boundary of the collision cost (Section 6-1-1).

Figure 7-7 shows a sequence of snapshots and the onboard grayscale images captured during the experiment (the third experiment in Figure 7-5).

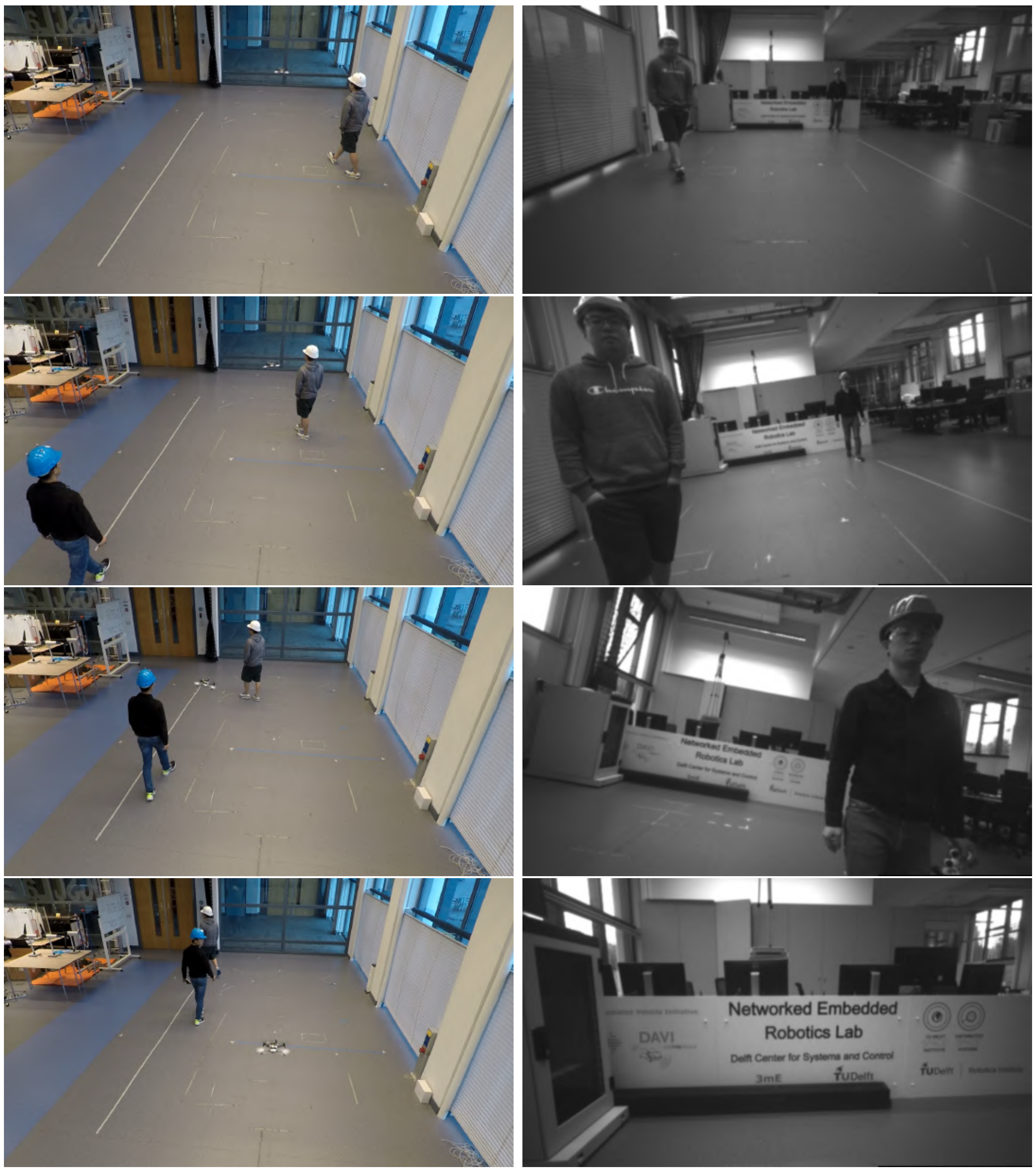


Figure 7-7: A sequence of images (from the top to the bottom) during the experiment in scenario 1 with two pedestrians. The snapshot (left) is captured by the external camera, while the onboard grayscale image (right) is taken by the MAV onboard camera.

7-2-2 Scenario 2

In the experiment with dynamic obstacles in the second scenario, the MAV successfully navigated along the long narrow corridor, where two static obstacles and two pedestrians were presented. Figure 7-8 shows a series of onboard grayscale images captured by the onboard camera and their corresponding visualization during the experiment. As can be seen from the visualization, all static and moving obstacles were detected and all the generated optimal trajectories were collision-free.

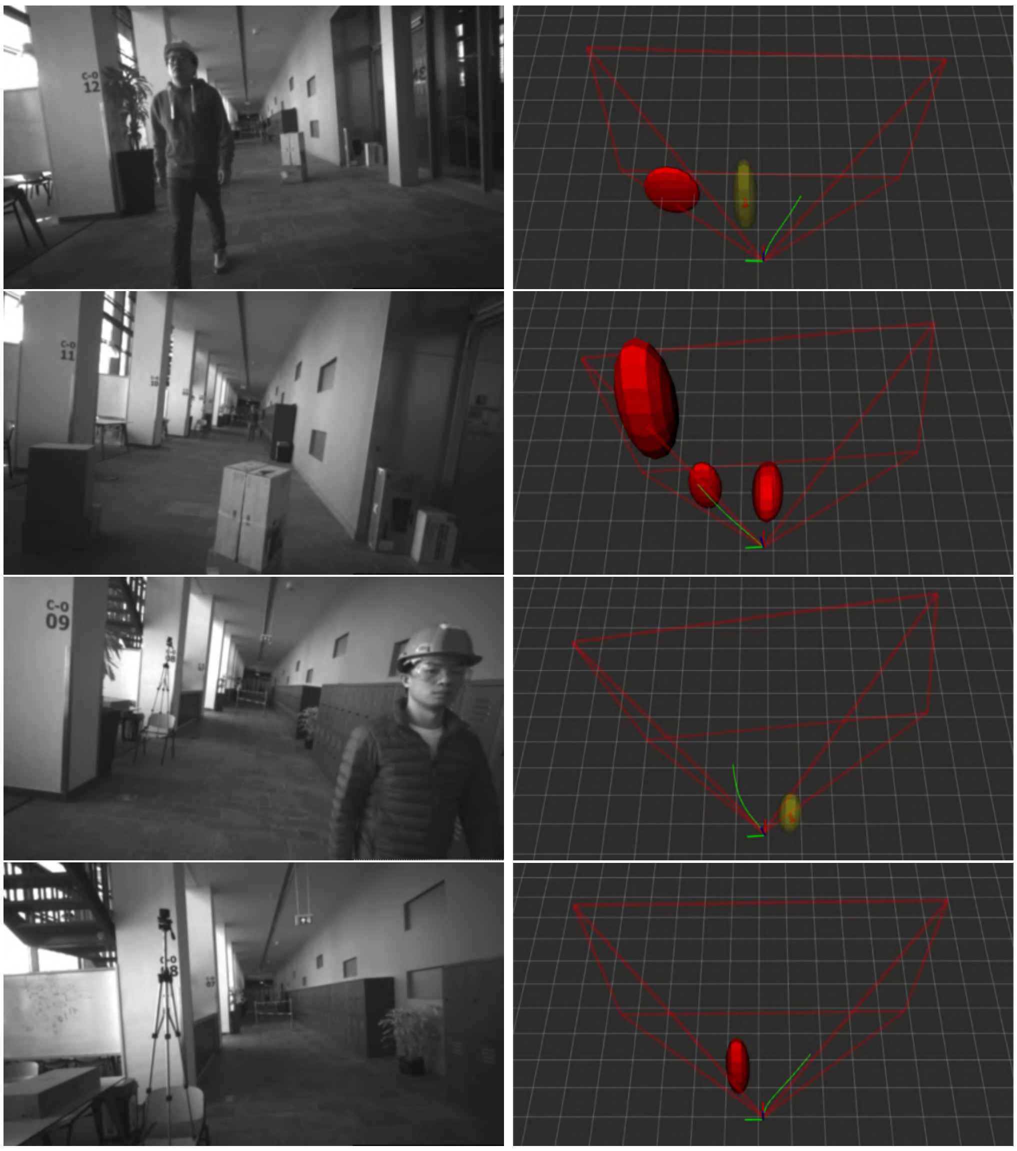


Figure 7-8: A sequence of images (from the top to the bottom) captured during the experiment in scenario 2 with two pedestrians and two static obstacles. And their corresponding obstacle sensing results are visualized. In the visualization, moving obstacles are marked by yellow while static obstacles are marked by red. And the green line represent the generated optimal trajectory.

7-3 Real-Time Performance

In addition to the obstacle avoidance performance, the real-time performance of the proposed MAV system was evaluated. Figure 7-9 shows the onboard runtimes of MAV state estimation, obstacle sensing and trajectory optimization. As can be seen, for the runtimes of the MAV state estimation, the 75^{th} percentile was always below 30 ms, which could be run at a rate of 15 Hz. For the obstacle detection and tracking, the 75^{th} percentile was always below 8 ms, which was fast enough to be run at 60 Hz. And for the trajectory optimization framework, the 75^{th} percentile was always below 22 ms, which indicateed that it could be performed efficiently in real-time.

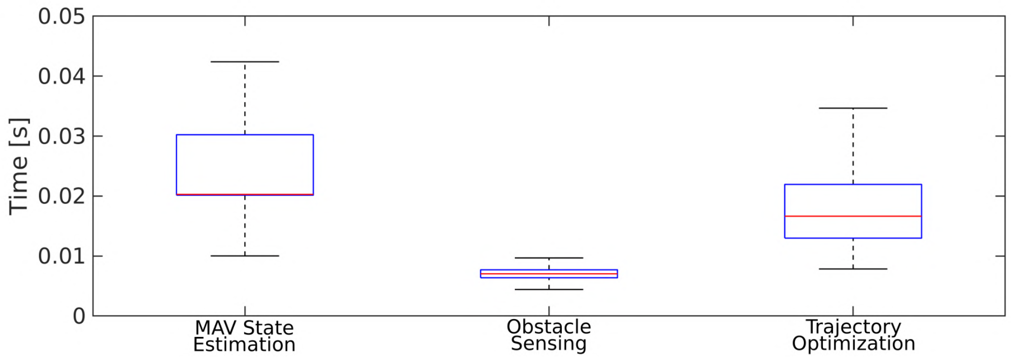


Figure 7-9: The computation time of different MAV state estimation, obstacle sensing and trajectory optimization.

7-4 Concluding Remarks

In this chapter, the autonomous MAV system was tested in different scenarios with static and moving obstacles. The experiment result shows that the MAV successfully navigated from the defined starting point to the given waypoint without any collisions, which indicates that the robust trajectory optimization method used in this thesis work was sufficiently robust. Besides, the computation time of each onboard module during the experiments suggests that the method was computationally efficient to be performed in real-time.

Chapter 8

Conclusions

This chapter summaries this thesis work and presents some guidelines for future improvements.

8-1 Summary

In this thesis work, a MAV system was built for real-time vision-based autonomous navigation in dynamic environments. The starting point was a study of past relative works on vision-based MAV autonomous navigation. As previous approaches were generally based on the assumption that the environment is static, a research question was then drawn: is it possible to build a real-time vision-based MAV system to navigate autonomously and safely in dynamic environments with onboard sensing and computing?

From the preliminary study of the state-of-art, approaches such as [3, 29] showed that by using the model predictive control, MAVs were able to react to dynamic objects (such as object tracking and obstacle avoidance). Based on this idea, a minimal MAV system was proposed, which consisted of state estimation, obstacle sensing, and trajectory optimization. And in order to perform these modules, an experimental platform was designed and built, which was constructed with a quadrotor, a depth camera, and a single-board computer.

The main challenge of building such an autonomous MAV system was that the computational modules have to be computationally efficient so that they could be performed on the power-limited onboard computer. For the state estimation, several state-of-art visual odometry and simultaneous localization and mapping algorithms were tested and evaluated on the experimental platform in order to select an accurate and computationally light approach. S-MSCKF was selected since its relative pose error was comparable with other algorithms while its average processing time was the lowest (26.9 ms). For the obstacle sensing, obstacles were detected with a fast obstacle detection method, which was extended from [11], and their linear velocities were estimated by a Kalman filter. The obstacle sensing module was so computationally light (has a mean computation time of 8 ms) that could be run at a rate of 60 Hz. However, the velocity estimation was very noisy and had a very large uncertainty covariance, which indicated the obstacle sensing uncertainty should be taken into account when the MAV was avoiding moving obstacles. To this end, a robust trajectory optimization method based on [31] was implemented on the experimental platform and was run at a rate of 15 Hz. Finally, to evaluated the proposed system's performance, flight tests were performed in a variety of environments with both static and moving obstacles. The result showed that the obstacle avoidance was robust and computationally efficient.

To conclude, the MAV system constructed in this thesis work was able to navigate autonomously and safely in dynamic environments with only the onboard sensing and computing.

8-2 Limitations and Recommendations

It must be admitted that the autonomous MAV can be further improved. Firstly, the obstacle sensing module models all the obstacles as bounding ellipsoids in order to estimate their movements, while in practice, this type of obstacle representation is too conservative. To this end, the detected static obstacles can be represented with occupancy cells (an example is the OctoMap [59]). Secondly, since the local trajectory planner used in this thesis work only optimizes a trajectory within a short-time prediction horizon, the MAV may get stuck by livelock. To avoid this, a global planner is needed so that a set of waypoints can be automatically extracted and the MAV can then navigate in more complex unknown environments. In addition to the software components, the hardware selection can be improved as well. During the experiments, flight instability was observed when the MAV was flying over obstacles' top. This is mainly caused by the inner controller of Parrot Bebop2, which always ensures the ranging data, measured by its bottom ultrasonic sensor, larger than a threshold value. Thus, another SDK (for example Paparazzi[1]) or another drone kit would be preferable.

[1]https://wiki.paparazziuav.org/wiki/Main_Page

Appendix A

System Identifications and Calibrations

In this section, several identifications and calibrations are performed. For example, the identification of IMU noise parameters gives a hint on how accurate the onboard inertial information is. The calibration of the visual-inertial sensor provides not only the relative transformation between camera frame and IMU frame but also time offset between the two sensors. These information is obviously necessary for fusing the visual-inertial information. Besides, when comparing the ground truth data (measured by the Motion Capture System (MOCAP)) with the estimation result, the relative transformation between the camera and MOCAP markers should be extracted by the calibration of camera-MOCAP system. This process is essential especially when evaluating the onboard obstacle detection as it is necessary to know the actual obstacles' positions relate to the camera. Finally, the model the whole MAV system is identified, which will be used for MAV control.

A-1 IMU Noise Model Identification

The onboard IMU sensor consists of an accelerometer and a gyroscope, which measures the linear acceleration $\tilde{a}$ and the rotational velocity $\tilde{\omega}$ respectively. In practice, these two measurements suffer from noises, which is commonly modeled as:

$$\begin{aligned} \tilde{a}[k] &= a[k] + b_a[k] + n_a[k] \\ \tilde{\omega}[k] &= \omega[k] + b_\omega[k] + n_\omega[k] \end{aligned} \tag{A-1}$$

where n_a and n_ω are white noises while b_a and b_ω are random walk noises. In the discrete-time signal, these white noises and random walk noises are expressed as:

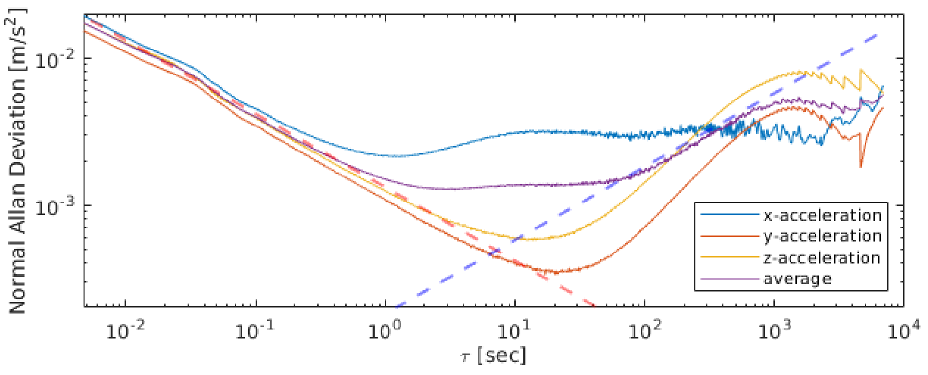
$$\begin{aligned} n[k] &= \sigma_n w[k] \\ b[k] &= b[k-1] + \sigma_b w[k] \end{aligned} \tag{A-2}$$

where $w[k] \sim \mathcal{N}(0,1)$ is the standard Gaussian noise, σ_n and σ_b is the strength of noise.

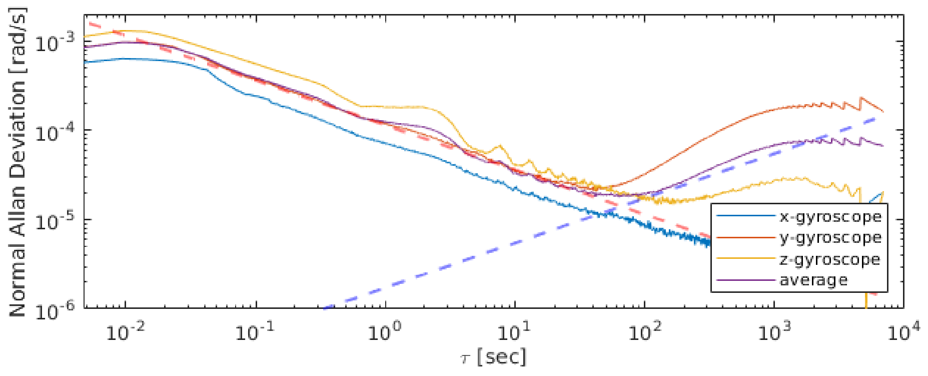
Allan variance analysis [66] is then used to determine these noise parameters of the onboard accelerometer and gyroscope (Bosch BMI055[1] is integrated in the Intel RealSense D435i). The

[1]https://www.bosch-sensortec.com/bst/products/all_products/bmi055

working principle of Allan variance analysis can be simply explained: It calculates the root Allan variance of the accelerometer and gyroscope measurements in time domain as a function of different averaging times τ and then it extracts the noise parameter from the log-log scale slopes of the Allan deviation curves. When the stationary measurement data is averaged over short periods, the Allan variance is high because of the white noise. By averaging the same data over longer periods, the Allan variance decrease since the white noise is averaged out. When the averaging period is even longer, it increases again due to the random walk noise. An example of the Allan deviation curves of the onboard accelerometer and gyroscope is shown in Figure A-1, the Allan deviation curves are extracted from a stationary data whose duration is approximately 4 hours.



(a)



(b)

Figure A-1: The Allan variance of the onboard (a) accelerometer and (b) gyroscope. The red and blue dashed lines are the fitted straight lines for white noise σ_n and random walk σ_b respectively.

And the noise parameter is then extracted from the Allan deviation curves. Given by the standard specification format [67]. For the white noise σ_n, it is extracted by fitting a straight line (red dashed lines in Figure A-1) in the region where the log-log slope is $-1/2$ in the Allan deviation curves, and its value is read at $\tau = 1$. And as for the random walk σ_b, the straight line (blue dashed lines in Figure A-1) is fitted in the 1/2 slope region in the Allan deviation curves and the value is read at $\tau = 3$. The identification result of the onboard IMU is listed in Table A-1.

Table A-1: The identified noise parameter of the onboard accelerometer and gyroscope.

Accelerometer		Gyroscope	
σ_n	σ_b	σ_n	σ_b
$\left(m/\left(s^2\sqrt{Hz}\right)\right)$	$\left(m/\left(s^3\sqrt{Hz}\right)\right)$	$\left(rad/\left(s\sqrt{Hz}\right)\right)$	$\left(rad/\left(s^2\sqrt{Hz}\right)\right)$
1.317×10^{-3}	3.163×10^{-4}	1.143×10^{-4}	3.013×10^{-6}

A-2 Camera-IMU Calibration

Although the transformation between the onboard IMU and the global shutter stereo camera is given by the manufacturer, however, the given transformation is inaccurate. Thus, a further calibration is performed in order to provide a more accurate transformation in order to fuse the inertial information with the visual measurement. Besides of estimating the transformation between the IMU and the onboard camera, the estimation of time offset between the two sensors is also necessary. In this thesis work, the visual-inertial temporal and spatial calibration is performed by following a well-known method [68]. Figure A-2 shows the calibration setup, and for the sake of simplicity, only the right camera is considered.

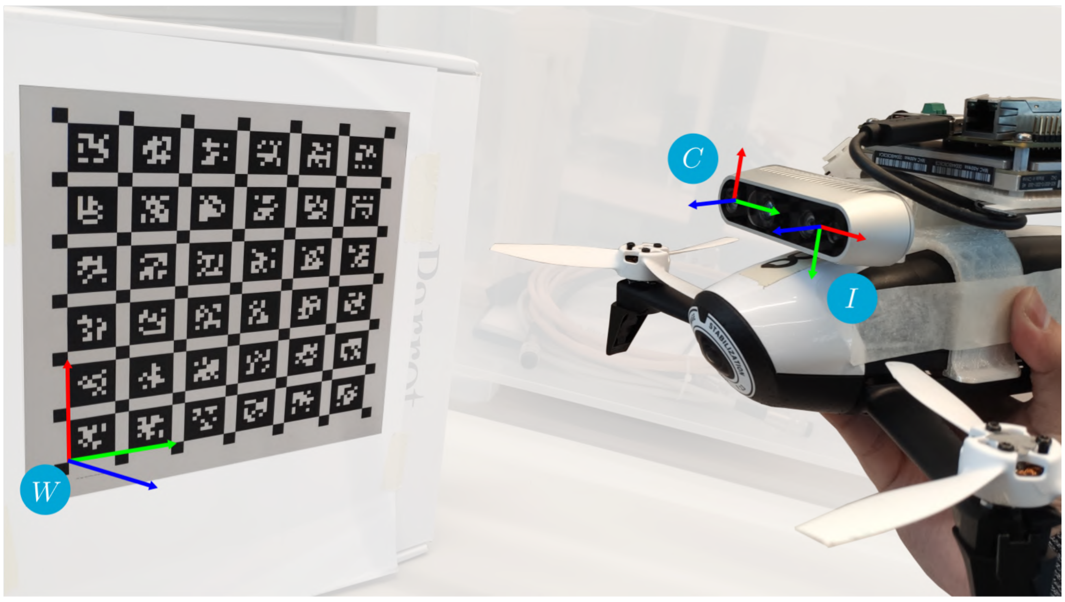


Figure A-2: The setup of the visual-inertial temporal and spatial calibration consists of a calibration pattern and a visual-inertial sensor. W is the world frame which is attached to the static calibration pattern[1], while C and I represent the right camera and the IMU frame respectively.

In order to perform the calibration, a short period (about 2 minutes) of data, in which all axis of camera are excited, should be logged. Then the recorded IMU pose is expressed as a 6×1 sixth-order B-spline function, with three degrees of freedom for position and three for orientation. By using the sixth-order B-spline function, the dynamic motion of the

[1]https://github.com/ethz-asl/kalibr/wiki/downloads

sensor is accurately captured as continuous-time signals and the corresponding IMU's linear velocity, acceleration and angular velocity can be simply expressed. Besides of these time-varying states, several time-invariant parameters also need to be estimated, which includes the gravity direction in world frame g_W, the transformation between visual and inertial sensors T_I^C and the time offset between them d_I^C. Finally, the estimated time-varying states and time-invariant parameters are extracted by using Levenberg-Marquardt (LM) algorithm [69] to minimize several cost terms including the reprojection error, the measurement error of accelerometer and gyroscope (Please see [68] for more details).

The calibration result of the onboard visual-inertial sensor can be found in Table A-2.

Table A-2: The calibration result of the onboard visual-inertial sensor.

Camera		Left	Right
Focal Length	f_x	212.13645	212.30128
	f_y	211.83443	211.99693
Principal Point	p_u	212.85793	213.30177
	p_v	119.22672	119.82929
Distortion Coefficients	k_1	0.00182	0.00169
	k_2	-0.00354	-0.00604
	r_1	0.00023	0.00013
	r_2	0.00034	0.00075
Translation (mm)	t_x	-3.52426	-53.29678
	t_y	-4.67344	-4.59911
	t_z	-6.04857	-6.06939
Rotational Matrix		$\begin{bmatrix} -0.0029 & -0.9999 & 0.0064 \\ -0.0033 & -0.0064 & -0.9999 \\ 0.9999 & -0.0029 & -0.0033 \end{bmatrix}$	$\begin{bmatrix} -0.0027 & -0.9999 & 0.0064 \\ -0.0033 & -0.0064 & -0.9999 \\ 0.9999 & -0.0029 & -0.0033 \end{bmatrix}$
TimeShift (ms)		6.81776	6.79232

Note that these relative transformations are expressed in the IMU frame, the pinhole camera model and the radial-tangential model are used as camera model and distortion model respectively.

A-3 Camera-MOCAP Calibration

The external measurement from the Motion Capture System (MOCAP) is millimeter accurate and is used as ground truth data for the evaluation of vision-based state estimation and obstacle sensing. However, MOCAP measures the position and orientation of the reflective markers instead of camera's. Thus, it is necessary to develop a method to estimate the relative transformation between the markers and the camera T_M^C. The main idea of the proposed method is estimating the relative transformation by adding an extra landmark. The landmark should be well designed so that the landmark frame can be measured accurately by the MOCAP as well as the camera. The relative transformation can be then extracted with the transformation between the landmark and the camera's marker T_L^M and the transformation between the landmark and the camera center T_L^C:

$$T_M^C = \left(T_L^M\right)^{-1} T_L^C \tag{A-3}$$

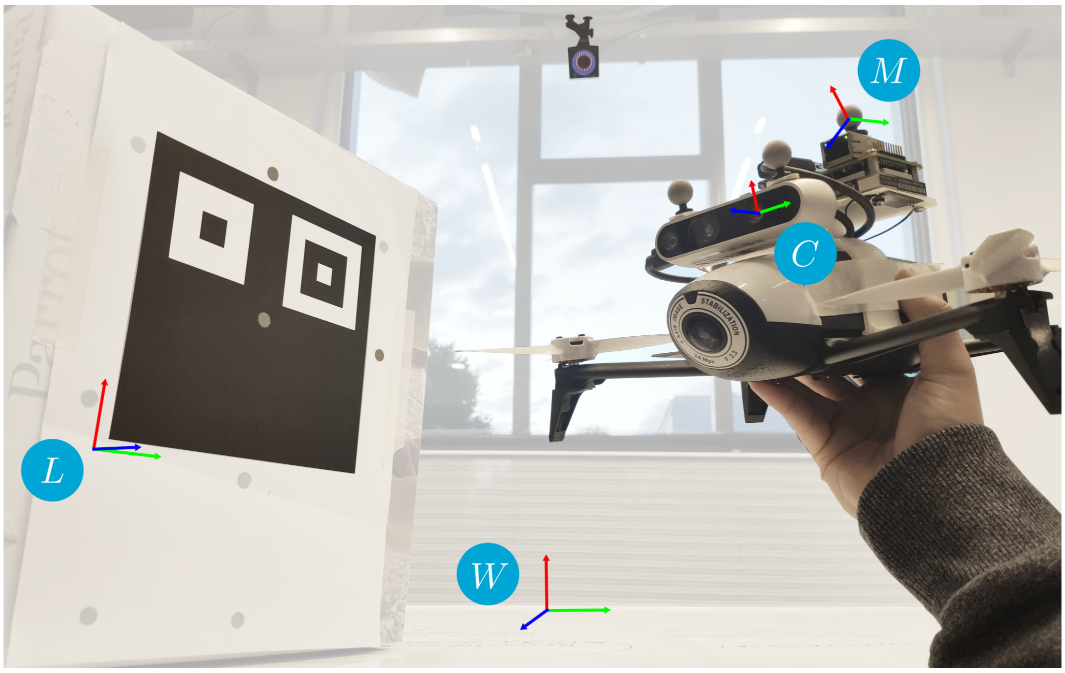


Figure A-3: The setup of camera-MOCAP calibration consists of a MOCAP system, a landmark which can be detected by MOCAP and camera, a camera with reflective markers. W is the world frame, which is consistent with the MOCAP coordination, while L is the frame of the additional landmark, C is the camera frame and M is the frame of the reflective marker which is attached to the camera.

For the MOCAP measurement, the position of the landmark in the world frame t_W^L is obtained by measuring the position of the adhesive marker at the center of the landmark. And its orientation R_W^L is obtained by calculating the normalized cross product of the vector from central adhesive marker to the right one and the vector from central adhesive marker to the top one (the placement of adhesive markers can be seen from Figure A-3). Then the relative transformation $\widetilde{T}_L^M$ can be computed by:

$$\widetilde{T}_L^M = \left(\begin{bmatrix} R_W^L & t_W^L \\ 0 & 1 \end{bmatrix}\right)^{-1} T_W^M \tag{A-4}$$

As for the visual measurement, the calibration pattern on the landmark is detected by image processing and the feature point on the calibration pattern is then extracted. Known these feature points' placement in the real setup, the relative transformation between the landmark and the camera T_L^C can then be estimated by using the Perspective-n-Point (PNP) algorithm

[70]. Since the camera is run at a rate of 15 Hz, the estimated transformation is generated at the same frequency, the relative transformation between the camera and the marker (computed by Equation A-3) is performed once a new estimated $\widetilde{T}_L^C$ is arrived and use the newest measurement $\widetilde{T}_L^M$. Although the time stamp of $\widetilde{T}_L^C$ and $\widetilde{T}_L^M$ are not exactly the same, this problem can be settle down by keeping the camera and the landmark stationary.

Since the initialized camera's marker's orientation various each time. It is convenient to develop an online calibration tool which allow the user to perform the calibration before each experiment. And in order to perform online and stable calibration, a learning factor α is introduced to average out the measurement noise. And the final calibration result is obtained with:

$$\widehat{T}_M^C = (1-\alpha)\,\widehat{T}_M^C + \alpha\left(\widetilde{T}_L^M\right)^{-1}\widetilde{T}_L^C \tag{A-5}$$

An example of the calibration result is shown in the Figure A-4.

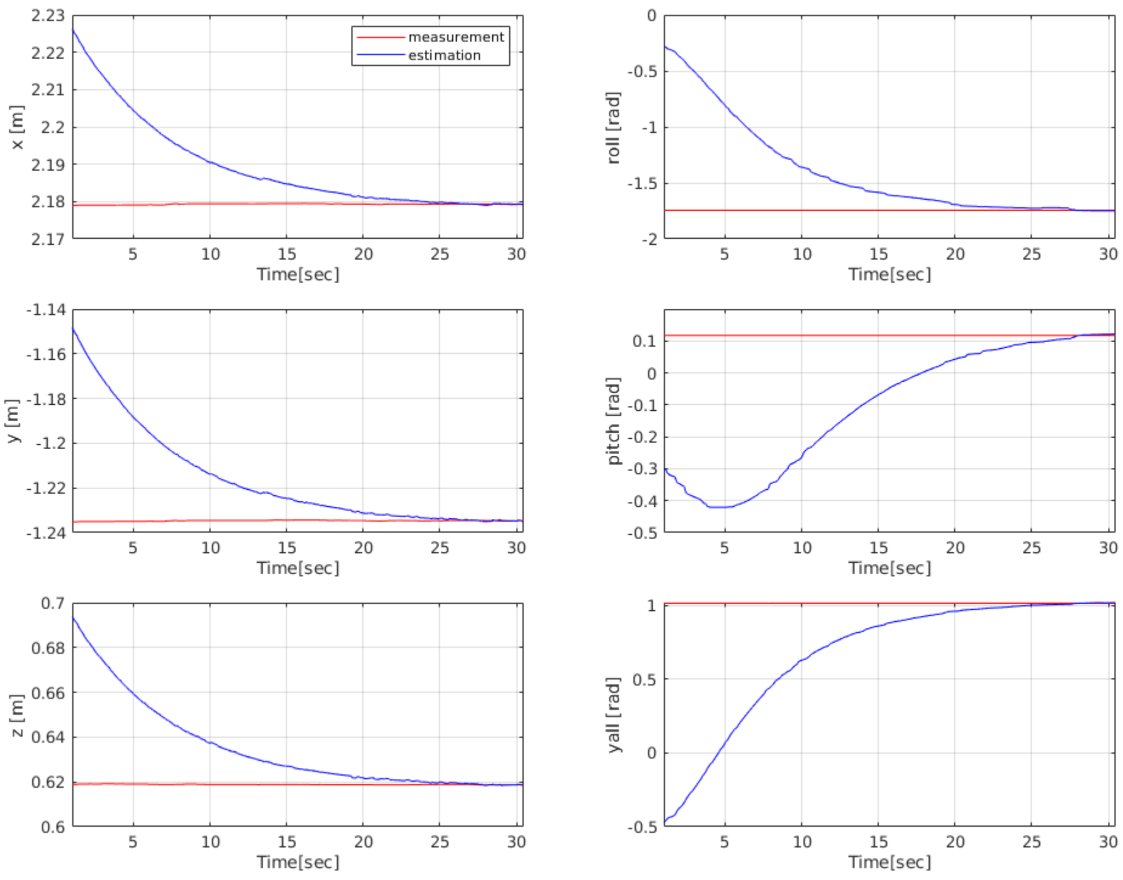


Figure A-4: The comparison between the MOCAP measurement $\widetilde{T}_W^M$ (red lines) and the estimated pose (blue lines) of the camera marker M which is computed with $\widehat{T}_M^C$, $\widetilde{T}_L^C$, $\widetilde{T}_W^L$ and the learning factor is $\alpha = 0.01$.

The result shows that the online estimation converge to the true value after approximately 25 seconds. The convergence time is adjustable, for example, increasing the learning factor α can accelerate this process, nevertheless, using a larger learning factor suggests that the measurement noise at a single frame will influence more on the final result. On the other side, using a small learning factor will slow the calibration process, however, the final result is less sensitive to a single measurement noise.

An intuitive example of the calibration result is shown in the Figure A-5. The depth camera is pointing to the ground plane in the world frame. Before applying the calibration result, the point cloud of the ground plane generated by the depth camera is not consistent with the plane $z = 0$ in world frame, this is because the camera marker's frame M is used instead of the camera frame C. After the relative transformation $\widehat{T}_M^C$ is applied, the point cloud is well aligned with the the plane $z = 0$, which illustrates that the camera pose in the world frame can be obtained after the calibration.

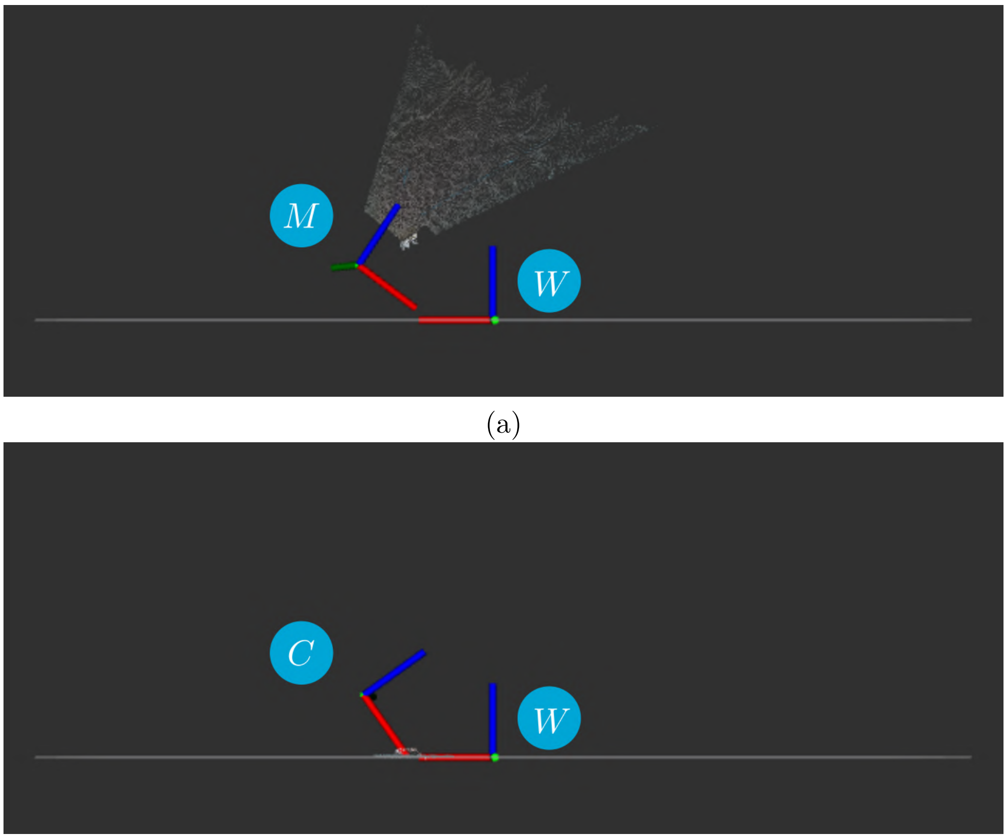


Figure A-5: A comparison between the extracted camera pose and corresponding point cloud (a) before and (b) after the camera-MOCAP calibration. Before the calibration, as the relative transformation $\widehat{T}_M^C$ is unknown, camera marker frame M is used instead of the camera frame C, while after the calibration, the camera frame is calculated and used.

A-4 Platform Identification

In order to avoid dynamic obstacles with MPC, the system need to perform collision checking in the short-time prediction horizon. This requires the system to predict the MAV's position and the obstacles' positions in the future time. Thus, the dynamic model of the MAV need to be modeled and identified. In this work, a nonlinear dynamic model of the Parrot Bebop 2 is used, which is expressed as below:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(\mathrm{t}), \mathbf{u}(\mathrm{t})) \tag{A-6}$$

where $\mathbf{x}$ is the system state, which consists of MAV's position x, y, z, linear velocity v_x, v_y, v_z , attitude ϕ, θ, ψ and yaw rate ω_z relate to the initial pose. While $\mathbf{u}$ is the control input which includes the desired roll ϕ_d, pitch θ_d, yaw rate ω_{z_d} and vertical velocity v_{z_d}. And the function $\mathbf{f}(\bullet)$ is expressed as:

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{z} = v_z \\ \dot{v}_x = \dfrac{\tan\theta}{\cos\phi} g - A_x v_x \\ \dot{v}_y = \tan\phi g - A_y v_y \\ \dot{v}_z = \dfrac{1}{\tau_{v_z}} \left(K_{v_z} v_{z_d} - v_z\right) & \text{(A-7)} \\ \dot{\phi} = \dfrac{1}{\tau_\phi} \left(K_\phi \phi_d - \phi\right) & \text{(A-8)} \\ \dot{\theta} = \dfrac{1}{\tau_\theta} \left(K_\theta \theta_d - \theta\right) & \text{(A-9)} \\ \dot{\psi} = \omega_z \\ \omega_z = \omega_{z_d} & \text{(A-10)} \end{cases}$$

where g is the constant gravity, A_x and A_y are air drag coefficients, K_{v_z}, K_ϕ, K_θ are the gains and τ_{v_z}, τ_ϕ, τ_θ are the time constants of the vertical velocity, roll and pitch respectively.

And as can be seen from Equation A-7 to A-9, first-order system is used to model the system between the desired and actual tilt angle, yaw rate and vertical velocity. Although the first-order system is less accurate than higher-order systems, it requires less computations than those higher-order systems which ensures the real-time performance of the autonomous navigation. The identification result is shown in Table A-3.

Table A-3: The identified parameter of the experimental platform's dynamic model.

Parameters	K_{v_z}	K_ϕ	K_θ	τ_{v_z}	τ_ϕ	τ_θ	A_x	A_y
Value	1.2270	1.0181	1.0167	0.3367	0.2386	0.2386	0.1840	0.2007

Appendix B

Simulation Tool

To validate our implementation of the online trajectory optimization, Parrot-Sphinx[1] is used as the simulation tool. Parrot-Sphinx is a Software In The Loop (SITL) simulator that allows developer to run Parrot Bebop2's inner controller without any hardware. Gazebo[2] is integrated in the simulator, which is mainly used for the simulation of the physical and visual surroundings. As for the state estimation, MAV's actual poses are logged at a rate of 15 Hz and used as the estimated states. For the obstacle detection, virtual obstacles are fed to the MPC solver. Our implementation of the simulation framework can be accessed via https://github.com/0Jiahao/bebop_simulator/tree/sim_tool.

[1]https://developer.parrot.com/docs/sphinx/
[2]http://gazebosim.org/

Bibliography

[1] S. Hogan, M. Kelly, B. Stark, Y. Chen, *et al.*, "Unmanned aerial systems for agriculture and natural resources," *California Agriculture*, vol. 71, no. 1, pp. 5–14, 2017.

[2] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," *arXiv preprint arXiv:1806.08548*, 2018.

[3] T. Nägeli, L. Meier, A. Domahidi, J. Alonso-Mora, and O. Hilliges, "Real-time planning for automated multi-view drone cinematography," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 132, 2017.

[4] G. Brunner, B. Szebedy, S. Tanner, and R. Wattenhofer, "The urban last mile problem: Autonomous drone delivery to your balcony," *arXiv preprint arXiv:1809.08022*, 2018.

[5] C. Potena, R. Khanna, J. Nieto, R. Siegwart, D. Nardi, and A. Pretto, "Agricolmap: Aerial-ground collaborative 3d mapping for precision farming," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1085–1092, 2019.

[6] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[7] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, *et al.*, "Fast, autonomous flight in gps-denied and cluttered environments," *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.

[8] H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, *et al.*, "Challenges and implemented technologies used in autonomous drone racing," *Intelligent Service Robotics*, vol. 12, no. 2, pp. 137–148, 2019.

[9] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Beauty and the beast: Optimal methods meet learning for drone racing," *arXiv preprint arXiv:1810.06224*, 2018.

[10] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2472–2477, IEEE, 2011.

[11] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for mav flight," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 50–56, IEEE, 2015.

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

[13] B. T. Lopez and J. P. How, "Aggressive collision avoidance with limited field-of-view sensing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1358–1365, IEEE, 2017.

[14] J. Tordesillas, B. T. Lopez, J. Carter, J. Ware, and J. P. How, "Real-time planning with multi-fidelity models for agile flights in unknown environments," *arXiv preprint arXiv:1810.01035*, 2018.

[15] T. Mori and S. Scherer, "First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles," in *2013 IEEE International Conference on Robotics and Automation*, pp. 1750–1757, IEEE, 2013.

[16] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[17] M. Pivtoraiko, I. A. Nesnas, and A. Kelly, "Autonomous robot navigation using advanced motion primitives," in *2009 IEEE Aerospace conference*, pp. 1–7, IEEE, 2009.

[18] A. Dhawale, X. Yang, and N. Michael, "Reactive collision avoidance using real-time local gaussian mixture model maps," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3545–3550, IEEE, 2018.

[19] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2520–2525, IEEE, 2011.

[20] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*, pp. 649–666, Springer, 2016.

[21] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[22] J. Tordesillas, B. T. Lopez, and J. P. How, "Fastrap: Fast and safe trajectory planner for flights in unknown environments," 2019.

[23] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a gps-denied environment," in *2008 IEEE International Conference on Robotics and Automation*, pp. 1814–1820, IEEE, 2008.

[24] S. Grzonka, G. Grisetti, and W. Burgard, "Towards a navigation system for autonomous indoor flying," in *2009 IEEE international conference on Robotics and Automation*, pp. 2878–2883, IEEE, 2009.

[25] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[26] B. Houska, H. J. Ferreau, and M. Diehl, "Acado toolkitâĂŤan open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[27] A. Domahidi and J. Jerez, *FORCES Pro: code generation for embedded optimization.* https://www.embotech.com/FORCES-Pro, September 2016.

[28] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, "Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system," in *Robot operating system (ROS)*, pp. 3–39, Springer, 2017.

[29] N. Potdar, d. C. Guido, and J. Alonso-Mora, "Online trajectory planning and control of a mav payload system in dynamic environments," Master's thesis, Delft University of Technology, http://resolver.tudelft.nl/uuid:28774acc-89e7-44a8-be43-b06d2db0056c, 2018.

[30] H. Oleynikova, C. Lanegger, Z. Taylor, M. Pantic, A. Millane, R. Siegwart, and J. Nieto, "An open-source system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments," *arXiv preprint arXiv:1812.03892*, 2019.

[31] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for mavs in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[32] M. Beul, D. Droeschel, M. Nieuwenhuisen, J. Quenzel, S. Houben, and S. Behnke, "Fast autonomous flight in warehouses for inventory applications," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3121–3128, 2018.

[33] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[34] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1053–1072, 2017.

[35] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[36] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[37] H. Zhu, J. Juhl, L. Ferranti, and J. Alonso-Mora, "Distributed multi-robot formation splitting and merging in dynamic environments," 05 2019.

[38] N. J. Sanket, C. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos, "Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight," *IEEE Robotics and Automation Letters*, 2018.

[39] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2502–2509, IEEE, 2018.

[40] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[41] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pp. 225–234, IEEE, 2007.

[42] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 15–22, IEEE, 2014.

[43] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2018.

[44] M. Sanfourche, V. Vittori, and G. Le Besnerais, "evo: A realtime embedded stereo odometry for mav applications," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 2107–2114, IEEE, 2013.

[45] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[46] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 2100–2106, Citeseer, 2013.

[47] F. Fraundorfer and D. Scaramuzza, "Visual odometry: Part i: The first 30 years and fundamentals," *IEEE Robotics and Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[48] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[49] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3d reconstruction," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 363–370, IEEE, 2006.

[50] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *null*, p. 1403, IEEE, 2003.

[51] R. Martinez-Cantin and J. A. Castellanos, "Bounding uncertainty in ekf-slam: the robocentric local approach.," in *ICRA*, pp. 430–435, 2006.

[52] N. de Palézieux, T. Nägeli, and O. Hilliges, "Duo-vio: Fast, light-weight, stereo inertial odometry," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 2237–2242, IEEE, 2016.

[53] M. Trajković and M. Hedley, "Fast corner detection," *Image and vision computing*, vol. 16, no. 2, pp. 75–87, 1998.

[54] J. K. Suhr, "Kanade-lucas-tomasi (klt) feature tracker," *Computer Vision (EEE6503)*, pp. 9–18, 2009.

[55] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 573–580, IEEE, 2012.

[56] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[57] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf.," in *ICCV*, vol. 11, p. 2, Citeseer, 2011.

[58] J. Shi and C. Tomasi, "Good features to track," tech. rep., Cornell University, 1993.

[59] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems," in *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, vol. 2, 2010.

[60] S. Song and J. Xiao, "Deep sliding shapes for amodal 3d object detection in rgb-d images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 808–816, 2016.

[61] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4490–4499, 2018.

[62] H. Badino, U. Franke, and D. Pfeiffer, "The stixel world-a compact medium level representation of the 3d-world," in *Joint Pattern Recognition Symposium*, pp. 51–60, Springer, 2009.

[63] B. Ruf, S. Monka, M. Kollmann, and M. Grinberg, "Real-time on-board obstacle avoidance for uavs based on embedded stereo vision," *arXiv preprint arXiv:1807.06271*, 2018.

[64] K. Schauwecker and A. Zell, "Robust and efficient volumetric occupancy mapping with an application to stereo vision," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6102–6107, IEEE, 2014.

[65] T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in *2017 Ieee/rsj International Conference on Intelligent Robots and Systems (iros)*, pp. 2135–2142, IEEE, 2017.

[66] "Allan variance: Noise analysis for gyroscopes," *Freescale Semiconductor Document Number: AN5087 Application Note Rev. 0*, vol. 2, 2015.

[67] "Ieee standard specification format guide and test procedure for single-axis interferometric fiber optic gyros," *IEEE Std 952-1997*, pp. 1–84, Feb 1998.

[68] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1280–1286, IEEE, 2013.

[69] J. Nocedal and S. Wright, *Numerical optimization.* Springer Science & Business Media, 2006.

[70] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 8, pp. 930–943, 2003.

Glossary

List of Acronyms

MAV	Micro Aerial Vehicle
VO	Visual Odometry
VIO	Visual-Inertial Odometry
IMU	Inertial Measurement Unit
ROVIO	Robust Visual Inertial Odometry
S-MSCKF	Stereo Multi-State Constraint Kalman Filter
VINS	Visual-Inertial System
OKVIS	Open Keyframe-based Visual-Inertial SLAM
VSLAM	Vision-based Simultaneous Localization and Mapping
ATE	Absolute Trajectory Error
RPE	Relative Pose Error
CPU	Central Processing Unit
ORB	Oriented FAST and Rotated BRIEF
MPC	Model Predictive Controller
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
LIDAR	Light Detection And Ranging of Laser Imaging Detection and Ranging
MOCAP	Motion Capture System
3D	Three Dimensional

2D	Two Dimensional
RGB	Red Green Blue
RGBD	Red Green Blue and Depth
RAM	Random Access Memory
GPU	Graphics Processing Unit
GFLOPS	Gigafloating-Point Operations per Second
WIFI	Wireless Fidelity
USB	Universal Serial Bus
SDK	Software Development Kit
ROS	Robot Operating System
FOV	Field of View
LM	Levenberg-Marquardt
PNP	Perspective-n-Point
RRT	Rapidly-exploring Random Trees
SURF	Speed-Up Robust Feature
GMM	Gaussian Mixture Model
ICRA	International Conference on Robotics and Automation
KLT	Kanade-Lucas-Tomasi
EKF	Extended Kalman Filter
SITL	Software In The Loop
IROS	International Conference on Intelligent Robots and Systems
ADR	Autonomous Drone Race