

Vision-based navigation for autonomous drone flights inside a forest

Väinö Karjalainen

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of
Science in Technology.

Espoo 13.4.2023

Supervisor

Prof. Ville Kyrki

Advisor

Prof. Eija Honkavaara

Copyright © 2023 Väinö Karjalainen



Author Väinö Karjalainen

Title Vision-based navigation for autonomous drone flights inside a forest

Degree programme Automation and Electrical Engineering

Major Control, Robotics and Autonomous Systems

Code of major ELEC3025

Supervisor Prof. Ville Kyrki

Advisor Prof. Eija Honkavaara

Date 13.4.2023

Number of pages 92+2

Language English

Abstract

In recent years, the autonomous flying of drones has been an actively researched topic in both commercial and academic organizations. Most autopilots can fly autonomously in open areas where Global navigation satellite systems (GNSS) are available. However, inside dense forest environments, the localization of the drone cannot rely on GNSS, and the drone also has to avoid obstacles in the path.

The objective of this thesis was to design and implement a prototype of an autonomous drone flying under the canopy for boreal forest research purposes. To establish a starting point, a literature survey on available open-source solutions was performed. Based on the literature survey, EGO-Planner-v2 with VINS-Fusion localization and stereo depth camera-based mapping was chosen as the base of the implemented prototype.

The system was tested both in a simulator and in real forest environments with custom drone hardware. The performance of the system, and its suitability for boreal forest environments, were evaluated based on the success of the mission, reliability of the obstacle avoidance, and the accuracy of the localization.

Based on the results, the performance of the system was promising in sparse forests. In the sparse mixed forest, eight of nine flights were successful, when approximate flight distances varied between 13 m and 18 m. However, in dense forests, the sensing of small needleless branches needs to be improved to increase reliability. In the dense spruce forest, nine of 19 test flights were successful, when approximate flight distances varied between 35 m and 80 m. In the longest, approximately 80 m long, test flight, the error of the VINS-Fusion estimate of the trajectory length was approximately 1 m.

Keywords drone, depth camera, autonomous flying, visual-inertial odometry



Tekijä Väinö Karjalainen

Työn nimi Droonien autonominen kamerapohjainen navigointi metsässä

Koulutusohjelma Automaatio- ja sähkötekniikka

Pääaine Sääntötekniikka, robotiikka ja itsenäiset järjestelmät **Pääaineen koodi** ELEC3025

Työn valvoja Prof. Ville Kyrki

Työn ohjaaja Prof. Eija Honkavaara

Päivämäärä 13.4.2023

Sivumäärä 92+2

Kieli Englanti

Tiivistelmä

Viime vuosina droonien autonominen lentäminen on ollut aktiivisesti tutkittu aihe sekä akateemisissa että kaupallisissa organisaatioissa. Useimmat droonien autopilotit pystyvät lentämään autonomisesti avoimilla alueilla, missä paikannuksessa voidaan hyödyntää globaaleja paikannussatelliitteja (GNSS). Lennettäessä tiheissä metsissä satelliittipaikannus ei kuitenkaan ole mahdollista, ja droonin täytyy lisäksi kyetä väistelemään esteitä.

Tämän työn tavoitteena oli toteuttaa metsätutkimukseen tarkoitettu prototyyppi autonomisesti metsän sisällä lentävästä droonista. Työn alussa toteutettiin kirjallisuustutkimus viimeaikoina julkaistuihin avoimen lähdekoodin ratkaisuihin metsän sisällä lentävistä autonomisista drooneista. Kirjallisuustutkimuksen perusteella toteutettavan prototyypin pohjaksi valittiin EGO-Planner-v2, joka käytti paikannukseen VINS-Fusionia ja kartoitukseen stereo-syvyyskameraa.

Systeemiä testattiin sekä simulaattorissa että oikeissa metsissä itserakennetulla droonilla. Suorituskykyä ja soveltuvuutta pohjoisiin metsäympäristöihin arvioitiin lentojen onnistumisen, esteiden väistelyn luotettavuuden ja paikannuksen tarkkuuden perusteella.

Tulosten perusteella suorituskyky oli lupaava puustoltaan harvoissa metsissä. Puustoltaan harvassa sekametsässä kahdeksan yhdeksästä testilennosta onnistui, kun testilentojen pituudet vaihtelivat noin 13 metrillä noin 18 metriin. Kuitenkin tiheissä metsissä pienten havuttomien oksien havainnointia täytyy kehittää navigoinnin luotettavuuden parantamiseksi. Tiheässä kuusimetsässä yhdeksän 19:sta testilennosta onnistui, kun lentojen pituudet vaihtelivat noin 35 metrillä 80 metriin. Pisimmässä, noin 80 metriä pitkässä, testilennossa VINS-Fusionin estimaatissa lennon pituudeksi virhe oli noin yksi metri.

Avainsanat drooni, syvyyskamera, autonominen lentäminen, visuaali-inertiaalinen odometria

Preface

Research in this thesis was funded by the Academy of Finland within projects “Finnish UAV Ecosystem” (decision no. 337018) and Fireman (decision no. 346710). This study has been performed with affiliation to the Academy of Finland Flagship Forest–Human–Machine Interplay—Building Resilience, Redefining Value Networks and Enabling Meaningful Experiences (UNITE) (decision no. 337127).

I would like to deliver my thanks to my advisor, D.Sc. Eija Honkavaara, for her constant support, guidance, and constructive comments throughout the thesis process. I would also like to thank my thesis supervisor Prof. Ville Kyrki for his guidance at the beginning of the thesis process and his patience during the delayed writing process.

I would also like to thank all the researcher colleagues in FGI who have helped and supported me during the process. Anand George helped me throughout the process, especially with his wide experience and knowledge of VIO solutions. Niko Koivumäki processed the simulated forest in Chapter 5.4 with Metashape. Juha Suomalainen gave valuable feedback during the regular research group meetings. Harri Kaartinen processed the point cloud of the forest in Chapter 6.4 with GeoSLAM Connect. Especially big thanks I want to deliver to Teemu Hakala. He assembled the drone hardware, worked as a backup pilot during the flight tests, and recorded the forest point cloud data in Chapter 6.4. He also spent countless hours with me tuning the parameters and doing the initial flight tests in the basement of the FGI office, sometimes even late in the evenings. I would also like to thank all my colleagues who were writing their own thesis at the same time for peer support on difficult days.

I would also like to express my thanks to my university friends. Studying without them would have been much more difficult, and doing group works would have been a much more boring experience.

Last but not the least, I would like to thank my parents, brother, and girlfriend for their support during my studies.

Otaniemi, 13.4.2023

Väinö E. Karjalainen

Contents

Abstract	3
Abstract (in Finnish)	4
Preface	5
Contents	6
Abbreviations	8
1 Introduction	9
2 Background	11
2.1 Depth imaging	11
2.2 Vision based motion estimation	13
2.2.1 Visual odometry	13
2.2.2 Visual-Inertial odometry	15
2.3 Mapping	16
2.4 Path planning	18
2.4.1 Traditional path planning algorithms	18
2.4.2 Machine learning based methods	19
3 Autonomous flying inside a forest	21
3.1 State-of-the-art solutions	21
3.1.1 Campos-Macías et al.	21
3.1.2 Loquercio et al.	22
3.1.3 Liu et al.	23
3.1.4 Zhou et al.	25
3.2 Proposal of the method	26
4 Selected method for autonomous flying inside a forest	29
4.1 Middleware	29
4.2 Localization	31
4.3 Mapping	32
4.4 Path planning	32
4.5 Trajectory tracking	36
5 Experiments in simulation	40
5.1 Simulator	40
5.1.1 Software	40
5.1.2 Simulated drone	41
5.2 Obstacle avoidance in simulated forests	41
5.2.1 Environment and the experimental setup	42
5.2.2 Results	43
5.3 VIO precision in simulated forests	46

5.3.1	Environment and the experimental setup	47
5.3.2	Results	47
5.4	Overall flight performance	50
5.4.1	Environment and the experimental setup	50
5.4.2	Results	50
5.5	Discussion	52
6	Experiments with real hardware	55
6.1	Description of the platform	55
6.1.1	Drone hardware	55
6.1.2	Calibration	56
6.1.3	Commissioning	58
6.2	Flying in a sparse mixed forest	58
6.2.1	Environment and the experimental setup	58
6.2.2	Results	59
6.3	Flying in a park woodland with dense understory vegetation	63
6.3.1	Environment and the experimental setup	63
6.3.2	Results	64
6.4	Flying in a snowy spruce forest	68
6.4.1	Environment and the experimental setup	68
6.4.2	Results	70
6.5	Discussion	76
7	Discussion	79
7.1	Limitations of the study	79
7.2	Ways to improve the system and potential applications	80
8	Conclusion	83
	References	84
A	Wind Speeds during the flight tests	93

Abbreviations

ATE	Absolute Trajectory Error
CPU	Central Processing Unit
DLT	Direct Linear Transformation
ESC	Electronic Speed Controller
GNSS	Global Navigation Satellite System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
LiDAR	Light Detection And Ranging
MINCO	Minimum Control -trajectory
NN	Neural Network
ODE	Open Dynamics Engine
PnP	Perspective from n Point
RLS	Recursive Least Squares
ROS	Robot Operating System
RPE	Relative Pose Error
RRT	Rapidly exploring random tree
SITL	Software In The Loop
SLAM	Simultaneous Localization And Mapping
UAS	Uncrewed Aircraft System
UWB	Ultra-Wideband
VIO	Visual-Inertial Odometry
VISLAM	Visual-Inertial SLAM
VO	Visual Odometry
VSLAM	Visual SLAM

1 Introduction

In recent years, uncrewed aircraft systems (UAS, drones) have been an actively researched topic in both academic and commercial organizations. Although manual remote control is still a common way to control flights, especially in obstacle-rich areas, autonomous flights have made new autonomous applications in various fields possible. Autonomous operations of drones can be useful, for example, in agriculture, military operations, search and rescue operations, mapping, and delivery of packages and medicines [1–6]. Drones are also widely used for research purposes. For example in forest research, data collected by drones can be used for estimating the height of the canopy and the stem volumes [7, 8], and flying inside forests with cameras could open new possibilities for data collection in biodiversity research.

To fly autonomously, a drone must be able to sense its environment, localize itself, and plan its path autonomously. Usually, also a map of the environment is created by the drone. Most autonomous drones use the Global Navigation Satellite System (GNSS) to estimate the location. However, GNSS signals are not available in all environments, and the signals can be jammed or spoofed [9]. Especially under the dense forest canopy, localization with a GNSS can be impossible due to blocked signals or reflections producing a multipath effect [10].

When GNSS is not available, drones must rely on alternative solutions to localize themselves. A typical sensor setup for localization in mobile robotics is a combination of an inertial measurement unit (IMU) and light detection and ranging (LiDAR). As many papers suggest, using LiDARs for localizing a drone is possible [11–13]. However, LiDARs are usually heavy in comparison to the weight of the drone itself. The heaviness of the sensor setup can be a limiting factor when the aim is to design a small drone. LiDAR is also an active sensor that increases the drone’s power consumption and hence reduces the flight time of the drone.

An alternative for LiDAR-based localization is camera-based Visual Odometry (VO). In VO, the ego-motion of the system is estimated based on the input from the camera or multiple cameras attached to the system [14]. The accuracy of the estimation can be increased by fusing IMU data with camera input and using visual-inertial odometry (VIO) [15]. In addition to the localization, the environment can be mapped simultaneously using Simultaneous Localization and Mapping (SLAM). By attaching either a camera system or both a camera system and an IMU to a drone, it can map the environment and localize itself in that map by using either visual SLAM (VSLAM) or visual-inertial SLAM (VISLAM) respectively [16].

In addition to localization, path planning and obstacle avoidance are needed to fly autonomously. The most common way of achieving that is building a map, planning a path based on that map, and executing a trajectory-tracking feedback control along that path. There exists also approaches that forgo maintaining a map and use only current sensor information for reacting to obstacles [17].

Even though autonomous flying in GNSS-denied environments has been an actively researched topic in recent years, the performances of the systems are quite rarely evaluated inside dense boreal forests. In boreal forest research, collecting data under the canopy is still based on manual flights [8].

The objective of this thesis is to design and implement a small-sized autonomous drone flying under the canopy. The aim of the system is to be a prototype for collecting image data for forest research autonomously. To fulfill the aim, the implemented drone has to be able to autonomously navigate and fly through dense boreal forests to a given goal point. However, doing higher-level planning and area coverage is beyond the scope of this thesis.

To achieve the objective, a literature survey of existing solutions that have been flight tested in an unknown forest environment is carried out. The specific research question to which the literature survey aims to answer is what kind of open-source solutions already exist in the literature. Based on the literature survey, the most promising solution is proposed and chosen as the base of the implemented system. The thesis evaluates the feasibility of the system for flying inside dense boreal forests and examines the limitations of the system concerning the forest type and the density of the forest. The evaluations are made both in simulated environments and in real forests with physical hardware. The evaluations examine how operational the system is in boreal forests and answer for following research questions. How does the flying velocity or forest density affect the performance of obstacle avoidance? What is the accuracy of the VIO estimate? What is the reliability of the navigation of the system in boreal forests? What kind of obstacles cause problems for navigation?

The remainder of this thesis is structured as follows. Chapter 2 provides the theoretical foundations for the thesis. Chapter 3 presents a literature survey of existing open-source solutions. At the end of the chapter, the most promising solution is proposed and chosen as the base of the implemented system. Chapter 4 gives a more detailed introduction to the chosen solution, and presents the algorithms used in its implementation.

In Chapter 5, the suitability of the solution is evaluated by flight tests in a simulator. The performance of the two main processes, obstacle avoidance and VIO localization, are first studied independently with short flights in simulated forests with artificial tree models. The first-mentioned experiments study the effect of forest density and flying speed on the performance of the system, and the latter evaluates the accuracy and the precision of the used VIO algorithm. After that, the performance of a complete system is tested in the longer flights with a simulated drone on a forest model generated from point cloud data from a Finnish pinewood forest. This experiment validates the joint operation of the aforementioned modules but also studies their performance in longer flights.

In Chapter 6 the performance of the system is evaluated with test flights with real drone hardware. Multiple test flights are performed in three different forest environments with varying vegetation and density. The experiments study the reliability of the navigation and the accuracy of the VIO in boreal forest environments. The test flights also study, what kind of objects cause problems for the system.

Chapter 7 contains a discussion about the known limitations of the system and the study. The chapter also gives proposals for potential future improvements to the system. Chapter 8 provides the conclusion and the summary of the thesis.

2 Background

Autonomous flying in GNSS-denied environments requires a capacity to navigate autonomously relying on only onboard perception and computation. The drone has to be able to sense its environment, localize itself, avoid obstacles, and plan a safe flying path from its current location to its destination. Following subchapters provide the theoretical foundations for the state-of-the-art solutions presented in Chapter 3.

Chapter 2.1 presents the theory of depth imaging. Every solution in Chapter 3 used VIO for localization, so the theoretical foundations of VO and VIO are presented in Chapter 2.2. Chapter 2.3 concentrates on mapping the obstacles. Chapter 2.4 presents the algorithms that were used for finding obstacle-free paths in the state-of-the-art solutions in Chapter 3.

2.1 Depth imaging

In order to be autonomous, the system needs to perceive its environment. When the system moves in the three-dimensional world, also the perception needs to happen in three dimensions. Point clouds are a typical method for representing three-dimensional knowledge about the environment. There exist various methods for obtaining 3D point clouds based on active vision and the methods can be divided into triangulation-based methods and time-of-flight-based methods [18, pp. 12-26].

Triangulation-based methods can be further divided into stereoscopic methods and structured light methods [18, pp. 12-20]. In stereoscopy, the depth is reconstructed based on the disparity occurring between two frames capturing the same scene from different points of view. Figure 1a shows a simplified two-dimensional model for triangulation in stereoscopy where two cameras having parallel optical axes, focal length f , and baseline B are observing a point P . Points p_l and p_r are the optical projections of the point P on the left and right camera planes. When the camera parameters are known, due to the similarity of the triangles $p_l P p_r$ and $O_l P O_r$, the depth z can be obtained by:

$$z = \frac{Bf}{x_r - x_l} = \frac{Bf}{d} \quad (1)$$

where d is called disparity. Three-dimensional triangulation is based on epipolar geometry. Figure 1b shows two cameras with centers \mathbf{C} and \mathbf{C}' observing the 3D point \mathbf{X} . The point is projecting point projects \mathbf{x} and \mathbf{x}' to camera planes. The epipolar plane π is defined by point \mathbf{X} and camera centers \mathbf{C} and \mathbf{C}' , and it is intersecting camera planes in epipolar lines. The baseline between camera centers intersects the camera planes at the epipoles \mathbf{e} and \mathbf{e}' . Having only one projection \mathbf{x} with camera center \mathbf{C} , the depth of point \mathbf{X} remains ambiguous. When the relative pose of the other camera with center \mathbf{C}' is known, the epipolar line \mathbf{l}' can be defined, and finding the projection \mathbf{x}' in the second camera along \mathbf{l}' solves the ambiguity. A detailed derivation of the solution is beyond the scope of this thesis and can be found from [18, pp. 14-15], but it can be shown that \mathbf{Z}^* and $\mathbf{Z}^{*'}$, corresponding to depth estimates of \mathbf{X} in camera frames of \mathbf{C} and \mathbf{C}' , can be obtained by minimizing the

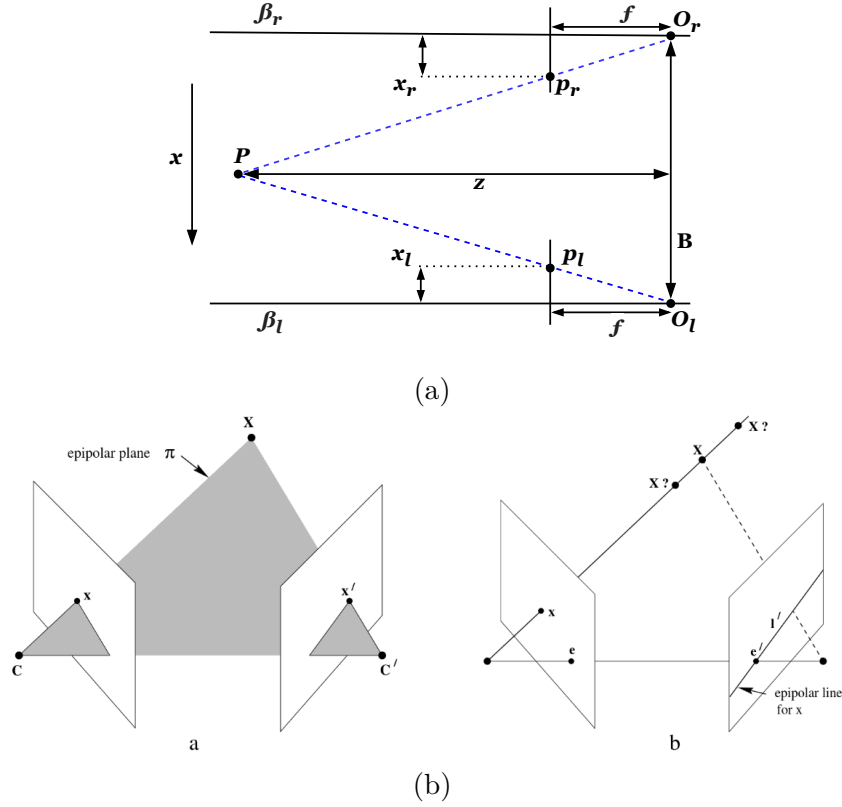


Figure 1: (a) Triangulation in stereoscopy. Adapted from [18]. (b) Epipolar geometry [19].

loss function over \mathbf{Z} and \mathbf{Z}' :

$$(\mathbf{Z}^*, \mathbf{Z}'^*) = \operatorname{argmin}_{\mathbf{Z}, \mathbf{Z}'} \| (\mathbf{Z}\mathbf{K}^{-1}\mathbf{x}) - (\mathbf{R}^{-1}\mathbf{Z}'\mathbf{K}'^{-1}\mathbf{x}' - \mathbf{R}^{-1}\mathbf{t}) \|_2 \quad (2)$$

where \mathbf{K} and \mathbf{K}' are the camera calibration matrices, and \mathbf{R} and \mathbf{t} are the rotation and the translation between the two cameras.

Before the triangulation can be performed, observed points need to be found by searching for correspondences between the frames of the two cameras. Solutions for matching can be divided into correlation-based methods and feature-based methods [18, pp. 16]. In correlation-based matching, a fixed-size window is defined around any point of an image, and the second image is correlated with it to find a corresponding point in another picture. Feature-based methods first detect points that can be unambiguously identified across frames, such as edges and corners, from both pictures and then matches them based on a comparison between descriptive features.

Active stereoscopy depth cameras project a random light pattern to cope with the objects that lack enough features [18, pp. 17-18]. A randomized dot pattern in an infrared domain is projected to force a texture with features to triangulate. Using infrared light aims to keep the pattern insensitive with respect to the ambient light. However, since the sunlight contains infrared light, in outdoor conditions the ambient light can still interfere with the features [20].

In structured-light-based cameras, one camera is substituted by a laser projector [18, pp. 18-20]. A laser is projecting a structured infrared pattern in a known direction and a single camera is used for triangulation. The pattern is formed either by a time-multiplexing strategy, a direct coding strategy, or a spatial neighborhood strategy. The time-multiplexing strategy uses a pattern that splits the scene into several areas creating edges that can be used for triangulation. Direct coding projects a priori-known color-coded or grey scale pattern allowing a direct depth measurement with a single frame. Spatial neighborhood strategy projects a spatially-structured pattern creating uniqueness in the neighborhood of projected pixels. The spatial neighborhood method makes it possible to reconstruct 3D depth from a single frame.

Time-of-flight cameras estimate depth with direct range measurement by exploiting the time-of-flight principle for every pixel of the camera [18, pp. 20-26]. The time-of-the-flight camera consists of a light transmitter and a receiving camera, and the depth is estimated based on the round-trip time of the flight from the transmitter to the camera. All depth cameras in this thesis use stereoscopy for estimating the depth so the more detailed foundations of time-of-the-flight cameras are beyond the scope of this thesis.

2.2 Vision based motion estimation

2.2.1 Visual odometry

The general problem of recovering the three-dimensional structure of the surface and the relative camera poses from a set of images is known as structure from motion (SfM). VO is a particular case of SfM, which aims to estimate the pose of a camera system with respect to its initial coordinate frame in real time based on the sequence of images [21]. The pose estimate is acquired by concatenating the relative poses of adjacent camera positions computed from visual features. VO methods can be divided into two main approaches. Appearance-based methods use the intensity information of all the pixels in two adjacent input images. Feature-based methods extract or track features across the sequence of images. Feature-based methods are computationally less expensive and more accurate than appearance-based methods and hence most VO implementations are feature-based [21].

Figure 2 summarizes the pipeline of VO. In the first two steps, features are detected for every new stereo image pair. In VO typical features are corners and blobs, since their position in the images can be defined accurately [22]. Blobs are image patterns whose texture differs from their environment. Corners are faster to compute and better localized in image position than blobs, but blobs are more robust to changes in viewpoint. Examples of popular feature detectors are Harris [23], Shi-Tomasi [24], and Fast [25] for corners, and SIFT [26] and SURF [27] for blobs.

For detected features, the region around every feature point is converted into a feature descriptor [22]. The simplest descriptor is the intensity of the pixels around the feature point, but more complex descriptors are needed for better robustness in larger camera motions. For example, the SIFT descriptor is a histogram of local

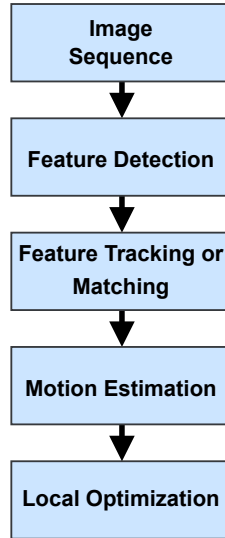


Figure 2: The main components of VO. Adapted from [21].

gradient orientations.

After detection, the features are either tracked or matched to features in the previous images. In feature-matching-based solutions, the features are detected independently in all images and matched to each other based on some similarity metric between the feature descriptors. In feature tracking-based methods the features are detected only in the first image of the sequence, and then the corresponding matches are searched from the following images using local search techniques.

The camera motion estimation between two frames k and $k - 1$ is modelled as rigid body transformation $\mathbf{T}_{k,k-1} \in \mathbb{R}^{4 \times 4}$ [21]. The rigid body transformation can be presented in the form

$$\mathbf{T}_{k,k-1} = \begin{bmatrix} \mathbf{R}_{k,k-1} & \mathbf{t}_{k,k-1} \\ 0 & 1 \end{bmatrix} \quad (3)$$

where $\mathbf{R}_{k,k-1} \in SO(3)$ is the rotation and $\mathbf{t}_{k,k-1} \in \mathbb{R}^{3 \times 1}$ is the translation. The choice of the motion estimation method depends on the dimensions where the image features \mathbf{f}_k and \mathbf{f}_{k-1} are specified.

In 2D-to-2D case, both \mathbf{f}_k and \mathbf{f}_{k-1} are specified in 2D image coordinates. Estimating the geometric relations between two images is based on calculating the so-called essential matrix \mathbf{E} , where rotation and translation can directly be extracted. The essential matrix can be calculated efficiently based on the epipolar constraint between two frames using the five-point algorithm by Nister [28] if the camera is calibrated or the eight-point algorithm by Longuet-Higgins [29] if the camera is uncalibrated. The absolute scale of the translations cannot be computed straight from 2D images. Hence the relative scale of the translations is computed by triangulating 3D points from two subsequent images, and that scale factor is used for rescaling the translations before the concatenation of transformations.

In the 3D-to-3D case, features are triangulated at every time instant, and both \mathbf{f}_k and \mathbf{f}_{k-1} are both specified in 3D coordinates. The transformation $\mathbf{T}_{k,k-1}$ is calculated

by minimizing the L_2 distance between two 3D feature sets. The transformations computed in the 3D-to-3D case have absolute scale, so the transformations can be directly concatenated.

In 3D-to-2D case, \mathbf{f}_{k-1} are specified in 3D coordinates, and \mathbf{f}_k are specified in 2D image coordinates. The transformation is obtained by finding $\mathbf{T}_{k,k-1}$ that minimizes the reprojection error between the 2D feature in frame k and the reprojection of the 3D point in frame $k-1$ into 2D image coordinates in frame k according to the transformation $\mathbf{T}_{k,k-1}$. The problem is known as perspective from n points (PnP)-problem. Having at least six 3D-to-2D point correspondences makes it possible to form a linear system that can be solved using the direct linear transformation (DLT) by stacking the constraints given by point correspondences. The rotation and translation acquired by linear solving can be further refined by nonlinear optimization of the reprojection error.

The camera poses computed by VO can be represented as a graph, where the camera poses are the nodes and the transformations are the edges between the nodes [22]. In addition to concatenating the transformations from two subsequent frames, the camera pose estimate can be further optimized by computing the transformation between the current frame k and some older frame, and using it as an additional constraint edge in a graph. The graph optimization seeks the camera poses that minimize the cost function defined by the edge constraints. Seeing a landmark after not seeing it for a while is called loop detection. Loop constraints are valuable since they typically form an edge between two nodes that are far from each other in the graph, and potentially a large drift has been accumulated between them.

A successful approach for detecting loops is representing the image with a bag of visual words [22]. A visual word represents feature descriptors with a single integer number, and the visual similarity between the images is computed as the distance of the visual word histograms. In addition to graph optimization, windowed bundle adjustment (also called local optimization) can be used for optimizing the camera poses and 3D landmark positions in cases where image features are tracked longer than two frames. Windowed bundle adjustment uses a window of n frames and performs the optimization for the reprojection error in this set of images. Since the reprojection error is a nonlinear function, typically the optimization is performed using the Levenberg-Marquardt algorithm.

2.2.2 Visual-Inertial odometry

Visual-Inertial odometry combines data from a camera and an IMU to increase the precision and robustness of the pose estimate. Cameras and IMUs can be seen as ideal complementary sensors [30]. Cameras provide rich information and are precise in slow motion, but limited output rate and motion blur are problems in high-speed motions. Also, homogenous scenes without a texture, and over- or under-exposure of images cause problems for pure visual odometry. IMUs have a high output rate and are unaffected by the aforementioned scene-dependent difficulties, but they suffer from poor signal-to-noise ratio at low angular velocities and low accelerations. The motion from an IMU alone drifts quickly, but a combination of a camera system and

an IMU can provide robust pose estimates in different scenarios.

VIO approaches can be divided into two categories, which Corke et al. [31] call *loosely coupled* and *tightly coupled*. Loosely coupled approaches run inertial and visual separately and exchange information between them. Tightly coupled approaches directly fuse the raw data from IMU and camera to get the pose estimate. The difference in the pipeline of the approaches is illustrated in Figures 3a and 3b. Tightly coupled approaches are more accurate than loosely coupled ones [30].

The VIO approaches can also be categorized into Filtering-based methods and Smoothing-based methods [30]. Filtering methods only estimate the latest state. The information from older states is absorbed into the estimation of the latest state and then dropped permanently. That has the problem that the linearization error of the filter is locked in the state, and cannot be corrected afterward. Instead of estimating only the latest state, Fixed-lag smoothers (also called sliding window estimators) estimate all states that are inside the given time window. Older states are marginalized out. Fixed-lag smoothers are more accurate than Filtering-based methods since part of the past measurements are relinearized as the estimate is updated, but estimating multiple states makes them computationally more expensive. Full smoothing methods estimate the entire state history by large-scale nonlinear optimization. Full-smoothing methods can update the linearization point of the complete state history making them the most accurate approach. However, the complexity of the optimization of all states makes the real-time operation quickly infeasible, so typically only selected keyframes are kept and used in optimization.

2.3 Mapping

The traditional way of mapping in mobile robotics applications is using occupancy grid maps first proposed by Moravec and Elfes in 1985 [32]. Occupancy grid maps present the environment in an evenly spaced grid, where every cell contains a binary variable corresponding to the occupancy of the location that the cell covers [33].

The basic idea behind the occupancy grid maps is calculating the posterior over the maps. However, calculating the posterior probability for all possible states of the map will quickly become computationally unfeasible when the number of grid cells increases. The standard approach is to estimate the posterior independently for all grid cells and approximate the posterior of the map as the product of independent cell posteriors:

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(\mathbf{m}_i|z_{1:t}, x_{1:t}) \quad (4)$$

where m is the map, \mathbf{m}_i is a grid cell, $z_{1:t}$ is the set of measurements up to time t , and $x_{1:t}$ is the path of the robot.

Occupancy grid maps typically use log odds representation of the occupancy:

$$l_{t,i} = \log \frac{p(\mathbf{m}_i|z_{1:t}, x_{1:t})}{1 - p(\mathbf{m}_i|z_{1:t}, x_{1:t})} \quad (5)$$

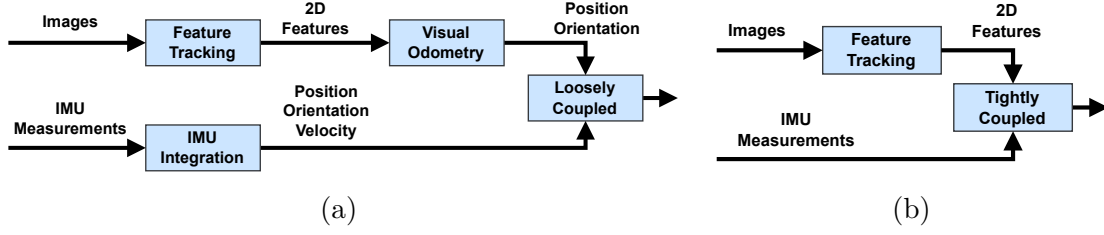


Figure 3: (a) Loosely coupled VIO. Adapted from [30]. (b) Tightly coupled VIO. Adapted from [30].

where the probabilities can be recovered by

$$p(\mathbf{m}_i | z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + \exp(l_{t,i})} \quad (6)$$

In simple, the occupancy grid mapping algorithms loop through all map cells, and update the log odds values for all cells that are in the sensing area with an inverse sensor model:

$$l_{t,i} = l_{t-1,i} + \text{inverse_sensor_model}(\mathbf{m}_i | z_{1:t}, x_{1:t}) - l_0 \quad (7)$$

Inverse sensor model

$$\text{inverse_sensor_model}(\mathbf{m}_i | z_t, x_t) = \log \frac{p(\mathbf{m}_i | z_t, x_t)}{1 - p(\mathbf{m}_i | z_t, x_t)} \quad (8)$$

and the occupancy prior

$$l_0 = \log \frac{p(\mathbf{m}_i)}{1 - p(\mathbf{m}_i)} \quad (9)$$

are also in log odds form.

The value $\text{inverse_sensor_model}(\mathbf{m}_i | z_t, x_t)$ is $l_{occ} > l_0$ if the sensor detects the cell occupied, $l_{free} < l_0$ if the sensor detects the cell as free and l_0 if the cell is behind a cell that was detected as occupied.

Instead of posterior values, the occupancy grid maps represent cells in binary values with the mode of the posterior. The mode of the map posterior is defined as

$$m^* = \underset{m}{\operatorname{argmax}} \log p(m | z_{1:t}, x_{1:t}) \quad (10)$$

Although the most common domain of occupancy grid maps is 2D maps, occupancy grid maps generalize to 3D representations also. However, the computational cost significantly increases especially with larger 3D occupancy grid maps. Popular open-source 3D mapping framework OctoMap reduces the computational cost by representing the map as a multi-resolution grid stored in octree [34].

In addition to grid maps, some robotic systems maintain also sparser maps based on detecting the features of certain types of landmarks. These maps are called semantic maps since they store high-level features of the objects [35].

2.4 Path planning

In order to avoid collisions during navigation, autonomous systems have to plan paths only in obstacle-free areas. A traditional way to avoid obstacles is to rely on grid maps, and during the last decades, numerous algorithms have been proposed for finding obstacle-free paths on maps. Chapter 2.4.1 introduces the basic versions of two famous algorithms, A* and rapidly exploring random tree (RRT), since variations of those algorithms are used in the state-of-the-art solutions presented in Chapter 3. The obstacle-free paths can be further processed to desired trajectories by considering also the dynamical limits of the system.

Some solutions forgo maintaining a map and use machine learning to plan trajectories straightly based on the latest sensor data. One of the state-of-the-art solutions presented in Chapter 3 relies on its vision-based navigation to a method called privileged learning, which is based on decomposing imitation learning problem into two parts. Chapter 2.4.2 presents the basic theories behind imitation learning and privileged learning.

2.4.1 Traditional path planning algorithms

A* is a graph search algorithm finding a path from a start node to a goal node in a graph, where nodes represent possible states and edges between the nodes represent the ability to move between the nodes [36]. The edges typically have a positive cost. A* finds efficiently the optimal path with minimum cost. The algorithm keeps up a sorted list of nodes that have not been explored yet and a list of nodes that have already been explored. The algorithm also needs information on the edges and equivalent costs. For every node, the minimum cost from the start to the node found is stored. The parent node preceding the node in the shortest path found is also stored.

The search is started by initializing the open list with the start node. The minimum cost of reaching the start node is initialized to 0. The cost of all other nodes is initially infinite. At each iteration, the first node from the open list is removed, marked as the current node, and added to the list of closed nodes. Since the list of open nodes is sorted, the first node of the list is always the one where the estimated cost to the goal is the smallest.

The estimated cost is calculated by the sum of the minimum cost of reaching the node and a heuristic, which is an underestimated cost of reaching the goal from that node. A typical value for heuristics in motion planning tasks is the straight-line distance to the goal. If the current node is the goal, the search is terminated, and the path is constructed from the parent links and returned. If the current node is not the goal, a tentative minimum cost for every non-closed neighbor node is calculated by a sum of the cost of reaching the current node and the cost of going from the current node to that neighbor node. If the tentative cost is smaller than the previous cost of reaching the neighbor, the found path is the new shortest path to that node. In such a case, the tentative cost is set to the cost of the neighbor, the current node is marked as the parent node of the neighbor node, and the neighbor is added to the list of open nodes. In case the neighbor already existed in the open list, it is moved

in the priority order based on the new cost. After going through all neighbors, the algorithm returns back to the main loop and removes again the first node from the list of open nodes marking, and marks it as the current node. If the list of open nodes is empty, there is no solution. As long as the heuristic is not overestimating the cost, the path returned by A* is guaranteed to be the optimal one.

When A* search is performed in 3D grid maps, the cost of the edges between the nodes is typically the distance between the cells

$$\text{cost} = \sqrt{d_x \cdot d_x + d_y \cdot d_y + d_z \cdot d_z} \quad (11)$$

where d_x , d_y , and d_z are the distances in x-, y-, and z-directions respectively. The neighbor cells are added to the list of open nodes only if they are obstacle-free.

RRT was first proposed by LaValle [37]. RRT belongs to the class of sampling-based path planners, which rely on a random or deterministic function to choose a sample from the state space [36]. Sampling-based methods give up on finding the optimal solution but aim to find a satisfactory solution quickly. The sampling-based algorithms are building a tree representing the feasible motions of the robot.

The RRT algorithm searches for a collision-free path from a start state to a set of goal states. The basic version of RRT grows a single tree from the start state and stops when the path to the goal is found or the maximum tree size is reached. On every iteration, one state is sampled randomly from an almost-uniform distribution over the given state space. The sampling is slightly biased towards the goal states. For every sampled state, the nearest state in the tree is searched by minimizing the Euclidean distance. A new state candidate to possibly be added to the tree is selected by choosing a state a small distance from the nearest state towards the sampled state. After that, a local planner is used to find a path from the nearest state to the new state candidate. Since the distance is small, a typical choice of the local planner is one that returns a straight path. If the path from the nearest state is collision-free, the new state candidate is accepted to the search tree.

In addition to the basic version of the RRT, a wide variety of modified versions of the algorithm have been proposed. Typical things that are modified are the sampling, the way to define the nearest node after the sampling, and the local path planner from the nearest state to the new state candidate.

2.4.2 Machine learning based methods

Instead of traditional path planning methods, also machine learning methods can be used for vision-based navigation. Imitation learning is a method where the agent tries to learn based on training data generated by an expert [38]. In autonomous navigation, the data contains the trajectories generated by the expert, along with the corresponding sensory input. According to Osa et al. [39, pp. 9-10], imitation learning is closely related to the better-known supervised learning, and particularly related to structured prediction, where the aim is to learn a mapping from input data to a complex and structured output. The key differences that distinguish it from supervised learning are: 1) in imitation learning the solution may have structural properties including constraints, stability, and dynamic smoothness, or leading to

a coherent multi-step plan; 2) the interaction between the learner’s decisions and its own input distribution; and 3) the increased need of minimizing the high cost of gathering examples.

Imitation learning is also related to reinforcement learning [39, pp. 11]. In reinforcement learning, the aim is to obtain a policy that maximizes an expected reward, and a reward function is used for encouraging a desired behavior. In imitation learning, good demonstrations from an expert are available for providing prior knowledge. That knowledge can drastically decrease the sample complexity of a learning task when compared to trial-and-error reinforcement learning. The comprehensive definitions of supervised learning and reinforcement learning are beyond the scope of this thesis, and the reader is referred to the literature for further details [40, 41].

Vision-based navigation with privileged learning, first proposed by Chen et al. [38], is based on decomposing the learning of vision-based navigation into two stages. In the first stage, the agent is trained to imitate the expert with access to perfect information about the simulated environment. In the second stage, another agent is trained to imitate the first agent. The second agent does not have privileged information about the environment but is relying purely on onboard sensing. The decomposition of the learning process is beneficial since the vision-based navigation can also be decomposed into two learning subprocesses: learning to see and learning to act. The first agent has perfect information about the environment and hence it can learn optimal actions faster and generalize them better. During the training of the second agent, the privileged agent can provide high-capacity on-policy supervision, and the second agent can concentrate on learning to perceive the environment.

3 Autonomous flying inside a forest

In this chapter, a literature survey on existing open-source methods for autonomous flying in GNSS-denied environments is presented. Based on the literature survey in Chapter 3.1, the most promising solution for forest research purposes is proposed in Chapter 3.2. The proposed method is chosen as the base of the system implemented and tested in the following chapters. In this literature survey, only solutions that have the source code openly available for research purposes are considered. The system has to be flight-tested in a real forest and published in a peer-reviewed scientific publication.

3.1 State-of-the-art solutions

3.1.1 Campos-Macías et al.

Campos-Macías et al. [42] have presented a complete framework for an autonomous small-sized drone that was successfully flight-tested in a previously unknown real forest environment. The source code of the framework is openly available¹. The framework is divided into three main elements: mapping, path planning, and trajectory generation. Their solution uses VIO for estimating the position and the orientation of the drone.

The drone is creating a voxel map of its environment representing the probability of occupation of the cells. The map is implemented as a linear octree, and it is updated based on the sensed point cloud. The point cloud is created based on the disparity image received from a forward-facing stereo camera. Detected points are first grouped into voxels. After grouping, based on the log-odds of the occupancy of the voxel at the last timestamp, the log-odds of the prior, and the log-odds of the inverse sensor model, the probability of being occupied is updated to all voxels that lie inside the field of view of the camera.

At the path planning stage, the drone generates a path inside the unoccupied space of the map using a variation of the RRT-connect algorithm [43]. RRT-connect is an extension of RRT, where two trees are incrementally built from the start and the goal.

At the trajectory generation stage, the drone generates a trajectory so that it flies from its current state to the next planned state given by the path planner [42]. The trajectory is generated so that the dynamic constraints of the drone, maximum separation from the path, and maximum deviation from the desired orientation are all considered. The trajectory generation consists of two components. The first component induces an asymptotically stable equilibrium point and invariant sets around that point. The second component moves that equilibrium point along the path so that the state of the drone always satisfies the constraints for the separation from the path and deviation from the desired orientation.

Campos-Macías et al. [42] compared the performance of their framework against two earlier methods with open code by Gao et al. [44] and Usenko et al. [45]. The experiments were performed in several simulated Poisson forest environments. In

¹<https://github.com/IntelLabs/autonomoussmavs>

these experiments, their framework clearly outperformed the earlier methods both in success rate and flying velocity. Their framework outperformed the earlier solutions also in terms of computation. Earlier works have required either a powerful CPU or even GPU to run in real-time, but the framework proposed by the authors used only a low-power Intel Atom CPU to run everything onboard. Their framework used Intel RealSense depth camera D435 [46] for mapping and obstacle avoidance, and Intel RealSense Tracking Camera T265 [47] for VIO. The used drone platform was Intel Aero Ready to Fly drone [48], which had a 360 mm frame and weighed 865 g without a battery. Real-world experiments were successfully performed in a 3D maze, a warehouse, a forest, and a lab with walking people acting as dynamic obstacles. The flight distances in the real-world experiments varied between 4.81 m and 28.68 m, and the average flight velocities varied between 0.15 m/s and 0.20 m/s. In the forest flight test, the flying distance was 11.58 m and the flying velocity was 0.20 m/s in a relatively sparse forest without low branches.

3.1.2 Loquercio et al.

Loquercio et al. [49] have proposed another solution for the autonomous navigation of drones at very high speeds with purely onboard sensing and computation. In their solution navigation is not divided into separate sensing, mapping, and planning subtasks. The subtasks are replaced with a single neural network (NN) trained in a simulated environment instead. That decreases the processing latency and prevents errors from compounding through the pipeline. The source code of the proposed solution is openly available².

The NN used for navigation is trained using privileged learning. A sampling-based motion planning algorithm is used as the privileged expert. The expert has a perfect knowledge of the platform state and a complete 3D map of the simulated environment and it generates a set of collision-free trajectories representing the desired state of the drone over the following second. Generated trajectories are then used to train the student policy that has access only to onboard sensor measurements. The student policy is represented as a neural network with two branches. The network produces a latent encoding of visual, inertial, and reference information and outputs three trajectories with their respective collision costs. Those three trajectories are projected on the space of polynomial trajectories and the trajectory with the lowest predicted collision cost is selected for execution and tracked by a model-predictive controller [50]. If multiple trajectories have similar predicted collision costs, the one with the lowest actuation cost is selected.

The training is done entirely in simulation. In their own experiments, Loquercio et al. [49] built the simulation environments by spawning either simulated trees or convex shapes in an off-the-shelf Unity³ environment. Totally 850 training environments were created by spawning either of two categories of items according to a homogeneous Poisson point process with varying intensity. For each environment, a global collision-free trajectory from starting point to goal was calculated for training the privileged

²https://github.com/uzh-rpg/agile_autonomy

³<https://unity.com/>

expert. The student policy was only provided with a straight trajectory from start to goal.

Loquercio et al. [49] compared the performance of their system against two state-of-the-art approaches. The approaches used as baselines were a reactive planner by Florence et al. [17] and the Fast-Planner, which is a mapping and planning-based method by Zhou et al. [51]. The comparison was made in four different simulated environments: forest, narrow gap, disaster, and city street. The proposed system outperformed the baselines in all environments, especially with high-flying speeds.

The performance was measured according to the success rate of missions. The system was also tested in different real-world scenarios by deploying the policy trained in simulation to a custom-built physical drone. The drone used RealSense D435 for acquiring the depth images and Intel RealSense T265 for state estimation. According to the supplementary materials of the original article [49], the drone platform had Armattan Chameleon 6 frame and weighed 890 g. Motor-to-motor dimension of the frame was 252 mm [52]. Real-world tests were carried out in various natural and urban environments, including for example different forests, buildings, ruins, and a narrow gap. In all experiments, the drone was provided with a non-collision-free reference trajectory and the flight speed varied between 3 and 10 m/s. Until the speed of 5 m/s, the success rate of the flights was 100%, and with a flight speed of 10 m/s, it was still 60%.

The proposed system was also compared against a commercial drone Skydio R1 [53] in a scenario where the task was to follow a person running through a narrow gap. The experiment was performed three times with Skydio and it failed to pass through the gap in all of those. The same experiment was performed six times with the proposed system, and it successfully flew through the gap in all of the performed experiments.

3.1.3 Liu et al.

The system proposed by Liu et al. [54] is targeted purely at large-scale autonomous flying inside a forest. The system is built upon the one proposed by Mohta et al. [55, 56] with a few improvements. The system uses a modified version of the real-time semantic SLAM algorithm, SLOAM, proposed by Chen et al. [57] to build a semantic map of the environment. The semantic objects used include tree trunks and ground planes. The source codes of both the autonomous flight stack⁴ and the semantic SLAM algorithm⁵ are open for research purposes.

The architecture of the proposed system has five modules. The first module contains inputs from IMU, 3D LiDAR, and a stereo camera. The module also performs a semantic segmentation on Point Clouds obtained from LiDAR. The segmentation is performed with RangeNet++ [58]. The second module converts the semantic data into constraints on both the robot poses and the landmark models. Those constraints are used by SLOAM to optimize the pose of the drone and the landmarks. The output of VIO algorithm [59] is given as an initial guess of the pose

⁴https://github.com/KumarRobotics/kr_autonomous_flight

⁵<https://github.com/KumarRobotics/sloam>

to SLOAM. The output of the SLOAM is used by a drift-compensation mechanism to reduce the odometry drift.

The third module is performing path planning with hierarchical mapping and planning framework. The top-level map consists of semantic models, the mid-level map is a large and coarsely discretized voxel map, and the low-level map is a small and finely discretized voxel map that is updated at the LiDAR rate. The high-level planner covers the user-defined region of interest. The mid-level global planner is used for generating the shortest collision-free path to the next coverage waypoint given by the high-level planner. The mid-level planner is based on the jump point search algorithm [60], which is an extension of A* for uniform-cost grid environments. Instead of expanding every neighbor, the jump point search algorithm allows longer jumps on the grid map potentially reducing the running time. The low-level local planner generates safe and dynamically-feasible trajectories to the local goal, which is generated by finding the first intersection of the boundary of the local map and the global path [54]. The low-level planner uses a motion-primitive-based algorithm proposed by Liu et al. [61].

The trajectory generated by the local planner is forwarded to the fourth module, that is taking care of the control and trajectory tracking. Since the generated trajectory is dynamically feasible, the trajectory tracker assumes that the drone can track the planned trajectory tightly. By making that assumption, the motion-primitive graph built by the local planner can be reused making the re-planning significantly faster. Distance from the current position to the trajectory is also constantly monitored so that the system detects if the drone fails to track the trajectory due to large external disturbances. In case of failure, the drone is stopped, and a new re-planning process is started. A nonlinear geometric-based controller [62] is used as the position controller that calculates the desired orientation, body rate, and thrust. The desired values are forwarded to a flight controller that calculates the motor speeds with an attitude controller. The last module of the system contains the actuators (the motors) of the drone.

Liu et al. [54] tested the system both in simulated environments and in a real forest. Simulated environments were created with Unity. The simulator was also first used for training RangeNet++, which was further fine-tuned on real-world data. The simulated forests were generated based on real-world tree positions collected using LiDAR data from overhead flights. The tree models were obtained by customizing the size, shape, orientation, and undergrowth conditions of various tree models from the Unity asset store.

The first simulated experiment was about large-scale mapping. The simulated environment contained a total of 10 692 trees, and the task of the drone was to generate a semantic map. Without pose noise, the drone was able to fly ≈ 10 km without any human intervention, and the final tree count of the semantic map was 10 897. The second simulated experiment measured the impact of the tree density. Based on the tree positions from a real forest, ten simulated forests were generated with the same distribution but different densities. For each environment, four missions were executed. The increasing tree density led to decreasing flight speed and increasing traveled distance.

Real-world experiments were performed in a North American pine forest. In the first experiment, the mission was to perform two square loops and return home. The drone platform was built on a 450 mm drone frame. The positioning of the autonomy stack relied only on VIO, and the total trajectory length was ≈ 800 m. In the second experiment, the drone was manually piloted ≈ 1.1 km loop so that the takeoff and landing positions were the same. The results showed that combining SLOAM and VIO increased the position accuracy drastically compared to relying on only VIO. In the first experiment, the combination of SLOAM and VIO agreed with a noisy GPS signal within a 10-20 m margin, but relying on only VIO caused ≈ 60 m drift. In the second experiment, the total drift relying on only VIO was 10.10 m and with relying on SLOAM and VIO it was 3.99 m.

3.1.4 Zhou et al.

Zhou et al. [63] have proposed a solution that has been successfully utilized for autonomously navigating swarms of drones through bamboo-forest. In addition to flight trajectory planning and obstacle avoidance, the planner can also avoid inter-robot collisions and coordinate the swarm formation. The planned trajectories are represented as MINCO [64] piece-wise trajectories. The source codes of both the solution itself ⁶ and MINCO⁷ are openly available for research purposes.

The proposed planner follows a goal-chasing scheme, so the goals can be given at any time during the tasks by either the user or a higher-level planner. The planner then selects a local target based on the predefined local planning distance along the direction of the global goal. The paper did not specify the used path planning algorithm, but based on the source code, the system uses A*. The trajectory is optimized by minimizing the penalty function iteratively. The penalty function consists of penalty terms for smoothness, total time, dynamical feasibility, obstacle avoidance, swarm collision avoidance, swarm formation, and optionally multiview tracking of a participant. Within each iteration, the solver returns a solution trajectory. The total penalty and the gradients of the polynomial coefficients of the trajectory are calculated and sent back to the solver. After optimization, a local trajectory is returned and executed. After a defined period or whenever the trajectory collides with a newly sensed obstacle, the planning is reactivated and a new local target is defined. This procedure is repeated until the global goal is reached.

The drones are sharing their trajectories with each other via the trajectory-broadcasting network. The system is using VIO for localization and the drift is corrected with help of onboard relative distances acquired with ultra-wideband (UWB) sensors. The VIO algorithm used by the system is VINS-Fusion [65][66]. Mapping is based on probabilistic occupancy grid maps.

Zhou et al. [63] compared the proposed system in simulation against two state-of-the-art swarm planners. The planners used as benchmarks were MADER [67] proposed by Tordesillas et al. and their own previous work EGO-Swarm [68], which is a swarm extension of the original EGO-planner [69]. Simulated tests were run in

⁶<https://github.com/ZJU-FAST-Lab/EGO-Planner-v2>

⁷<https://github.com/ZJU-FAST-Lab/GCOPTER>

two different scenarios: six drones flying through a narrow gate and ten drones flying across obstacles. In the obstacle avoidance scenario, desired velocities varied from 1 m/s to 3 m/s and average obstacle spacing varied from 2 m to 1 m. The proposed planner showed smaller total jerk than benchmarks with comparable flight times. The proposed algorithm and EGO-Swarm had comparable minimum clearance, but MADER did not manage to avoid crashes with a flight speed of 3 m/s. The proposed method was also computationally the fastest one.

The proposed system was also tested with real palm-sized drone hardware with a motor-to-motor distance of 114 mm. The system used Intel Realsense D430 depth camera module [70] and IMU for mapping and localization and UWB modules for drift correction. In the first experiment, the drone swarm was able to fly 65 m forward through a dense bamboo forest (the density was not specified). The desired flying speed of the swarm was 1 m/s. The second experiment was about formation navigation in the wild, and the drone swarm was successfully keeping the preferred moving shape when flying through a less dense forest than in the first experiment. The third experiment tested flying in random directions. 10 drones had goals on a 3-m-radius circle and successfully avoided inter-robot collisions. In the fourth experiment, the drone swarm was successfully tracking and videoing a moving human target from multiple points of view and simultaneously avoided obstacles.

3.2 Proposal of the method

In the literature survey performed in subsection 3.1, four candidate systems with open-source codes were found. All of them were published in peer-reviewed scientific publications and successfully flight-tested in a real forest environment. However, none of them was flight tested in dense Northern European boreal forests. The forest tests of the framework proposed by Campos-Macías et al. [42] were performed in sparse deciduous forests. The system proposed by Loquercio et al. [49] was tested in multiple forest types, but based on the images from the tests, none of them seem very dense. Liu et al. [54] tested their system on a relatively dense North-American pine forest, but the environment under the canopy still seems quite sparse when compared to a dense spruce forest, since there are fewer low-level branches. Zhou et al. [63] tested their drone swarms in a Chinese bamboo forest. The bamboo forest was dense, but the amount of small low-level branches and understory vegetation was minimal, so the test environment was drastically different from a boreal forest.

The strength of the solution proposed by Campos-Macías et al. [42] is its computational efficiency. The solution was able to successfully navigate in an unknown forest environment relying on onboard visual sensing with a low-power CPU. The weakness of the system is the very slow flight speed. In the performed experiments the trajectory lengths were also very short.

The solution proposed by Loquercio et al. [49] had an amazing performance in high-speed flying. The system is not saving the maps of the environment but instead concentrates on flying as fast as possible. However, the high flying speeds can not be applied in data collection for forest research purposes, since too fast a flying speed makes the recorded videos and images blurry. The lack of built-in mapping is also a

small drawback since produced 3D maps could potentially contain useful data from the point-of-view of forest research.

The solution proposed by Liu et al. [54] is the most complete framework for forest research since it is purposed for mapping large-scale forest areas autonomously. It is the only open-source solution found in the literature that has the highest-level planner taking care of coverage planning directly included in the system. The weakness of that system is its dependency on LiDAR. In theory, 3D LiDAR can be replaced with any other sensor providing a 3D point cloud. However, in the project wiki [71], the authors mention that they have tried using stereo depth estimation algorithms to compute the map, but the computational burden turned out to be too high to run in real-time with the rest of the navigation stack. The need for LiDAR makes the system larger than with the other solutions, which is a big drawback in forests with low branches and dense understory vegetation.

The solution proposed by Zhou et al. [63] was capable of flying to a global goal through a bamboo forest based on only onboard visual perception. The test forest was denser than in the other systems, but being a bamboo forest, it had a minimal amount of low branches. The system was running on a very small and light drone and uses the occupancy grid maps for navigation. The system has been tested only in swarm navigation. The system uses relative distances between the drones for drift correction, so likely the localization performance is not as good when using only one drone. However, the earlier version of the system [69] has been successfully flight-tested without the swarm drift correction, so the performance of the system could be satisfactory also with only one drone. The system was also the only solution that performed obstacle avoidance and VIO with the same camera.

The summary of the main properties of the solutions is presented in Table 1. The summary contains the path planning method, used sensors, onboard computer, and the longest autonomous test flight in the forest reported in the paper.

Based on the literature survey, the solutions by Loquercio et al. [49] and Zhou et al. [63] seem to be the most suitable ones for the desired purpose. The solution proposed by Campos-Macías et al. [42] has only been validated in very short and slow flights in sparse forests, and the solution by Liu et al. [54] is dependent on 3D LiDAR making it larger than the other solutions.

From these, the solution by Zhou et al. [63] was chosen as the basis for development, mainly due to its ability to rely on only one camera and its proven ability to fly through a dense forest. Even though it has not been tested at high flying speeds, the documented swarm flying velocity of ≈ 1 m/s is enough for data collection in forest research. The possibility of extending the system to swarm operations is also an advantage from the point of view of future forest research applications.

Table 1: Summary of the main properties of the solutions.

Reference	Main properties
Campos-Macías et al.	<ul style="list-style-type: none"> • Occupancy grid map, path planning based on modified RRT-connect, and custom dynamically feasible trajectory generation algorithm • Point cloud from D435, VIO from T265 • Intel Atom CPU x7-Z8750 • Test flight of 11.58 m with an average velocity of 0.20 m/s in sparse forest • 360 mm frame
Loquercio et al.	<ul style="list-style-type: none"> • No environment mapping, trajectory generation based on a pre-trained NN • Depth images from D435i, VIO from T265 • NVIDIA Jetson TX2 • Multiple 40 m test flights in relatively sparse forests. 100 % success rate with 5 m/s average velocity. • 252 mm frame
Liu et al.	<ul style="list-style-type: none"> • Large-scale semantic map, coarse mid-level occupancy grid map, and fine local occupancy grid map. The high-level planner performing area covering, the mid-level planner generating paths with the jump point search algorithm, and the local planner generating safe and dynamically-feasible trajectories with a motion-primitive-based algorithm. • SLAM algorithm using 3D LiDAR, stereo camera, and IMU • Intel NUC with 12-thread i7-10710 U CPU • Test flight of 800 m consisting of two square loops relying only on VIO in a relatively dense forest without low branches. No collisions, but the VIO drifted ≈ 60 m. The proposed SLAM algorithm reduced drifting drastically, but was not tested as a pose estimate in real-world autonomous flights. • 450 mm frame
Zhou et al.	<ul style="list-style-type: none"> • Occupancy grid map, path planning based on A* and MINCO • Depth images for mapping from D430, VIO by VINS-Fusion with IMU and stereo images from D435 • NVIDIA Xavier NX • 65 m swarm flight through a dense bamboo forest • 114 mm frame

4 Selected method for autonomous flying inside a forest

This chapter provides a detailed description of the solution proposed by Zhou et al. [63] that was chosen as a base of the implemented system based on the literature survey in Chapter 3. The original research paper did not give a name for the method, but the GitHub repository is named EGO-Planner-v2. From now on, that name is used for the method.

The system architecture of the proposed solution is shown in Figure 4. In the following subchapters, essential modules of the solution, namely localization, mapping, planning, and tracking controller, are presented. Since this thesis focuses on the navigation of individual drones, the modules related to swarm operations (trajectory broadcast network, drone removal, and drift correction) are omitted here. The system architecture after omitting the swarm operation modules is presented in Figure 5.

4.1 Middleware

EGO-Planner-v2 is running on top of the Robot Operating System (ROS). ROS is an open-source meta-operating system for robots [72]. ROS provides the services normally provided by an operating system such as hardware abstraction, message-passing between processes, package management, implementation of commonly-used functionality, and low-level device control. ROS also provides libraries and tools for writing, building, and running code across multiple computers.

The Computation Graph of the ROS system is the peer-to-peer network of processes processing data together [73]. ROS system consists of ROS Nodes that are independent processes performing computation. Nodes communicate by messages. ROS messages are data structures containing a field with a predefined data type. Nodes send messages by publishing to a ROS topic that is a name to identify the messages. Nodes can receive data published by other nodes by subscribing to the appropriate topic. In addition to publish/subscribe model by messages, ROS Nodes can communicate by request/reply paradigm via services. ROS Services are defined by a message structure for both request and reply messages.

Name registration and lookup to the Computation Graph are provided by ROS Master [73]. Without ROS Master, nodes would not be able to find each other or communicate. ROS Master contains also a ROS parameter server, that allows data to be stored. The data published to ROS topics can also be recorded to bag files [74]. In bag files, message data from selected topics are stored and serialized as it was received. Bag files can then be played back in ROS making it possible to use recorded data in the future. In drone applications, `mavros` package can be used for communication between autopilot and the ROS system [75]. The communication works with MAVLink communication protocol [76].

In addition to the ROS system running on the onboard computer of the drone, the PX4 autopilot is performing middleware operations. PX4 is a modular open-source autopilot software, consisting of two main layers: the flight stack layer for estimation and flight controlling and the middleware layer [77][78]. The middleware layer of

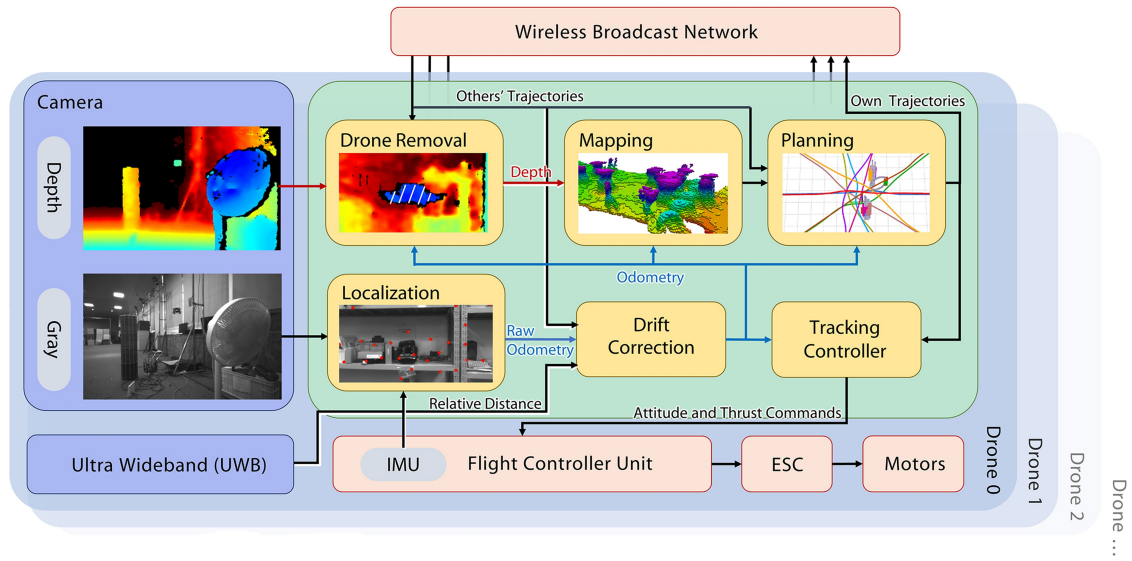


Figure 4: The system architecture of the solution proposed by Zhou et al. [63] (EGO-Planner-v2).

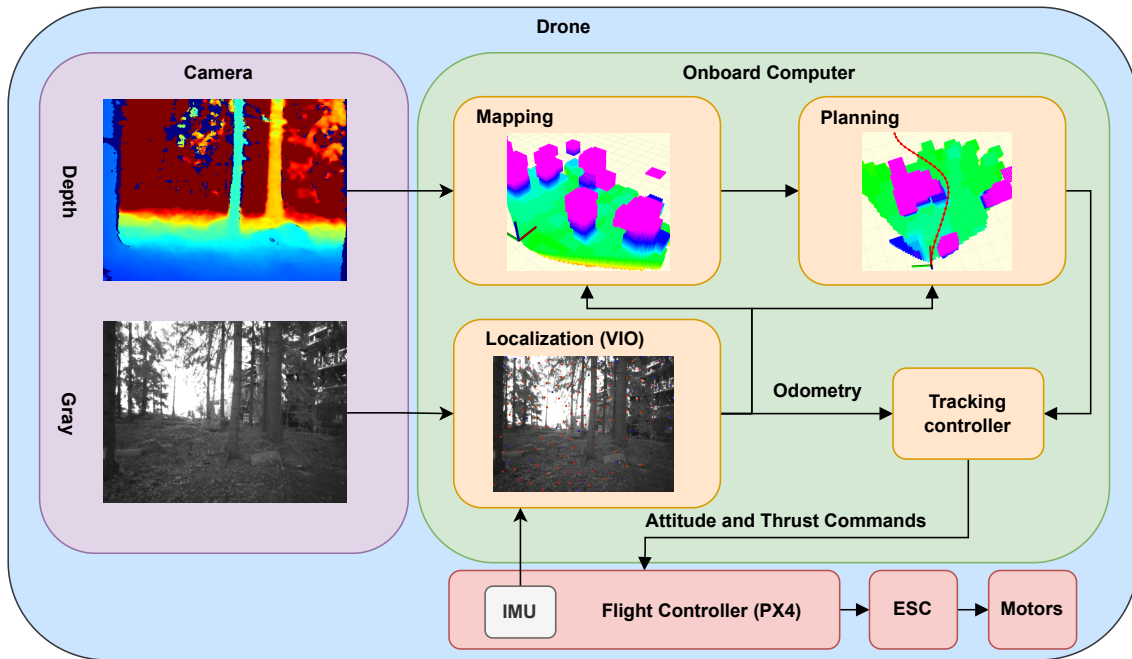


Figure 5: The system architecture used in this thesis.

the PX4 consists of drivers for sensors, the uORB publish-subscribe message bus, and communication with external devices. Examples of external devices are remote controllers and the onboard computer.

4.2 Localization

EGO-Planner-v2 used VIO with a stereo camera and an IMU for estimating the pose of the drone. In the swarm operations, the system used also UWB sensors to measure relative distances between the drones. UWB readings were used in a drift-correction algorithm to avoid inter-robot collisions in long-range operations. Since this thesis focuses on the navigation of individual drones, the drift correction and UWB sensors are omitted here.

As VIO-algorithm, Zhou et al. [63] used VINS-Fusion⁸. VINS-Fusion is an extension of VINS-Mono [65]. VINS-Mono supports only monocular camera and IMU, but VINS-Fusion is a general framework for fusing multiple sensors in pose estimation [66].

Full utilization of VINS-Fusion supports multiple features that are not used in this thesis. Since the scope of this thesis is restricted to flying to a local goal through a forest, features related to relocalization based on loop closures, global optimization, and reusing the maps are omitted here.

In VINS-Fusion, every sensor is treated as a factor, that provides a measurement [66]. Factors share common state variables and are summed together to build the optimization problem. For every new stereo image pair, the system detects Shi-Tomasi corner features. Uniform distribution of the features is enforced by setting a minimum distance between two features [65]. The features are matched between the stereo images and tracked in the previous frame by KLT tracker [79].

In a sensor setup with a stereo camera and IMU, the inertial data is collected at a higher frequency than stereo images, and therefore there exist multiple IMU measurements between two frames. VINS-Fusion uses quaternion-based IMU preintegration to avoid the need for repropagation of IMU measurements after every pose adjustment during the optimization [65]. In IMU preintegration, position, velocity, and orientation are integrated from IMU measurements between two frames. Preintegration terms are obtained with IMU measurements by using accelerometer and gyroscope biases modeled as random walks, whose derivatives are Gaussian white noise. IMU propagation is a computationally demanding process, thus the IMU preintegration reduces the computational demand of optimization-based algorithms.

VINS-Fusion is initialized by loosely aligning vision-only structure and IMU preintegration values [65]. VINS-Fusion uses a sliding-window optimization-based framework for state estimation [66]. The state vector contains positions and orientations of the body expressed in the world frame, the depth of each feature observed in the first frame, and the IMU variable composed of velocity and biases of the accelerometer and the gyroscope. The optimization is performed by minimizing a non-linear cost function by Gauss-Newton or Levenberg-Marquardt approaches. Implementation of VINS-Fusion uses Ceres solver [80] for the optimization. VINS-Fusion uses also marginalization to convert old measurements to a prior term to bound the computational complexity of the system.

In autonomous flying, state estimates are needed in high frequency. VINS-Fusion uses IMU forward propagation between the frames to achieve IMU-rate state

⁸<https://github.com/HKUST-Aerial-Robotics/VINS-Fusion>

estimation, which can be used as state feedback for the closed-loop closure [65]. VINS-Fusion also adopts an online temporal calibration method for inertial and visual measurements [81], which can be used for estimating the time offset between the camera and the IMU online.

4.3 Mapping

The mapping of the system is based on probabilistic occupancy grid maps [32]. The mapping module uses depth images produced by the RealSense D435 as an input. The original article by Zhou et al. [63] did not present the mapping module, but based on the source code the system adopts the fixed-sized circular buffers for maintaining the local map as proposed by Usenko et al. [82]. A circular buffer makes maintaining the map computationally efficient, but due to the fixed size of the circular buffer, only the current local information in front of the drone is stored on the map, and old information is cleared when the system moves.

The maps also have a virtual floor and a virtual ceiling. The virtual floor defines the minimum altitude of the grid map, and the virtual ceiling defines the maximum altitude of the grid map. Virtual floor and virtual ceiling can be used for restricting the altitude where the system is allowed to plan paths.

4.4 Path planning

Quadrotors are differentially flat systems, which means that the inputs and the states can be presented as algebraic functions of selected flat outputs and their derivatives [83]. The proposed system uses a trajectory representation called MINCO (minimum control) [64] designed for differentially flat systems. Due to the scope of this thesis, and limited paper length, only a very brief introduction to the main details of MINCO is presented here. The reader is referred to the supplementary materials of Zhou et al. [63] for further details of the trajectory optimization and to the original MINCO article by Wang et al. [64] for detailed proofs.

Generation of a MINCO trajectory utilizes a sampling-based or search-based global path planning method to generate a low-dimensional collision-free path that is then further optimized [64]. Optimization generates a dynamically feasible trajectory that is homotopic to a given low-dimensional path. Unlike sampling-based or search-based methods, the generation process of MINCO trajectories is designed to flexibly incorporate system state-input constraints. In this way, the complexity of both the system dynamics and the environment is divided and conquered. The original EGO-Planner-v2 article by Zhou et al. [63] did not contain any mentions of the chosen global path-searching algorithm. Based on the source code, the system uses A*.

MINCO trajectories are piece-wise trajectories where the space and time parameters of the trajectory are decoupled. In the supplementary materials of the original

article, the definition of the MINCO polynomial trajectory class is presented as

$$\text{MINCO} = \left\{ p(t) : [0, t_M] \mapsto \mathbb{R}^m \mid \mathbf{c} = \mathcal{M}(\mathbf{q}, \mathbf{T}), \right. \\ \left. \mathbf{q} \in \mathbb{R}^{m \times M-1}, \mathbf{T} \in \mathbb{R}_{>0}^M \right\} \quad (12)$$

where $p(t)$ is an m -dimensional N -degree polynomial trajectory consisting of M -pieces. The degree is defined as $N = 2s - 1$, where s is the order of the integrator chain of the system model. $\mathbf{c} = (\mathbf{c}_1^T, \dots, \mathbf{c}_M^T)^T \in \mathbb{R}^{2Ms \times m}$ is the polynomial coefficient matrix where $\mathbf{c}_i \in \mathbb{R}^{2s \times m}$. The parameters of MINCO trajectory are the intermediate waypoints $\mathbf{q} = (\mathbf{q}_1, \dots, \mathbf{q}_{M-1})$ between connected pieces and time durations $\mathbf{T} = (T_1, T_2, \dots, T_M)^T$ of the pieces. Total time t is defined as $t_i = \sum_{j=1}^i T_j, i \in [1, M]$, and $t_0 = 0$. Notation \mathcal{M} is the mapping between polynomial coefficients \mathbf{c} and MINCO parameters \mathbf{q} and \mathbf{T} . The i -th piece $p_i(t) : [0, T_i] \mapsto \mathbb{R}^m$ is defined by

$$p_i(t) = \mathbf{c}_i^T \beta(t), \quad \forall t \in [0, T_i] \quad (13)$$

where $\beta(t) := [1, t, \dots, t^N]^T$ is the natural basis. Figure 6 presents an instance in MINCO and its parameters.

MINCO converts given parameters $\{\mathbf{q}, \mathbf{T}\}$ to polynomial parameters \mathbf{c} and time profile \mathbf{T}_p with a linear complexity $O(M)$ [63]. The correspondence can be more specifically expressed as

$$\mathbf{M}(\mathbf{T})\mathbf{c} = \mathbf{b}(\mathbf{q}), \mathbf{T}_p = \mathbf{T} \quad (14)$$

where $\mathbf{b}(\mathbf{q}) \in \mathbb{R}^{2Ms \times m}$. For any $\mathbf{T} \succ 0$, $\mathbf{M}(\mathbf{T}) \in \mathbb{R}^{2Ms \times 2Ms}$ is a nonsingular banded matrix, and linear complexity of recovering a trajectory is achieved via banded PLU (permutation, lower-, and upper-triangular matrices) factorization. The partial gradients for the polynomial coefficients are also propagated to MINCO parameters in linear time, and then the optimization is directly applied to MINCO. The details of $\mathbf{M}(\mathbf{T})$, $\mathbf{b}(\mathbf{q})$ and the gradients needed by the numerical optimizer are omitted here but can be found in supplementary materials of Zhou et al. [63].

To be feasible, trajectories are required to avoid obstacles and fulfill the dynamical constraints of the vehicle [63]. Obstacle avoidance is achieved by deforming the shape of the trajectory, and due to differential flatness, dynamical constraints can be forced by restricting the magnitude of velocity, acceleration, and jerk. Constraints along the trajectory consist of infinitely many inequalities, and the proposed system uses a two-step procedure to convert the problem to an unconstrained one for more efficient solving. The system first enforces the constraints via integrals of penalty functions with large penalty weights and then evaluates the integrals by a finite sum of equally spaced samples along the timeline. The optimization of time-dependent objectives J

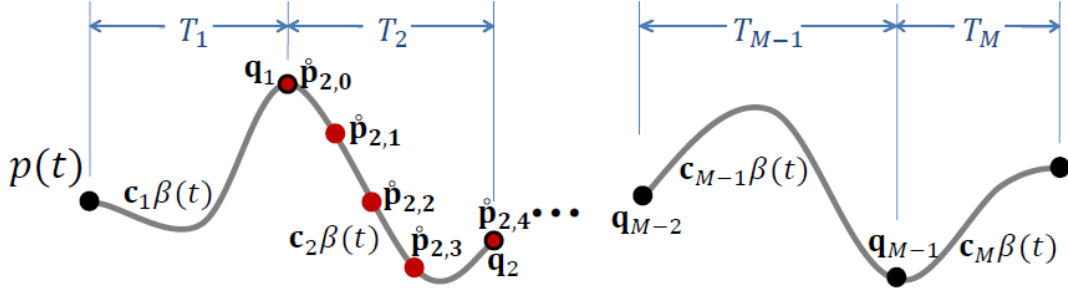


Figure 6: MINCO trajectory is parameterized with intermediate waypoints \mathbf{q}_i and time durations T_i . Red dots $\dot{\mathbf{p}}$ are constraint points. In the constraint points, constraints are evaluated and propagated to every \mathbf{q}_i and T_i . [63, supplementary materials]

under inequality constraints \mathcal{G} and equality constraints \mathcal{H} can be then presented as

$$\min_{\mathbf{q}, \mathbf{T}} \int_{t_0}^{t_M} J(\mathbf{q}, \mathbf{T}, t) dt \quad (15)$$

$$\text{s.t. } \mathcal{H}(\mathbf{q}, \mathbf{T}, t) = 0, \quad \mathcal{G}(\mathbf{q}, \mathbf{T}, t) \leq 0 \quad (16)$$

\Downarrow

$$\min_{\mathbf{q}, \mathbf{T}} \left(\int_{t_0}^{t_M} J dt + \int_{t_0}^{t_M} \|\mathcal{X}_H \cdot \mathcal{H}\|_2^2 + \max(\mathcal{X}_G \cdot \mathcal{G}, 0)^3 dt \right) \quad (17)$$

\Downarrow

$$\min_{\mathbf{q}, \mathbf{T}} \sum_{i=0}^{\kappa} \omega_i \cdot (J(t_i) + \|\mathcal{X}_H \cdot \mathcal{H}(t_i)\|_2^2 + \max(\mathcal{X}_G \cdot \mathcal{G}(t_i), 0)^3) \quad (18)$$

where \mathcal{X}_H and \mathcal{X}_G are the user defined weights. Arguments \mathbf{q} , \mathbf{T} , and t are omitted in equations 17 and 18 for simplicity. $t_i = t_0 + (t_M - t_0)i/\kappa$ are a timestamps of the samples, where $\kappa + 1$ is the sample number, and ω_i is the interval of integral evaluation. Zhou et al. [63] present the equation 18 also in a more accessible form

$$\min_{\mathbf{q}, \mathbf{T}} \sum_x \lambda_x J_x \quad (19)$$

where J_x are penalty terms and λ_x are corresponding weights. Subscripts $x = \{s, t, d, o, u\}$ stand for smoothness, total time, dynamical feasibility, obstacle avoidance, and uniform distribution of constraint points, respectively. In addition to these, the system has also penalty terms for swarm operations, but those are omitted in this thesis.

Solving the minimization in equation 18 would be a very complex problem with raw piece-wise polynomial trajectories due to the inevitably dense matrix inversion. However, linear-complexity operations of MINCO along with the compact parameter representation greatly reduce the convergence time.

EGO-Planner-v2 uses a jerk-control system model, so $s = 3$, and hence the control

effort optimization is given by

$$J_s = \int_{t_0}^{t_M} \|\ddot{\mathbf{p}}_t\|_2^2 dt \quad (20)$$

where $t \in [t_0, t_M]$ is the domain of the current trajectory and $\ddot{\mathbf{p}}$ is the jerk (a third derivative of the position). With MINCO trajectory representation, this integral can be analytically calculated.

The penalty function for total flight time minimizes the sum of times of each piece

$$J_t = \sum T_i \quad (21)$$

The dynamical feasibility of the trajectory is guaranteed by adding a penalty if the amplitude of the derivatives exceeds the thresholds

$$J_{d,v} = \sum_{i=0}^{\kappa} \max\{(\dot{\mathbf{p}}_{t_i}^\top \dot{\mathbf{p}}_{t_i} - v_m^2), 0\}^3 \quad (22)$$

$$J_{d,a} = \sum_{i=0}^{\kappa} \max\{(\ddot{\mathbf{p}}_{t_i}^\top \ddot{\mathbf{p}}_{t_i} - a_m^2), 0\}^3 \quad (23)$$

$$J_{d,j} = \sum_{i=0}^{\kappa} \max\{(\dddot{\mathbf{p}}_{t_i}^\top \dddot{\mathbf{p}}_{t_i} - j_m^2), 0\}^3 \quad (24)$$

$$J_d = J_{d,v} + J_{d,a} + J_{d,j} \quad (25)$$

where v_m , a_m , and j_m are the maximum values given for velocity, acceleration, and jerk respectively. Penalties $J_{d,v}$, $J_{d,a}$, and $J_{d,j}$ have different units but they share similar magnitude and hence direct sum is used.

The system adopts the obstacle modeling methodology from previous work by Zhou et al. [69]. Obstacles are modelled as planes $(\mathbf{x} - \mathbf{s})^\top \mathbf{v}$, $\mathbf{x} \in \mathbb{R}^3$. The obstacle side of the plane is treated as occupied and the other side of the plane as a free region. In the plane equation, $\mathbf{s} \in \mathbb{R}^3$ is a point on a plane and $\mathbf{v} \in \mathbb{R}^3$ is a normal vector pointing towards the free side of the plane. The obstacle avoidance penalty is presented as

$$J_o = \sum_{i=0}^{\kappa} \max\{(\mathcal{C}_0 - d_0(\mathbf{p}_{t_i})), 0\}^3 \quad (26)$$

where $\mathcal{C}_0 > 0$ is an obstacle clearance parameter, and d_0 is a point's distance to obstacles, defined as $d_0(\mathbf{p}_{t_i}) = (\mathbf{p}_{t_i} - \mathbf{s})^\top \mathbf{v}$.

Uniform distribution penalty penalizes the variance of the squared distances between each pair of constraint points $\dot{\mathbf{p}}$. The penalty aims to make the constraint points $\dot{\mathbf{p}}_{i,j}$ equally spaced for all $1 \leq i \leq M$ and $0 < j \leq \kappa$. The uniform distribution penalty is needed since in free optimization of MINCO parameters \mathbf{q} and \mathbf{T} some pieces of the trajectory tend to vanish. That phenomenon could cause problems since $T_i = 0$ would break the nonsingularity of MINCO. The non-uniform distribution of constraint points also increases the possibility of missing thin obstacles in collision avoidance constraint evaluation as shown in Figure 7.

The penalty is computed as

$$J_u = E[\mathbf{R}^2] - E[\mathbf{R}]^2 = \frac{1}{N_c} \|\mathbf{R}\|_2^2 - \frac{1}{N_c^2} \|\mathbf{R}\|_1^2 \quad (27)$$

where $N_c = \sum_{i=1}^M \kappa_i$ is the total number of distances and $\mathbf{R} \in \mathbb{R}^{N_c}$ is defined as

$$\mathbf{R} = (\|\dot{\mathbf{p}}_{1,1} - \dot{\mathbf{p}}_{1,0}\|_2^2, \dots, \|\dot{\mathbf{p}}_{M,\kappa_M} - \dot{\mathbf{p}}_{M,\kappa_M-1}\|_2^2) \quad (28)$$

The trajectory planning procedure runs as follows. In step 1, the global goal position is given to the planner. In step 2, the planner selects a local target in the direction of the goal within a predefined planning distance. Then the system starts the iteration with an initial guess of the path to that local target. In step 3, the solver returns a solution trajectory of the optimization. In step 4, the total penalty and the gradients of the penalties are calculated and sent to the solver before returning to step 3. In step 5, the planner returns a local trajectory that satisfies the task requirements, and the system executes it. Step 6 is executed when the trajectory collides with a new object or always after a given period. There the system returns back to step 2 and the planning is reactivated. The procedure is repeated until the system reaches the given global goal. EGO-Planner-v2 uses L-BFGS solver⁹ for optimization. The article did not describe the method of how the local target is selected in step 2, but based on the source code, the system plans a global minimum trajectory path from the start to the goal after step 1, and local targets are selected from that path.

Since the system transfers the safety constraints to penalties, a post-check after trajectory planning is necessary to ensure feasibility [63]. If a safety constraint is violated in a post-check, the planner increases the corresponding weight and makes another trial. If the trajectory remains infeasible after several trials, the current plan is terminated. After 10 ms the planner activates new replanning. Since the map is not static, the post-check guarantees the feasibility only at a certain time point. For that reason, a safety checking process continuously checks collisions in the background. If the safety process detects a safety violation, it activates replanning immediately. If that replanning process fails, and the predicted time to collision is under a given threshold, an emergency stop is activated. After the system has stopped, the planning process tries to start again.

4.5 Trajectory tracking

The original article by Zhou et al. [63] did not describe the algorithm used in the tracking controller shown in Figure 4. However, in the discussion¹⁰ in the EGO-Planner-v2 Github repository the author suggested using a package named "px4ctrl". The package is openly available for research purposes, and it is included in the source code¹¹ of Zhejiang University's drone building course.

⁹<https://github.com/ZJU-FAST-Lab/LBFGS-Lite>

¹⁰<https://github.com/ZJU-FAST-Lab/EGO-Planner-v2/issues/4>

¹¹<https://github.com/ZJU-FAST-Lab/Fast-Drone-250>

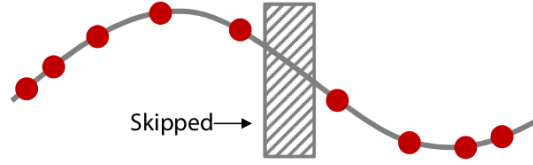


Figure 7: Non-uniform distribution of constraint points leads to missing a thin obstacle. [63, supplementary materials]

The trajectory tracking package acts as an interface between the path planner and the used PX4 flight controller. In addition to the trajectory tracking controller, the package contains a finite state machine for different system states. The structure of the state machine is visualized in Figure 8.

After reboot, the system starts from `MANUAL_CTRL` state. The takeoff command for the drone is given by the user. After the takeoff command is triggered, the system checks that the drone is still on the land, it is receiving odometry data, and it is not receiving commands from the path planner yet. If the conditions are fulfilled, the system moves to `AUTO_TAKEOFF` state and sends an arming command to PX4. PX4 performs its own pre-flight checks, which are related for example to sensor calibrations. If the PX4 approves the arming, the drone starts the takeoff. When the system reaches a pre-defined takeoff altitude, it moves to `AUTO_HOVER` state, where the controller tries to keep the drone still in the air. From the hovering state, the system moves either to `CMD_CTRL` state, if the path planner starts to send navigation commands, or to `AUTO_LAND` state if the user sends a landing command. If the system moves to `CMD_CTRL` state, it remains there as long it receives new commands from the path planner and then returns to `AUTO_HOVER` state. From the `AUTO_LAND` state, the drone moves back to `MANUAL_CTRL` state when the system detects landing to be successfully performed. If the visual odometry is lost, the system moves back to `MANUAL_CTRL` state from every state.

The package also has the option to connect a remote controller to the state machine. In that case, there would be more conditions for transitions, and moving from `AUTO_HOVER`, `CMD_CTRL`, and `AUTO_LAND` to `MANUAL_CTRL` would be possible by the controller.

The tracking controller of the `px4ctrl` package takes as an input the desired values for position, velocity, acceleration, and yaw-angle produced by the path planner. The planner outputs a quaternion containing a setpoint for the attitude and a scalar setpoint for the thrust. The summary of the whole controller architecture of the system is visualized in Figure 9.

At first, the controller calculates a setpoint for acceleration by the following equation:

$$\ddot{\mathbf{p}}_{sp} = \ddot{\mathbf{p}}_{des} + \mathbf{K}_v(\dot{\mathbf{p}}_{des} - \dot{\mathbf{p}}) + \mathbf{K}_p(\mathbf{p}_{des} - \mathbf{p}) + \mathbf{g} \quad (29)$$

where $\ddot{\mathbf{p}}_{des}$, $\dot{\mathbf{p}}_{des}$, and \mathbf{p}_{des} are the desired values for acceleration, velocity, and position respectively, and $\dot{\mathbf{p}}$ and \mathbf{p} are the velocity and position from the visual

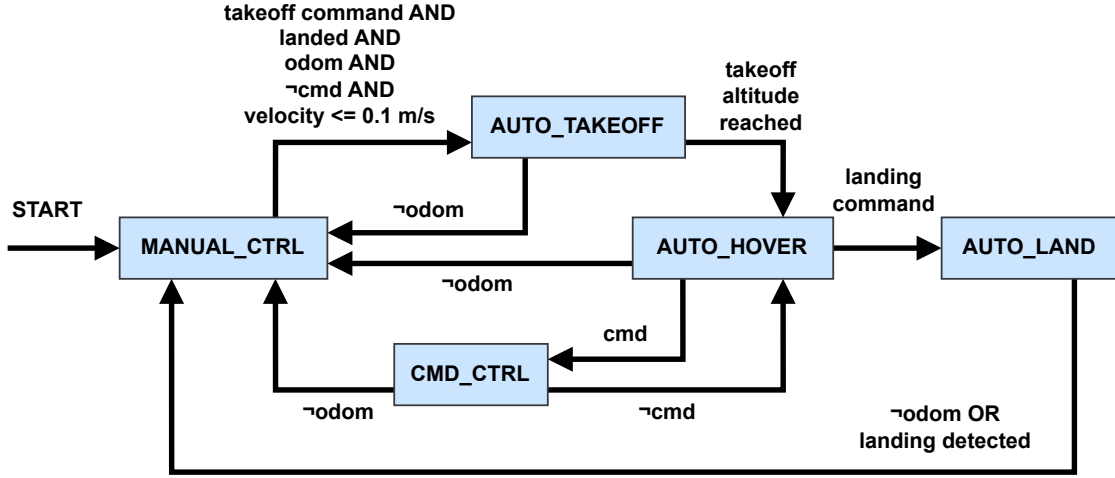


Figure 8: Simplified representation of the state machine of px4ctrl package.

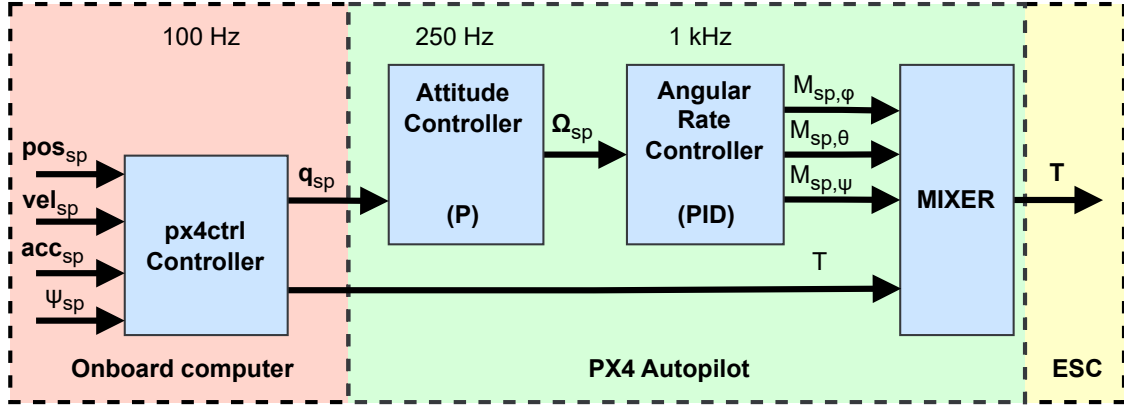


Figure 9: Controller architecture of the system. The path planner outputs desired values for position, velocity, acceleration, and yaw angle, and the trajectory tracking controller running on the onboard computer converts them to the attitude setpoint quaternion and the scalar thrust value. The setpoint quaternion is forwarded to the attitude controller, which outputs angular rate setpoints for the rate controller. The rate controller outputs torque setpoints in roll, pitch, and yaw directions, which are mixed with the thrust value, and the resulting desired thrust values for all four motors are forwarded to the ESC.

odometry respectively. \mathbf{K}_v and \mathbf{K}_p are the proportional gains for velocity and position respectively, and $\mathbf{g} = [0, 0, g]$ is the gravitation.

The setpoints for roll and pitch are calculated from the acceleration setpoint by the following equations:

$$\phi_{sp} = \frac{1}{g}(\ddot{p}_{1,sp} \sin \psi - \ddot{p}_{2,sp} \cos \psi) \quad (30)$$

$$\theta_{sp} = \frac{1}{g}(\ddot{p}_{1,sp} \cos \psi - \ddot{p}_{2,sp} \sin \psi) \quad (31)$$

where ψ is the current yaw-estimate from visual odometry. The setpoint for yaw is given by the path planner. The attitude setpoints are converted to quaternion formation and forwarded to the attitude controller of the autopilot on the user-defined frequency. The default value of the frequency is 100 Hz.

The scalar setpoint coefficient for the thrust is calculated by:

$$T = \frac{\ddot{p}_{3,sp}}{u_t} \quad (32)$$

where u_t is a scaling parameter mapping the thrust to acceleration. The update interval of the thrust scaling parameter is 35 - 45 ms, and it is estimated online by Recursive Least Squares (RLS) algorithm (definition e.g. in [84]). The equations for u_i estimation are

$$\gamma_t = \frac{1}{\rho + c_t \cdot P_{t-1} \cdot c_t} \quad (33)$$

$$K_t = \gamma_t \cdot P_{t-1} \cdot c_t \quad (34)$$

$$u_t = u_{t-1} + K_t \cdot (\ddot{p}_{3,t} - c_t \cdot u_{t-1}) \quad (35)$$

$$P_t = \frac{(1 - K_t \cdot c_t)P_{t-1}}{\rho} \quad (36)$$

where c_i is an old thrust setpoint coefficient from 35 - 45 ms earlier, $\ddot{p}_{3,t}$ is the current acceleration in the z-direction from the IMU, and ρ is set to constant value 0.998. After the startup, u_i is initialized by dividing g by a user-defined hover percentage, which is an approximate thrust value needed to hover still. Initial value for P_t is 10^6 .

The attitude controller of the PX4 autopilot is a P-controller running on 250 Hz [85]. The attitude controller outputs setpoints for the angular rate controller of PX4. The angular rate controller is a PID controller running on 1 kHz. The angular rate controller outputs torque setpoints for all three axes [86]. PX4 angular rate controller consists of three independent PID controllers, and one controller is controlling the rate along one axis.

The torque setpoints from the rate controller and the scalar thrust value from the px4ctrl controller are forwarded to the PX4 mixer. Mixer converts the desired torques into individual motor thrust commands ensuring also the physical feasibility of the commands. The motor thrust commands from the mixer are forwarded to the electronic speed controller (ESC) of the motors.

5 Experiments in simulation

In this chapter, the system was tested and its performance was evaluated in a simulated environment with a simulated drone. Chapter 5.1 contains a short description of the used simulator and the simulated drone. Chapter 5.2 describes the first experiment, where the path planning and obstacle avoidance of the system was evaluated in simulated spruce forests. The experiment aimed to answer the question of how flying velocity and forest density affect the performance of obstacle avoidance. This experiment concentrated on evaluating the planner, so the drone was using a precise global position estimate. Chapter 5.3 concentrates on the evaluation of the performance of VIO. The experiment aimed to answer the question of what is the accuracy of the used VIO algorithm, VINS-Fusion. In addition to the error of the pose estimate, the experiment evaluated the deviation of the VIO estimates by running the estimation with the exactly same data multiple times. In Chapter 5.4 the system in its entirety was tested in a simulated forest that was generated based on the point cloud data from a real boreal forest in Finland. This experiment evaluated the performance of the system in longer flights than in the previous experiments. The experiment was also used for ensuring that running VIO and path planning work together without unexpected problems. The results of the simulations are discussed in Chapter 5.5.

5.1 Simulator

5.1.1 Software

Simulation experiments were performed with a desktop computer running Ubuntu 18.04 LTS and ROS Melodic. The system was simulated with PX4 Software In the Loop (SITL) simulation where the entire flight stack was running on a desktop computer [87]. The Gazebo simulator version 9.0 was used. First published in 2004, Gazebo is an open-source 3D simulator designed for simulating robot systems [88].

Gazebo has distributed architecture [89]. Physics simulation, rendering, user interface, communication, and sensor generation have their own libraries. Gazebo provides also two executable programs: a server for simulating the rendering, physics, and sensors, and a client providing a graphical interface to visualize and interact with the simulation.

The communication library supports publish/subscribe paradigm and it uses Google Protobuf¹² for the message serialization and Boost.Asio¹³ for the transport mechanism. The sensor generation library was used for generating various types of sensors, listening to state updates from the physics simulator, and producing sensor outputs. Users can interact with the simulation by using graphical Qt¹⁴ widgets provided by the GUI (Graphical User Interface) library. The rendering library provides an interface for rendering 3D scenes to both sensor libraries and

¹²<https://github.com/protocolbuffers/protobuf>

¹³<https://github.com/boostorg/asio>

¹⁴<https://code.qt.io/cgit/>

the GUI with OGRE¹⁵. The Physics library provides an interface to fundamental simulation components, including joints, rigid bodies, and collision shapes [90]. The interface supports four open-source physics engines: Open Dynamics Engine (ODE)¹⁶, Bullet¹⁷, Simbody¹⁸, and Dynamic Animation and Robotics Toolkit (DART)¹⁹. The default physics engine is ODE, and hence it was used also in this thesis.

5.1.2 Simulated drone

The test flights were performed with a simulated 3DR Solo drone since it was the smallest quadcopter whose Gazebo model was included in the PX4 software. The diagonal distance between the motors of the model was approximately 41 cm, and the radius of the propeller model was approximately 12.5 cm, so the radius of the whole drone model was approximately $(41 \text{ cm} / 2) + 12.5 \text{ cm} = 33 \text{ cm}$.

A simulated Intel RealSense D435 camera was attached to the drone. For simulating the D435, a Gazebo ROS plugin published by PAL Robotics was used²⁰. The camera was publishing both depth images and grayscale stereo images at a frequency of 30 Hz. The resolution of all images was 640×480 pixels.

The extrinsic parameters between the left and right cameras and the IMU for VINS-Fusion were obtained straightly from the model configuration files of the simulated drone and the camera. The used camera plugin published the intrinsic camera parameters to ROS topics. The intrinsic parameters of the left and right cameras were copied to VINS-Fusion configuration files from corresponding ROS topics. For the mapping code, intrinsic parameters of another Gazebo depth camera model were accidentally forgotten in the configuration file. However, the maximum difference between the used values and the correct values published to the ROS topic was 0.5, so the effect on the mapping was minimal.

The suitable proportional gain values for the tracking controller were adopted from the default gains of `mavros_controllers`²¹ tracking controller.

A picture of a simulated drone configuration is presented in Figure 10.

5.2 Obstacle avoidance in simulated forests

The first experiment in the simulated environment tested the impacts of flying velocity and forest density on the ability of the system to avoid collisions with trees. In this experiment, the system did not use VIO for positioning, but it was given a precise local position estimate by the simulated PX4 flight controller instead. PX4 estimates the local position by using Extended Kalman Filter to fuse sensor measurements from different sensors [91]. To provide a precise position estimate,

¹⁵<https://github.com/OGRECave/ogre>

¹⁶<https://bitbucket.org/odedevs/ode/src/master/>

¹⁷<https://github.com/bulletphysics/bullet3>

¹⁸<https://github.com/simbody>

¹⁹<https://github.com/dartsim/dart>

²⁰https://github.com/pal-robotics/realsense_gazebo_plugin

²¹https://github.com/Jaeyoung-Lim/mavros_controllers



Figure 10: Intel RealSense D435 camera attached to 3DR Solo drone in Gazebo

GPS noise, Magnetometer noise, accelerometer random walk, and gyroscope random walk were all set to a minimum.

5.2.1 Environment and the experimental setup

For the experiment, a total of 12 different forests with three different densities were generated. The tree densities of the forests were 0.1 trees/m², 0.15 trees/m², and 0.2 trees/m², and in each case, four replicates were generated. The simulated forests did not have understory vegetation. As a reference for determining meaningful order of magnitude for densities, an article by Liang et al. [92] was used. The focus of that study was the laser scanning of forests by drones, and there the test areas in Finnish boreal forests were classified into three categories. In "easy" forests, the tree density was 0.07 trees/m² with minimal understory vegetation, in "medium" forests the density was 0.1 trees/m² with moderate understory vegetation, and in "hard" forests the density was 0.2 trees/m² with dense understory vegetation.

The generation process was adapted from the forest generation script used in Oleynikova et al. [93]. The original script for generating randomized forests is openly available for research purposes²². The script uniformly draws the position, orientation, and scale of the trees, and writes the world description file for Gazebo. The tree model is drawn from given meshes. In this thesis, Norway spruce models provided by Globe Plants were used²³. The package included six different spruce models in .blend format, that were converted into .dae format with Blender [94].

Due to the high computational burden of the highly textured 3D models, the tops of the tree models were cut out. For the same reason, the test forests needed to be small. Thus the width of the test forests was set to 10 m and the length was set to 20 m. In addition to the trees, walls were placed at the side edges of the forests so that the drone could not plan routes outside the forest area. The ground was flat in all forests. The texture of the grounds was adapted from an aerial photograph of a clearcut forest area. Figure 11 presents an example of a generated forest.

²²https://github.com/ethz-asl/forest_gen

²³<https://globeplants.com/products/picea-abies-norway-spruce-3d-model>

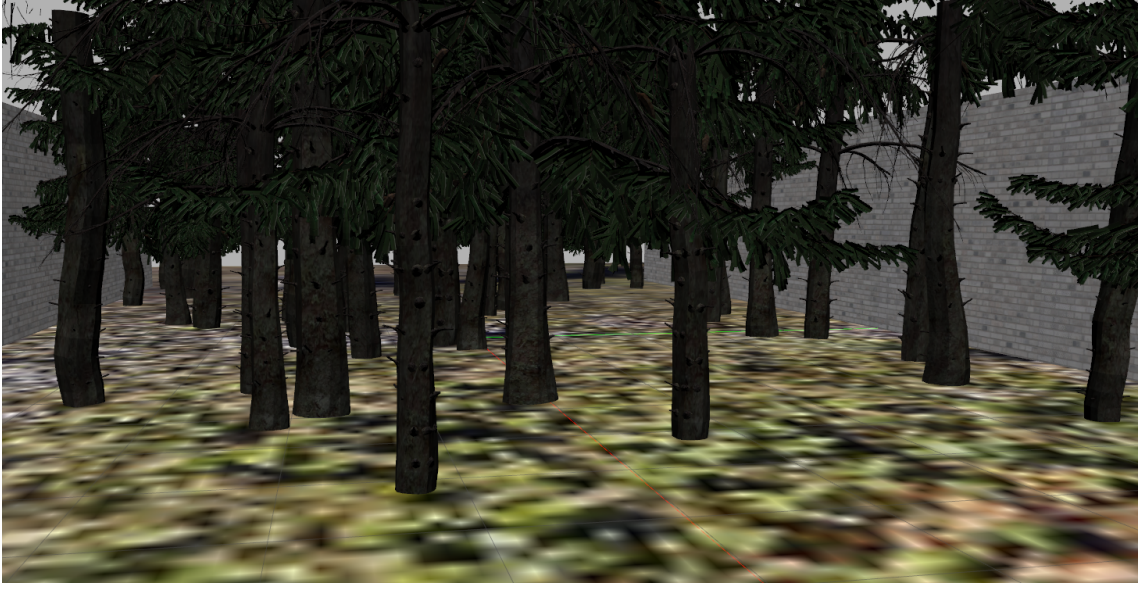


Figure 11: Example of generated simulated forest with a tree density of 0.2 trees/m².

The flight distance of the test flights was 30 m. The start point was 5 m before the forest, and the goal point was 5 m behind the forest. The takeoff altitude and the height of the goal point were 1 m. For every forest, the system was tested with four different maximum flight velocities, and five flights were performed with each forest and velocity combination. The used maximum flight velocities were 1.0 m/s, 1.5 m/s, 2.0 m/s, and 2.5 m/s.

For every forest and velocity combination, the success rate of the flights was calculated. The flight was considered successful if the system managed to navigate to the goal. For successful flights, also the need for emergency stops during the flight was measured.

5.2.2 Results

A summary of the success rates of the flights is presented in Table 2. The amounts of successful flights without any emergency stops with every tree density and maximum velocity combination are presented in Table 3. Figure 12 presents the paths with a maximum flying speed of 2.0 m/s and forest density of 0.2 trees/m² as an example. With that maximum flight velocity and tree density there were 17 successful flights and three failed flights.

The results showed that obstacle avoidance performed very well when the forest was sparse or the maximum flying speed was slow. With the slow speed in a sparse forest, all flights were successful. However, fast flying in a dense forest caused problems for obstacle avoidance, and with the highest velocity in the densest forest, only 30 % of the flights were successful.

In the sparsest forest, system successfully navigated to the goal in all 20 flights with maximum velocities of 1.0 m/s and 2.5 m/s. However, with maximum velocities of 1.5 m/s and 2.0 m/s, the system failed in one flight at each velocity. The system

Table 2: Flight success rates depending on the maximum flight velocity and the tree density.

$\begin{array}{c} \text{m/s} \\ \text{trees/m}^2 \end{array}$	1.0	1.5	2.0	2.5
0.1	20/20	19/20	19/20	20/20
0.15	20/20	18/20	17/20	17/20
0.2	18/20	13/20	12/20	6/20

Table 3: Amount of successful flights without emergency stops depending on the maximum flight velocity and the tree density.

$\begin{array}{c} \text{m/s} \\ \text{trees/m}^2 \end{array}$	1.0	1.5	2.0	2.5
0.1	17/20	16/20	12/20	11/20
0.15	15/20	12/20	9/20	6/20
0.2	8/20	7/20	2/20	1/20

did not collide with the trees, but in both cases, the path planner died after an emergency stop near the ground, after the planner gave multiple warnings saying "The drone is in obstacle" and "Unable to handle the initial or end point". After the planner died, the system started hovering, since new waypoints were not received. Both failed flights occurred in the same random forest. That forest was difficult for the system also in other flights since the forest contained low branches which forced the system to fly near the ground. In that forest, with the highest velocity, the landing gears of the drone did slightly touch the ground in two of the five flights, but the system managed to continue flying in both cases.

With a tree density of 0.15 trees/m², the system managed to navigate to the goal in all flights when the maximum velocity was 1 m/s. With a maximum velocity of 1.5 m/s, two flights failed. The reason was dying of the planner after an emergency stop near the ground. With a maximum velocity of 2.0 m/s, three flights failed. In two flights the reason for failure was again dying of the planner after an emergency stop near the ground, but in one flight the drone collided with a tree. With the largest maximum velocity, three of the flights failed. The reason for the failure was a collision with a tree in all cases.

With the tree density of 0.2 trees/m², the system failed in two flights when the maximum velocity was 1 m/s. The reason for the first failure was a collision with the ground. In the second failed flight the system got stuck so that it flew for 1.5 minutes without finding the path. The last error, where the system stopped the path

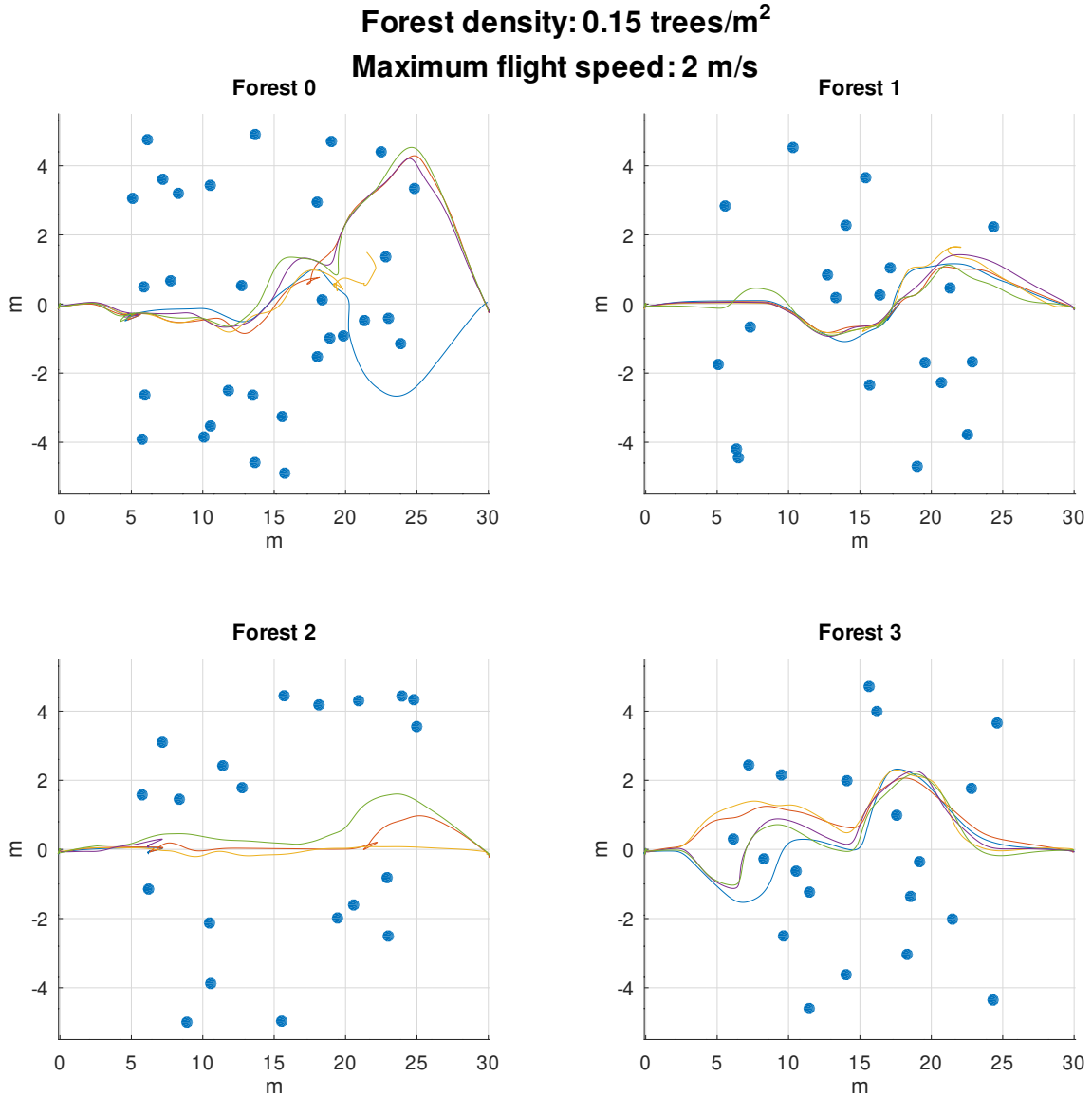


Figure 12: Paths of the flights with a maximum flight speed of 2.0 m/s in forests with a tree density of 0.15 trees/m². There was one failed flight in forest 0, and two failed flights in forest 2. The blue dots mark the locations of the tree trunks. Branches are not visible in the figures.

planning and started hovering, was an error saying "odom or depth lost". That error is raised, if the time from the last received depth image or odometry data exceeds one second. After that error is raised, the planner goes to "emergency" -mode so that it does not recover before rebooting. However, there were no significant drops neither in depth image frequency nor odometry frequency. With a maximum velocity of 1.5 m/s, seven flights failed. The reason for failure was a collision with the ground in two flights, and a collision with a tree in two flights. On two flights the planner died after an emergency stop near the ground. In one flight the navigation was manually aborted after the drone had been stuck and flying without progress for almost 1.5

minutes. With the maximum velocity of 2.0 m/s, eight flights failed. The reason for the failure was colliding with a tree in five cases, and colliding with the ground in one flight. Two times the reason was dying of the planner. With a maximum flying velocity of 2.5 m/s, only six flights were successful. The reason for the failure was colliding with a tree in flights, and colliding with the ground in three flights. In addition, in one flight the propellers had a hard collision with the ground, but the simulated drone still managed to continue flying. In that case, the navigation was manually aborted, since continuing after that collision would have been unrealistic. For two flights, the reason for the failure was again the "odom or depth lost" -error. The rosbag recordings in both of these flights had problems with exceeding the record buffer, so the flight data, and therefore also the real update rate of the sensors, cannot be perfectly restored from the bags. However, from the bags, it can be seen that there were multiple depth image and odometry data updates during the second when the system did not receive them.

In addition to the avoidance of collisions, another important aspect of the path quality is the smoothness of the trajectory. For that reason, also the need for emergency stops during successful flights was examined with every density and velocity combination.

With the lowest tree density and lowest maximum flight speed, only in three of the 20 flights, an emergency stop was needed. In the same forests but with a maximum velocity of 2.5 m/s, in nine flights an emergency stop was needed. With a tree density of 0.2 trees/m², there were only eight smooth and successful flights without an emergency stop. with that density and maximum velocity of 2.5 m/s, only one flight was performed successfully without emergency stops.

Based on the results, increasing either the flying speed or the forest density decreases the flight success rate and increases the need for emergency stops. With a low flying speed in a sparse forest, the flight success is reliable and trajectories are usually smooth. With the lowest flying speed, there were only two failed flights even in the densest forest, but the trajectories were less smooth since only 8 flights were performed without emergency stops. With the highest forest density and highest flight speed 70 % of flights failed. The results showed that the performance of obstacle avoidance was sufficient even in dense forests. However, the maximum flying speed should not be much higher than 1.0 m/s unless the environment is very sparse.

5.3 VIO precision in simulated forests

The second experiment evaluated the accuracy and the precision of the proposed VIO algorithm with one forward-looking simulated RealSense D435. The simulated drone was still using the precise position estimate provided by the flight controller for flying, but the stereo images and IMU values were recorded to rosbag. VINS-Fusion algorithm was run for the recorded data, and the results were compared to the ground truth acquired with Gazebo ground truth plugin for ROS systems [95].

5.3.1 Environment and the experimental setup

All flights were performed in the same forest. The forest was the `forest0` with a density of 0.1 trees/m² used in the first experiment. In total, six flights were recorded. From those, the first three flights were performed with a maximum velocity of 1.0 m/s and the last three flights with 2.0 m/s. With the maximum velocity of 1.0 m/s, there were no emergency stops, but in the flights performed with the maximum velocity of 2.0 m/s, there was one emergency stop in each flight. The flight distance was 25 m, and the altitudes of the start and goal points were 1 m. Since the test flights were just straight flights from the start point to the goal, there were no loop closures.

Generally, since the VIO is not a deterministic process, the tracking result it produces varies with every run even if the input data remain the same. Due to that reason, in this test, the VINS-Fusion was executed 20 times in total for every recorded flight. In the simulation, there was no synchronization between IMU and camera, so the time difference t_d between the sensors needed to be estimated. For every flight, the VINS-Fusion was executed 10 times using its online estimation of t_d , and 10 times with t_d fixed to a constant value. The constant value used for time difference was -4.25 ms, which was acquired by roughly testing different values. In the configuration of VINS-Fusion, the default values of IMU noise parameters were used.

VINS-Fusion measures the translation of the camera with regard to the initial position, and the ground truth plugin gives the position of the base link of the drone in the world coordinates. The accuracy of the positioning was evaluated by comparing the VINS position estimate at the end of the rosbag to the difference between the last and the first ground truth position values. Since the start position was flat, the connection between the camera and drone was rigid, and their coordinate systems were aligned, the orientation error was estimated by measuring the difference between the ground truth and VINS orientations at the end of the rosbag.

5.3.2 Results

Figure 13 presents the average position errors of VINS-estimations per flight. From the results it can be seen, that in five out of the six flights, the average position error was smaller with fixed t_d than with using online t_d estimation. The average position error over all flights was 0.4893 m with fixed t_d and 0.8496 m with using online t_d estimation.

The average position errors per xyz-directions with online estimated t_d are presented in Figure 14a and with fixed t_d in Figure 14b. It can be seen, that in both cases the error is highest in the x-dimension. That is logical, since most of the moving happened in that direction. There is no significant difference in errors in y- and z-coordinates between fixed t_d runs and online estimated t_d runs.

In addition to the accuracy, another important factor to consider in the evaluation of the localization is the precision of the estimate. The standard deviation of the position estimation results with online estimated t_d are presented in Figure 15a and with fixed t_d in Figure 15b. From the results, it can be seen that also the deviation

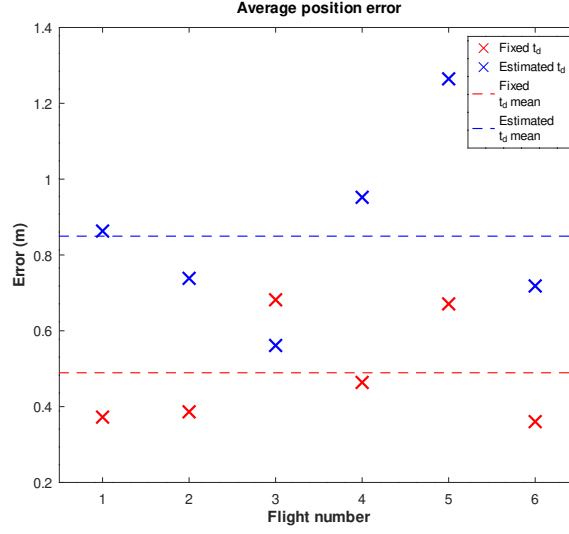
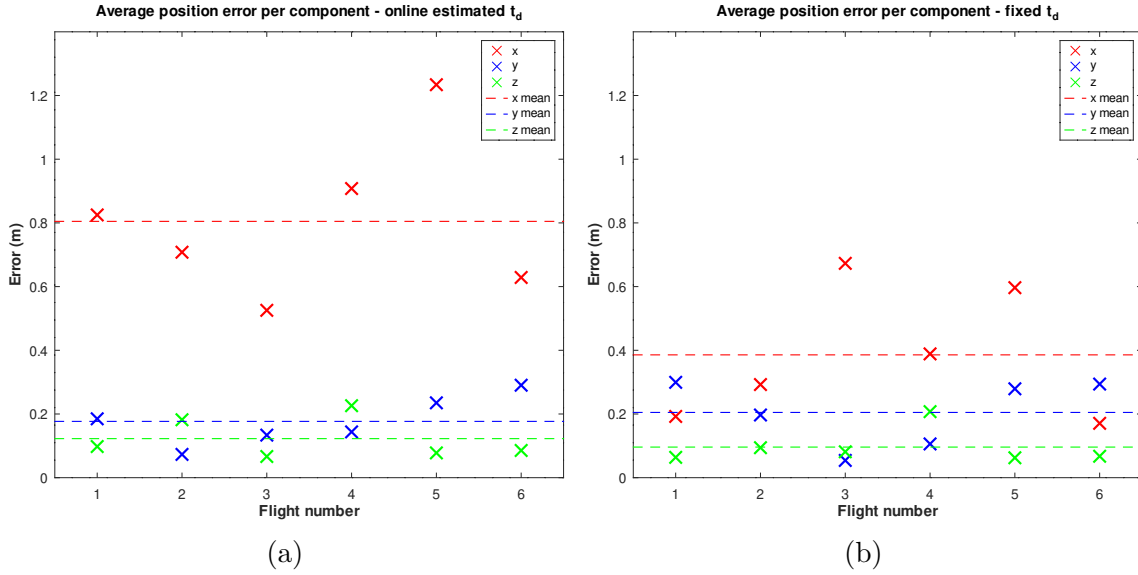


Figure 13: Average position error

Figure 14: (a) Average position error per component with online estimated t_d (b) Average position error per component with fixed t_d

in the z-direction was smaller than in the x- and y-directions. The average standard deviation in the y-direction was a little bit higher with online estimated t_d than with fixed t_d . There was no significant difference in average standard deviations in the x-direction between online estimated t_d and fixed t_d . However, the deviation seems to vary more between the flights with fixed t_d than with online estimated t_d . With online estimated t_d the standard deviation was very high (≈ 0.36 m) in the estimates of the first flight, but for all other flights, the standard deviation was between ≈ 0.14 m and ≈ 0.19 m. With fixed t_d the standard deviation varied between ≈ 0.27 m and ≈ 0.05 m uniformly.

The average errors of VINS-Fusion orientation estimates with online estimated t_d

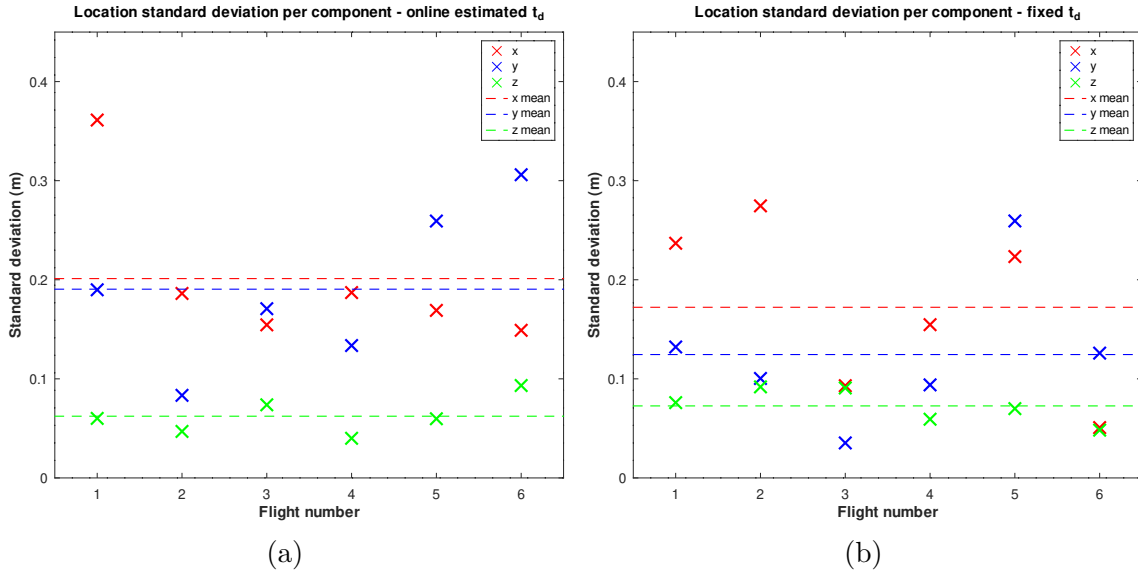


Figure 15: (a) The standard deviation of the position estimate per component with online estimated t_d (b) The standard deviation of the position estimate per component with fixed t_d .

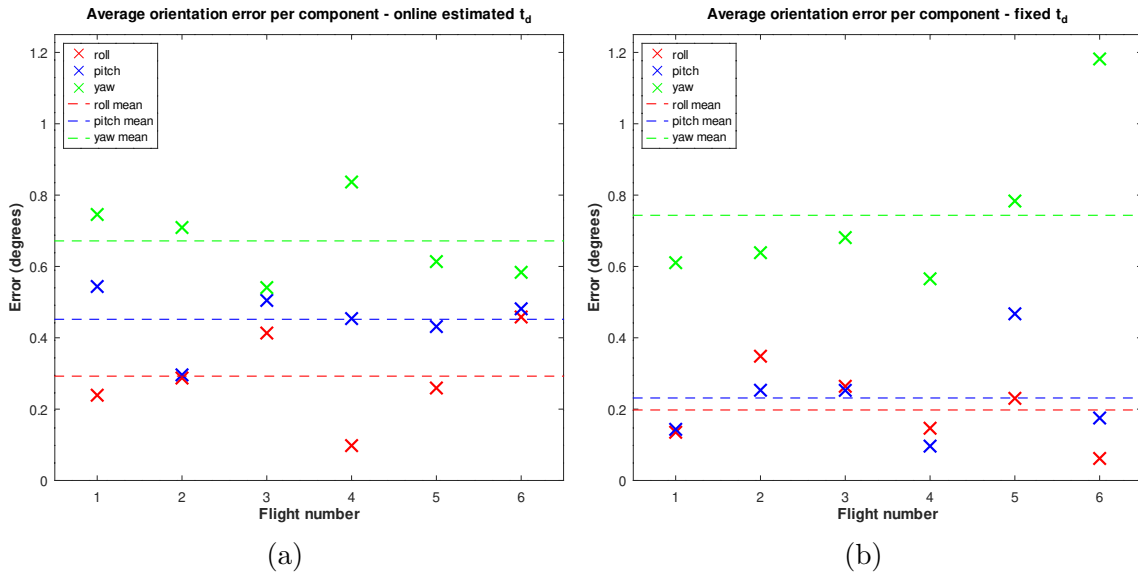


Figure 16: (a) Average orientation error per component with online estimated t_d (b) Average orientation error per component with fixed t_d .

are presented in Figure 16a and with fixed t_d in Figure 16b. From the results, it can be seen that in both cases the average orientation error in yaw-direction was higher than the error in roll- and pitch-directions in every flight. However, the average orientation errors even in the yaw-direction were relatively small. The average yaw error across all six flights was 0.6716° with online estimated t_d and 0.7434° with fixed t_d . Only in one flight, the average yaw estimate error was over one degree.

The results showed that the position estimate was more precise with fixed t_d than

with using an online estimation of t_d . With fixed t_d , the accuracy and precision were sufficient to use for pose estimation of the drone. Even in the worst-case scenario, the error of the pose estimate was small enough for short flights through forests. However, in the longer operations, that are beyond the scope of this thesis, the errors accumulate quickly, and especially in the worst-case scenario the pose estimate would become unusable without corrections to it.

5.4 Overall flight performance

In the third experiment, the system was tested in its entirety, so that the drone was avoiding trees and the pose estimate was given by VINS-Fusion. In this experiment, a computationally lighter environment based on the data from a real forest was used making it possible to test the performance in longer flights than in the previous experiment. The evaluation criteria for the performance of the system were the ability to avoid obstacles and the accuracy of the VIO. The experiment also validated the joint operation of obstacle avoidance modules and VINS-Fusion.

5.4.1 Environment and the experimental setup

The simulated forest was generated using data from a real boreal forest in Finland. The forest was relatively sparse and easy pine forest with a minimal understory canopy. The data was collected with DJI P1 Zenmuse camera [96] attached to DJI M300 RTK Drone. The flying altitude was 120 m. The data was processed and the 3D model was created with Agisoft Metashape [97].

Unlike in the other simulated models in this thesis, the ground in this experiment was uneven. The forest was on a gentle slope, and tussocks, rocks, and tree stumps were included in the model. The screenshot from the simulated forest model is presented in Figure 17.

In total, ten test flights were performed. The goal was set to 90 m forward from the start position. The flying direction was up the slope. The altitude of the ground increased by approximately 6 m during the flight path. For that reason, the goal was also set to 8 m higher than the starting point, so that the goal point's altitude from the ground was approximately 2 m. Maximum flying velocity was 2.0 m/s in all flights, and the takeoff altitude was 1.0 m.

5.4.2 Results

The system made all ten flights successfully without collisions. The drone managed to navigate to the approximately right place on every flight, and there was only one emergency stop during all ten flights.

The average position error of the ten flights was 1.3189 m. The highest position error was 2.6754 m, and the lowest position error was 0.6505 m. The position errors of all ten flights are presented in Figure 18.

The position error was largest in the y-direction. The average and largest position errors in the y-direction were 0.9756 m and 2.5040 m, respectively. The error was

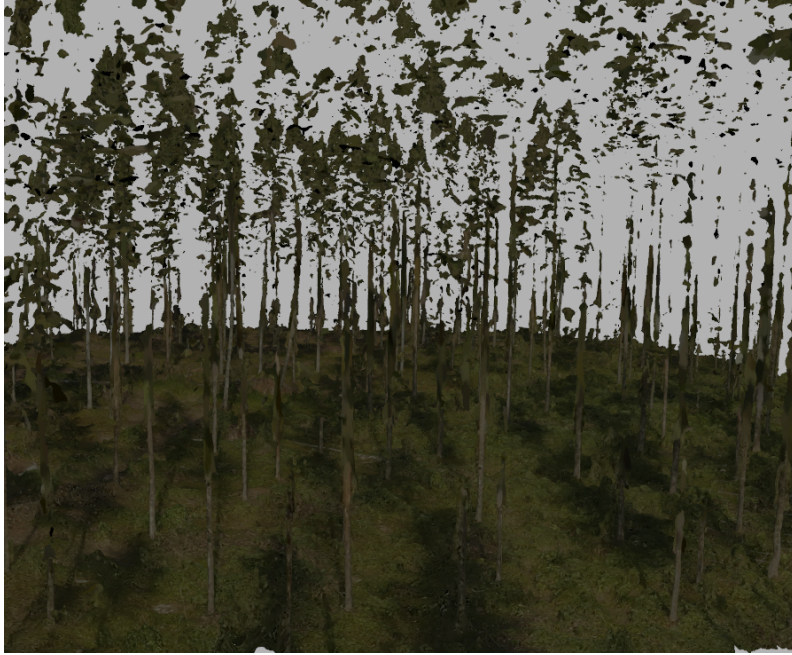


Figure 17: Simulated forest generated from real point cloud data imported into Gazebo.

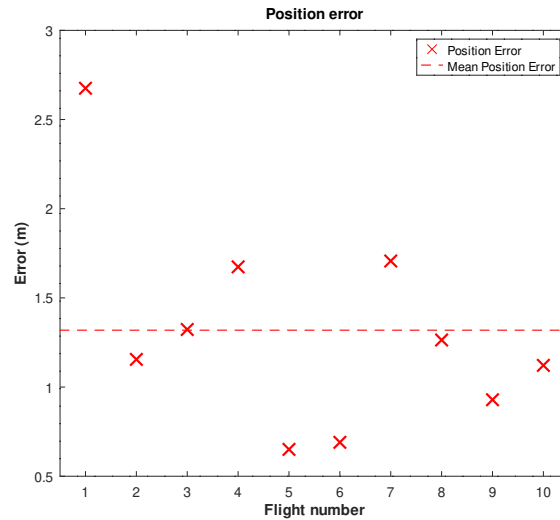


Figure 18: Position errors in flights.

lowest in the z-direction, where the average error was 0.4399 m. Position errors per component of all ten flights are presented in Figure 19a.

The orientation error was again highest in the yaw-direction, the average and largest error were of 0.8452° and 1.8644° , respectively. Average errors in roll and pitch estimates were much lower, 0.2411° and 0.2595° , respectively. Orientation errors per component of all ten flights are presented in Figure 19b.

The results showed that VINS-Fusion and obstacle avoidance can be combined without problems. The system was able to navigate and avoid obstacles inside a

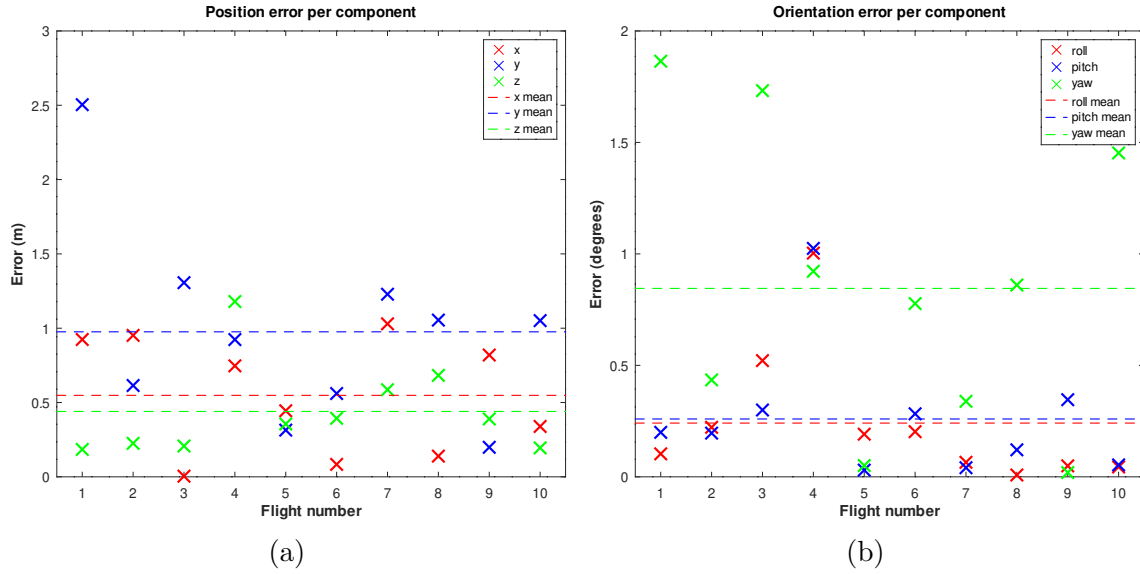


Figure 19: (a) Position error per component (b) Orientation error per component.

forest by relying only on VIO in its positioning. Inside a sparse forest environment, the performance of the obstacle avoidance was perfect and all trees were avoided reliably. The average position accuracy was still sufficient for navigating through forests, even though the flight distances were longer than in the previous chapter. However, in the worst case, the position error was over 2.5 m, which could already be problematic, if the chosen open area for the goal is not very big.

5.5 Discussion

The experiments in the simulator first tested independently the performance of the two main components of the system, namely the obstacle avoidance and the VIO. The experiments were performed with the Gazebo physics simulator, and the environments were built from artificial spruce models. After these experiments, the system was tested in its entirety in a sparse forest model generated using data from a real pine forest in Finland.

The obstacle avoidance was tested by varying the maximum flight speed and the density of the forest. The maximum flight speeds varied between 1.0 m/s and 2.5 m/s, and the densities of the forests varied between 0.1 trees/m² and 0.2 trees/m². The results showed that the flight success rate decreased when the tree density or flying speed increased. An increase in the maximum flying speed or the tree density increased also the number of emergency stops during successful flights. The results are logical since the complexity of the navigation is dependent on the density of the forest, and with higher flying speed the system has less time to react to newly sensed objects.

In the original article by Zhou et al. [63] the system was tested only in swarm operations. The simulator used in the simulated tests of the original article was also highly simplified concentrating only on path planning, and it did not simulate the

physics or sensors. For these reasons, comparing the obtained results to the results of the original paper is difficult.

In the simulated tests of the original article, the radius of the drones in the swarms was 0.2 m. The obstacle avoidance was tested by swarm flying with maximum flying speeds varying from 1 m/s to 3 m/s and average obstacle spacings varying from 2 m to 1 m. There were no collisions during the tests. There exists multiple reasons that explain why the system performed better in the simulations of the original paper. First, the simplified simulator model of the drone model was smaller. Secondly, in the simulations, the obstacles were simple shapes without small branches, and there was no ground. Thirdly, since the sensors were not modeled in the original paper, the drone had perfect and very wide local maps making it possible to detect all small gaps in paths. Fourthly, since the simulator did not model the physics, there was no possibility for failures related to the physical limits of the drone or tracking the trajectory.

In addition to the aforementioned reasons, it can not be guaranteed that the computational limits of the computer did not limit the performance of the system during the simulated flights. Every world with the highest density was tested once so that the CPU usage was monitored and it was checked that the utilization rate did not reach 100% in any cores during these flights. However, the utilization rate was very high. It is possible that it may have reached 100% on some other flights, or that the computer has slowed the processes due to the high utilization rate even without reaching the 100% rate.

However, based on the results obtained in chapter 5.2, in the simulator, the performance of the obstacle avoidance is good enough for data collection in forest research. With a maximum speed of 1.0 m/s, the drone managed to avoid collisions in almost every flight even in the densest forest. In the middle-density forest, the system managed to avoid collisions in every flight. In the sparsest and the middle-density forest, most of the flights were also smooth without the emergency stops.

The original article did not evaluate the VIO performance. In the literature, the VIO algorithms are rarely evaluated in simulated data. In the original article defining VINS-Fusion for local sensors [66], the performance was evaluated with a dataset collected by a drone with a forward-looking stereo camera and an IMU. However, the error metrics used in that article were different from the ones used in chapter 5.3 of this thesis. The original article used Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) method proposed by Geiger et al [98] instead of just comparing the pose of the VINS-Fusion and ground truth at the end of the trajectory. In the original article, the evaluation data did contain also loops. It was not mentioned in the article if the loop closure attribute of the VINS-Fusion was used or not. For these reasons, comparing the results of VINS-localization obtained in the chapters 5.3 and 5.4 to the results of the original VINS-Fusion article is difficult.

Based on the results obtained in chapters 5.3 and 5.4, VINS-Fusion can be used for localization inside a forest, but in the longer flights a drift can be notable. The results showed that VINS-Fusion works better with fixed t_d than with using its built-in online estimation method for t_d . With a fixed t_d value, the average position error was 0.4893 m in the 25 m flights in chapter 5.3 and 1.3189 m in the 90 m

flights in chapter 5.4. Therefore the total positioning error seems to grow logically when the flight distance increases, but the largest component of the error differed between the experiments. In the 25 m flights, the position error was highest in the x-direction, but in the 90 m flights, the error was highest in the y-direction. The reason for the difference remained unknown in the study. The average orientation errors were under 1° with respect to all axes in both experiments.

The deviation of the position error between flights was relatively high. In the 90 m flights performed in chapter 5.4 the highest position error was 2.6754 m and the lowest position error was 0.6505 m. One reason for the high deviation in the positioning results is the used error metric since only the pose error of the last position was measured. Errors at the beginning of the flight cumulate to high errors on long flights. Similar error in the tracking affects the pose error differently depending on the time when it happens. Errors at the beginning of the flight cumulate to high errors in the long flights, but a similar error right before the end of the flight leads to barely notable errors in the end pose.

The experiments evaluating the performance of the system in the simulation contained many limitations. In the evaluation of VINS-Fusion, it would have been better to use some other error metric than a simple pose error at the end of the trajectory. By using for example ATE, the results could have been compared to the results presented in the original VINS-Fusion paper. Also using a fixed t_d value without precise calibration is based on guessing and testing to find a good enough value, which is not the exact time difference between the camera and the IMU. In addition, it cannot be guaranteed that the time difference is even a static value when there is no synchronization between the sensors. It is possible, that in a simulator the actual t_d value changes depending on the state of other processes running on the computer. In the experiments presented in chapters 5.2 and 5.3, the ground was flat and there was no understory vegetation, so despite having realistic tree models, the forest model was unrealistic.

As in the obstacle avoidance experiment, the computational heaviness was a problem with the VINS-Fusion experiments also. From the point of view of evaluation, it would have been better to have longer flights, but the computational heaviness of the simulation limited the forest sizes. For this thesis, Gazebo was chosen as the simulator since it seemed to be the most widely used option with PX4. Thanks to the wide support and documentation for PX4, it was a big help during the initial testing and the commissioning of the system. However, the computational heaviness limited its usability in the evaluation of the system. To have computationally lighter simulations, other simulator options need to be tested in the future.

6 Experiments with real hardware

In this section, the system was tested and its performance was evaluated with a real drone. Chapter 6.1 contains the description of the drone hardware that was used in these test flights. The experiments tested the ability of the system to detect and avoid trees, branches, and understory vegetation during navigating in various Nordic forest environments. For some of the flights, also the precision of the VIO was evaluated. Specifically, the tests aimed to answer for following research questions. What is the reliability of the navigation in various environments? What is the accuracy of the VIO estimate with real hardware? What kind of situations cause emergency stops or collisions?

The flight tests were performed in different weather conditions, and the maximum flying speed and the distance to the goal were adjusted during the tests. The performance of the system was tested in three different forests in Southern Finland. Chapter 6.2 presents the flight tests and results in sparse mixed woodland. In Chapter 6.3 flight tests were performed in park woodland with dense understory vegetation. In Chapter 6.4 the performance of the system was tested in dense and snowy spruce forest. The research questions were otherwise the same in all environments, but in the park woodland, VIO accuracy was not measured. The results of the test flights are discussed in Chapter 6.5.

6.1 Description of the platform

6.1.1 Drone hardware

The test flights were performed with custom-built drone hardware. Due to the simplicity and fastness of the building, and the relatively strong protection against collisions during the testing phase, the drone was bigger and heavier than the small drones in the original paper by Zhou et al. [63].

The used onboard computer was ASUS PN51-E1 [99] with AMD Ryzen 7 5700U processor. The used PX4-compliant autopilot was Hex Cube black [100], which contained also the IMU. For mapping and VIO, Intel RealSense D435 [46] camera was used since it has the same depth module D430 [70] that was used in the original article, but it has also shielding protecting against collisions. RealSense D435 is based on active stereoscopy, which allows it to output high-quality depth images. However, laser emitter dots cause problems for VIO since they are moving with the camera, and therefore, instead of the official ROS-RealSense driver, modified driver²⁴ used also in the first EGO-Planner article [69] was utilized. The modified driver turns the laser emitter of the camera constantly on and off so that the camera can produce high-quality depth images in every other frame, and stereo images without the laser dots in every other frame. The system received both stereo images and depth images at a frequency of 30 Hz. The resolution of both images was 640×480 pixels.

²⁴<https://github.com/ZJU-FAST-Lab/ego-planner>

The computer, the camera, and the autopilot were attached to a 330 mm drone frame. The propellers had three pieces of 6.5 cm long blades each. The measured weight of the drone was 1248 g without the batteries, and 1613 g with batteries for the motors and the onboard computer included. The drone is shown in Figure 20. The camera and the autopilot containing the IMU were attached to the drone so that there was damping against the drone vibrations. The computer, the camera, and the autopilot were also covered with smooth foam so that they would be less likely to break in the case of collisions during the test flights.

6.1.2 Calibration

The intrinsic and extrinsic parameters of the left and right grayscale cameras and the time shift between the camera and the autopilot IMU were calculated using an open-source camera calibration toolbox, kalibr²⁵. IMU noise values needed by kalibr were estimated using imu_utils²⁶, which is an open-source ROS package for analyzing the IMU performance.

The intrinsic parameters of the stereo camera were determined by kalibr’s Multiple camera calibration -tool [101]. The result of that tool was used as an input for the Camera-IMU calibration tool [102][103]. In both calibrations, stereo images were collected approximately at a 30 Hz rate and IMU measurements approximately at a 200 Hz rate. Aprilgrid was used as a calibration target [104]. The calibration target was kept stationary and the drone was moved so that translation in every dimension and rotation with respect to every angle got covered. Figure 21 presents the aprilgrid target during the calibration.

For the configuration of the mapping, the intrinsic parameters of the depth image were also needed. For that, the values published by the realsense ROS driver were used. The autopilot IMU was calibrated with QGroundControl software [105].

During the calibration, a couple of mistakes were made. The mistakes were only noticed after the test flights had already been completed, so they might have affected the accuracy of the VIO in the tests, and hence they are listed here.

First, although the imu_utils package was listed in kalibr wiki [106] as one option for obtaining the IMU parameters, the parameters that it produces do not have the same units as the values expected by kalibr. The values provided by imu_utils were drastically too small. Due to that reason, in the initial tests with the system, the VINS-Fusion estimate was more stable with default IMU parameters than with the ones obtained with imu_utils. For that reason, in the VINS configuration, the default IMU values were used during the actual flights described later in this chapter.

Secondly, in the Multiple Camera Calibration non-optimal distortion model was used. The recommended distortion model for RealSense D345 would have been radial-tangential, but an equidistant distortion model was used in the calibration. In the configuration of VINS-Fusion, the distortion coefficients were left to all zero, according to the statement by the support of RealSense SDK²⁷, but the wrong

²⁵<https://github.com/ethz-asl/kalibr>

²⁶https://github.com/gaowenliang/imu_utils

²⁷<https://github.com/IntelRealSense/librealsense/issues/1430>

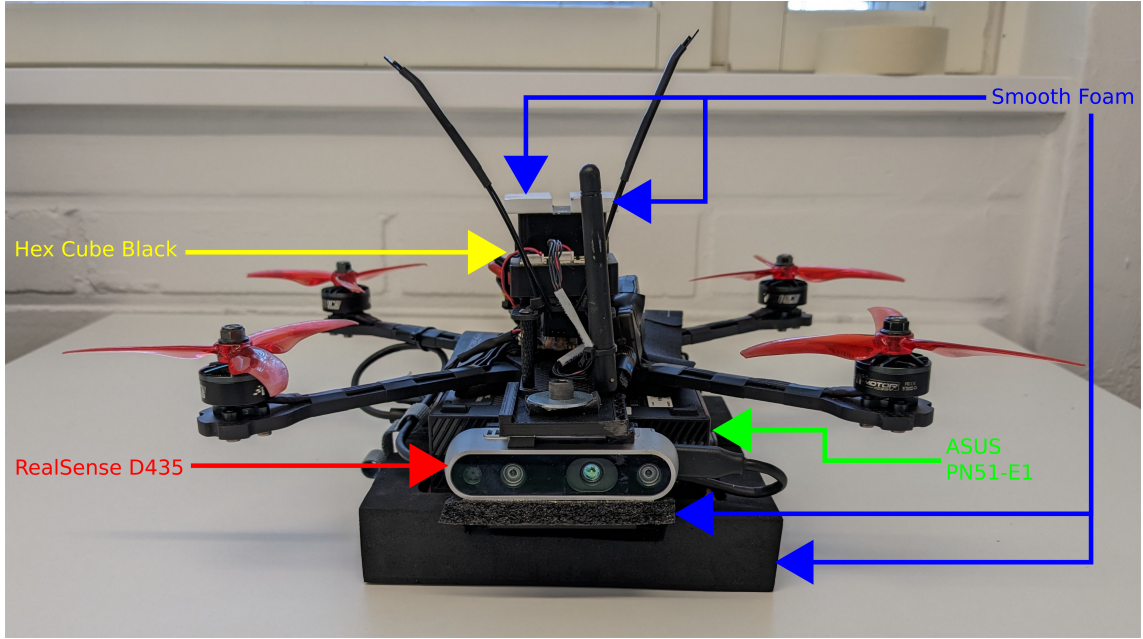


Figure 20: The custom drone used in the flight tests.

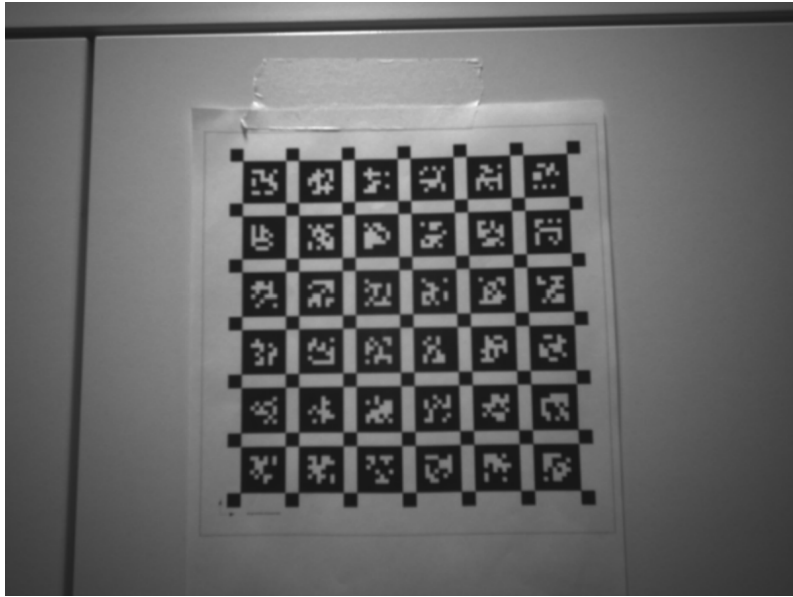


Figure 21: Aprilgrid captured during the calibration.

distortion model could have slightly affected to kalibr's estimation of the intrinsic and extrinsic parameters.

Thirdly, kalibr estimated the time shifts between the IMU and the cameras to be 0.00517074152752 and 0.00515974767192. From these, the average, 0.0051652445997, was taken and given as t_d to VINS-Fusion. However, positive t_d caused problems for VINS-Fusion, and hence the t_d was multiplied with -1 because it was assumed that kalibr gives only the absolute value of the time shift. The value turned out to be a

relatively good value for the t_d , but later it turned out that kalibr gives the t_d values with the sign. The value obtained by calibration was probably corrupted due to too small IMU noise parameters and the wrong distortion model used in the calibration.

6.1.3 Commissioning

Before the flight tests, also the controllers presented in Figure 9 were tuned. The Attitude controller and Angular rate controller were tuned with QGroundController by manually flying the drone and adjusting the values. The tuning was started from the Angular rate controller. The P-value was increased until the system started to have high-frequency oscillations. The D-value was increased so that there were no notable overshoots. The I-value of the controller was set so that there were no notable steady-state errors in the angular rates. After that, the P-value of the Attitude controller was increased until the drone started to oscillate.

The gains for the position and velocity controller in the px4ctrl node were also adjusted during the flight tests. At first, the values used in the simulated tests were adopted. During the first outdoor flight tests, the gains were increased to have more responsiveness to the position and velocity commands given by the planner.

During the initial flight tests performed during the commissioning, one critical problem was encountered. During a flight, the drone did not find a path to the goal and entered into the hovering mode to continue replanning. After a few seconds of hovering, due to an unknown reason, the start point of the path planner jumped from the hovering position to approximately four meters forward. From that point, the planner found a path to the goal and forwarded again a desired state to the controller. However, since the starting point of the planning was 4 m forward instead of the current position, the desired position was also four meters forward from the real position of the system. Four-meter difference between the desired and current position was too big for the position controller, and it resulted in a high-speed dive toward the ground. To prevent hardware breaks, a new condition was added to the state machine presented in Figure 9. The new condition prevents moving from the AUTO_HOVER state to the CMD_CTRL state if the difference between the desired position and the current position is over 25 cm. In such a case, a landing is performed automatically. However, the problem leading to the uncontrolled dive did not appear again during the outdoor flight tests.

6.2 Flying in a sparse mixed forest

6.2.1 Environment and the experimental setup

The first outdoor flying tests were performed in a sparse mixed forest. The test area was a small forest in Otaniemi, Espoo (60°10'50.1"N 24°49'36.4"E). The purpose of these tests was to validate the system's performance in outdoor conditions, so the flight distances were short, and the forest environment was relatively easy. The sun was shining so that there were shadows and changes in the light intensity, but the direction of the sunlight was not against the camera. The wind was very calm. The

wind speeds during the flight tests are presented in Appendix A. The general look of the test area is presented in Figure 22.

In total, nine test flights were performed in the first environment. In the first test flight, the maximum velocity was set to 0.5 m/s. After the first test flight, the maximum flight speed was set to 1.0 m/s for other test flights. In the first four test flights, the goal was set to 13 m forward from the start point, and in the last five test flights, the goal was set to 18 m forward from the start point. The altitude of the goal point was set to 1 m, which was also the takeoff altitude from where the planning started. The location of the start point was changed after six flights so that in the last three flights, the forest between the start and the goal was denser than in the first six tests. For the last three test flights, also approximate distances between the landing locations were measured. The distance from the takeoff location to the landing area was also measured with a laser rangefinder.

6.2.2 Results

The drone managed to successfully navigate to the goal in all of the 13 m flights. However, in the first flight, the drone did not avoid a small needleless branch but instead flew straightly towards it. The branch was so smooth and thin, that the collision did not cause problems for the system, and the drone managed to continue to its goal. However, the branch was not detected on the drone map at all, unlike all other obstacles during the flight. Figure 23a presents the onboard camera's left stereo image and Figure 23b presents the depth image just under a second before the collision to the small branch. The branch can hardly be seen in the left stereo image, but in the depth image, it cannot be seen.

The system did not detect the branch even after the collision. During the collision, the branch wiped right in front of the camera quickly. Figure 24a presents the left stereo image and Figure 24b presents the depth image during the collision to the branch at the moment when the branch was visible in both cameras. The drone's local map at the same moment is presented in Figure 25a. On the map, there was no obstacle in front of the drone. A close shot of the branch that was not detected is presented in Figure 25b.

Since the needleless branch was not detected at all, it was removed after the first flight to reduce the risk of hardware breaks. At the same time, the maximum flight speed was increased to 1.0 m/s, and the drone managed to navigate to the goal without problems in the following three 13 m flights.

In the fifth flight, the distance to the goal was changed from 13 m to 18 m. At the beginning of the fifth flight, the system performed a successful emergency stop, when it detected a previously undetected dry branch right before it. After the emergency stop, the system successfully planned a new path and continued toward the goal. This time the goal was not in a clear area, but behind a spruce and right next to another spruce, and the system did not manage to navigate to the goal. The system performed another successful emergency stop in front of the spruce and started to search for a suitable path, but it was manually landed when it had not found the path even after a 25 s long hovering. The approximate position of the goal in the left



Figure 22: Sparse mixed forest.

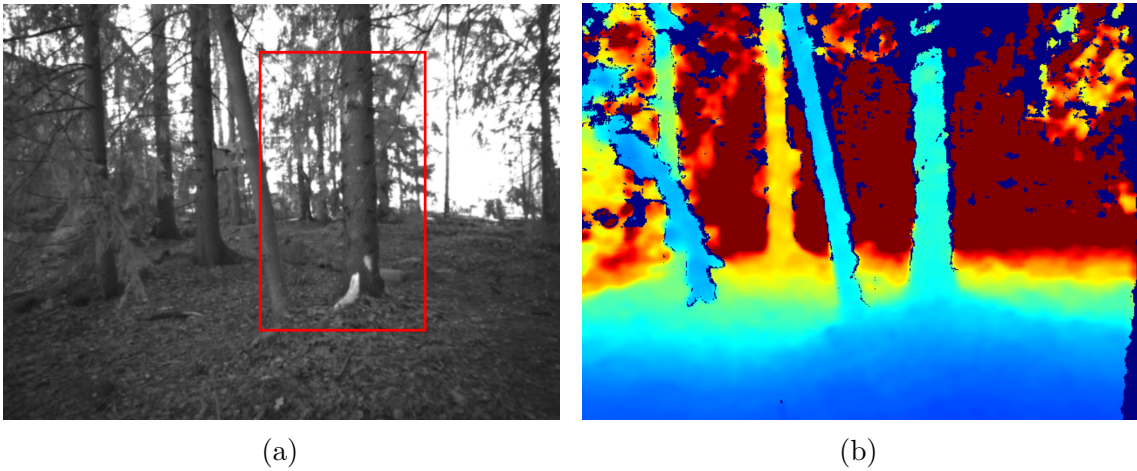


Figure 23: (a) Left stereo image and (b) depth image just under a second before the collision to small branch.

onboard camera is marked in Figure 26a. Figure 26b presents the local map during the hovering.

For the sixth flight, the takeoff location and the distance to the goal were given as same as in the fifth flight, but the initial yaw-pose of the drone was rotated so that the goal was now in the clear area. At the beginning of the sixth flight, the drone performed again a successful emergency stop in front of the same branches as in the first emergency stop of the fifth flight, and after that navigate to the goal

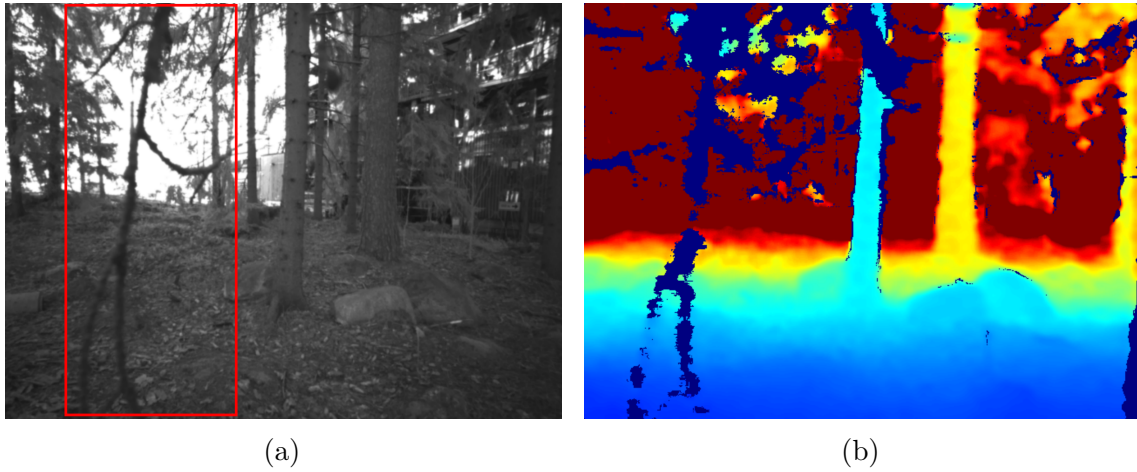


Figure 24: (a) Left stereo image and (b) depth image during the collision to a small branch.

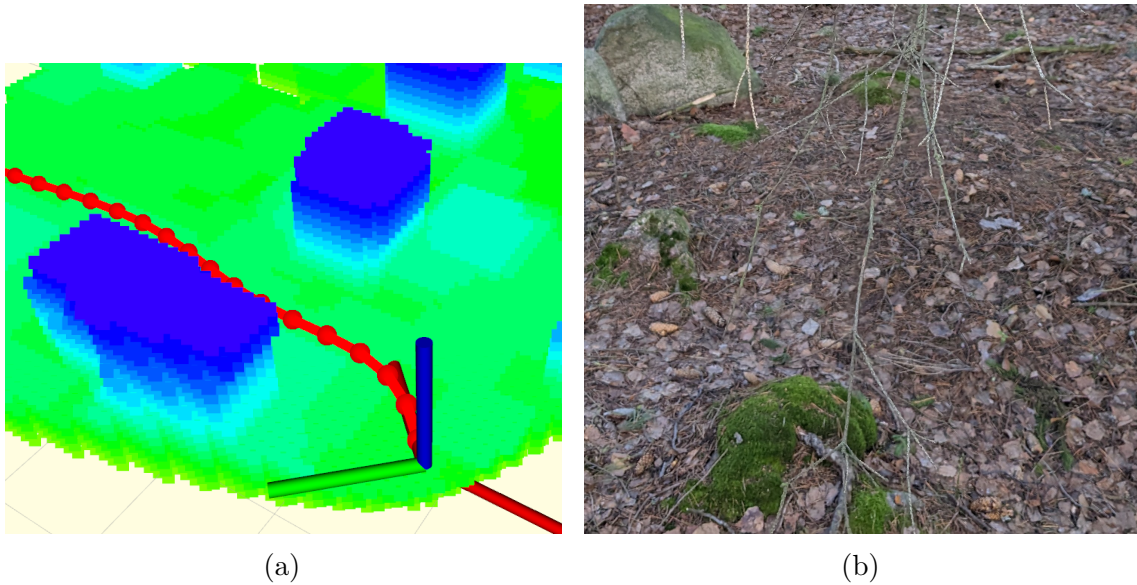


Figure 25: (a) The local map and planned path during the collision (b) Close shot of the small needleless branch that was not detected by the drone.

without problems.

For the last three flights, the takeoff location was changed so that the forest between the takeoff location and the goal was denser. The drone was able to successfully navigate to the goal on all three flights. On the seventh flight, the system performed one emergency stop, and on the eighth flight, there were no emergency stops. In the ninth flight, the system planned the path so that it got stuck behind a leafless bush, which was not fully detected until the system was in front of it. The system managed to bypass the bush, but there were in total five emergency stops on the ninth flight. Figure 27a presents a spruce branch, that was successfully avoided in the last three flights. The difference to the problematic branch in Figure 25b was

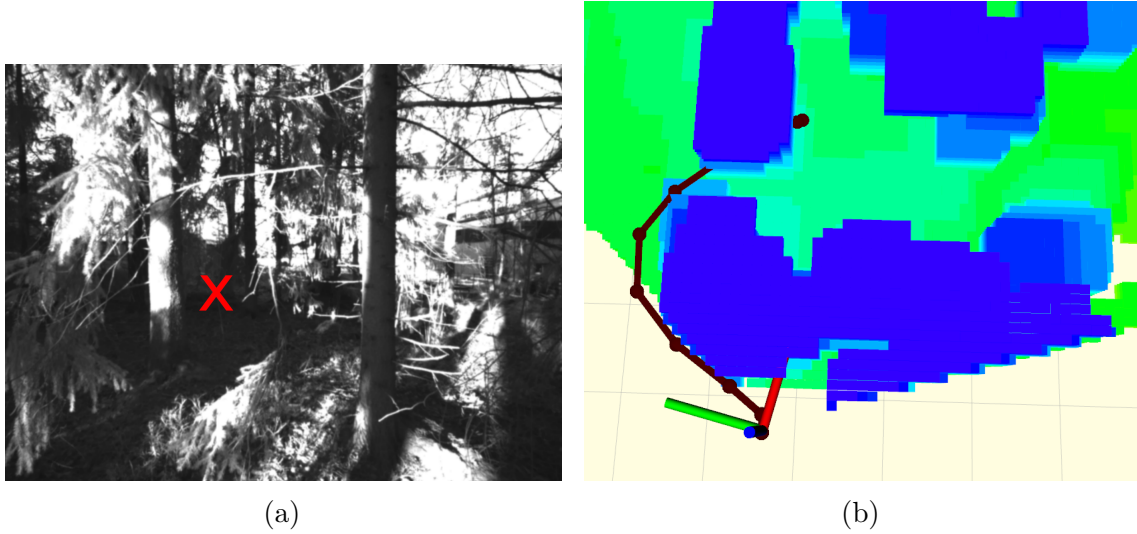


Figure 26: (a) The approximate position of the goal in the fifth flight marked in the left stereo image. (b) The local map from up during the hovering in the fifth flight. The system did not find a suitable path even though there were gaps in the map. A failed path iteration from the location of the system to the goal is shown in the picture.

that there were some needles and hence the branch was better detected. Figure 27b presents the map and the path of the eighth flight as an illustrating example.

For the last three flights from the second takeoff location, also the approximated landing positions were marked. The measured distances between the marks were approximately 30 cm, 30 cm, and 50 cm. The deviation was largest in the y-direction. The distance from the takeoff location to the approximate center of the landing locations was measured with a laser rangefinder, and the measurement result was 18.008 m.

A summary of the test flights performed in a sparse mixed forest environment is presented in Table 4.

The results showed that the performance of the system was good and reliable in a sparse forest. Only one flight failed, and in that flight, the goal was in a difficult place close to a tree. The flight distances in these experiments were quite short for the evaluation, but the localization was accurate and precise. Even though there was some deviation between the landing positions, most of it was in the y-direction. Since the initial heading was determined by the hand, a small deviation in the y-direction is natural.

The results also showed that the system had difficulties with detecting small needleless or leafless branches. Bigger dry branches were detected a couple of times so late, that the system had to perform an emergency stop to avoid a collision. An emergency stop was also performed when the system did not find a path to the goal in the only failed flight. All emergency stops were successful. In the first flight, the system did not detect a very thin dry branch at all and flew through it. That indicates that very thin branches could cause collisions.

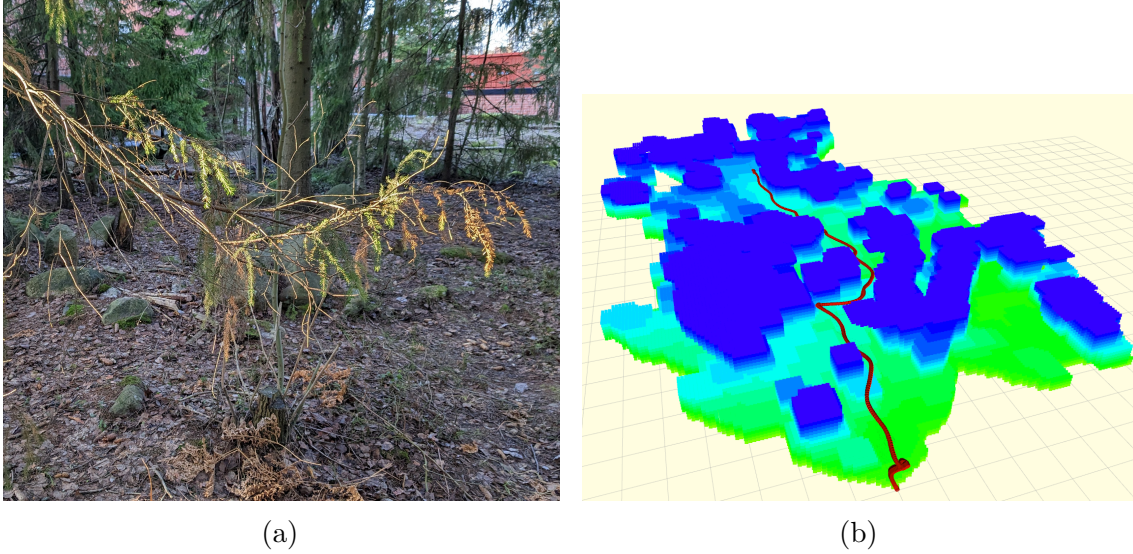


Figure 27: (a) Successfully avoided branches. Behind the spruce branch is the leafless bush that was bypassed in the ninth flight. (b) The map and the path of the drone in the eighth flight.

6.3 Flying in a park woodland with dense understory vegetation

6.3.1 Environment and the experimental setup

The second test area was a park located in Otaniemi, Espoo (Hagalundinpuisto, $60^{\circ}10'58.8''\text{N}$ $24^{\circ}49'18.1''\text{E}$). The trees in the area were mainly big oaks, but the area had dense understory vegetation. Since the test flights were performed in January, the trees and understory vegetation did not have leaves. The weather was partly cloudy and during the sunny moments, the drone's onboard camera was pointing against the sunlight. The sun was shining during the third, fourth, and fifth flights. The wind was moderate. The wind speeds during the flight tests are presented in Appendix A. Figure 28 presents a picture of the drone during the test flight giving also a general look at the test area. Despite being still a relatively flat environment, the terrain of the park was more uneven than in the sparse forest presented in the chapter 6.2.

In total, nine test flights were performed. Since there was no exact positioning system available, the relative 3D goal positions given as inputs were determined by walking with the system from the start points to the flat goal area and reading the VIO position estimate. In the first three flights, the given goal was 21.0 m forward, 1.0 m left, and 1.25 m up from the start position. In these flights, the maximum velocity was set to 1.0 m/s. After three flights the start position was moved. The goal area remained the same, but the flight distance was longer and there was also a gentle rise in the terrain between the start and the goal area. Since also the z-coordinate of the goal point is given with respect to the start position, the rise in the terrain was taken into account in the given goal point. The given goal for the

Table 4: Summary of the test flights performed in the sparse mixed forest environment.

Flight No.	Takeoff location	Goal (xyz)	Max. Vel.	Flight successful
1.	First	[13.0, 0.0, 1.0]	0.5 m/s	Yes
2.	First	[13.0, 0.0, 1.0]	1.0 m/s	Yes
3.	First	[13.0, 0.0, 1.0]	1.0 m/s	Yes
4.	First	[13.0, 0.0, 1.0]	1.0 m/s	Yes
5.	First	[18.0, 0.0, 1.0]	1.0 m/s	No. The system failed to find a path after emergency stop and was manually landed.
6.	First	[18.0, 0.0, 1.0]	1.0 m/s	Yes
7.	Second	[18.0, 0.0, 1.0]	1.0 m/s	Yes
8.	Second	[18.0, 0.0, 1.0]	1.0 m/s	Yes
9.	Second	[18.0, 0.0, 1.0]	1.0 m/s	Yes

rest six flights was 40.0 m forward, 3.0 m left, and 2.25 m up from the new start position. In these flights, the given maximum velocities varied between 1.0 m/s and 1.5 m/s. The maximum velocity was 1.0 m/s in the fourth, eighth, and ninth flights, 1.25 m/s in the seventh flight, and 1.5 m/s in the fifth and sixth flights. The virtual ceiling restricting the flying altitude was also adjusted. During the first three flights, the ceiling was on 2.0 m, during the fourth and fifth flights at 3.0 m, and during the last four flights at 4.0 m. The takeoff altitude was 1.0 m on all flights.

6.3.2 Results

The leafless branches of the understory vegetation turned out to be very difficult for the system to detect. Sometimes the system did not detect the branches at all, and also successful flights contained many emergency stops due to the late detection of thin branches.

In the first test flight, the system successfully navigated to the goal, but one of the propellers did have contact with a leafless branch. One small branch was also detected so late that the system performed a successful emergency stop. In the second flight, the system successfully navigated to the goal again, but it experienced multiple collisions with small leafless branches during the flight. A couple of the collisions between propellers and branches were not notified by the system. The system performed also eight emergency stops due to the late detection of branches.



Figure 28: The drone flying in park woodland with leafless understory vegetation.

Four of those emergency stops were performed too late, and the system did collide with branches but still managed to recover and plan a new path after the stop. Figure 29a presents the left stereo image at the time when the first emergency stop was activated and Figure 29b presents the depth image at the same timestamp. The branches were detected and the emergency stop was activated so late that the drone collided with the branch. After the emergency stop, the branch disappeared from the map, and the system planned the path again via the same gap and collided with the same branch again. The system needed four emergency stops to get past the first trees since thin and leafless branches always disappeared from the map after the emergency stop.

On the third flight, there was a notable difference in the weather conditions compared to the first two flights, since the sun started to shine against the camera. The system was not able to fly to the goal. It performed two successful emergency stops at the beginning of the flight and got past the first trees, but got stuck in front of the next small tree. The system performed again an emergency stop but failed to find a new path after that, even though there would have been enough space for bypassing the tree. After 15 seconds of hovering and failing to replan the path, the system was manually landed. During the hovering, one propeller of the drone was hitting to small tree. The drone did not detect that, since the propellers are not visible in the camera. Figure 30a presents the left stereo image at the time the system was stuck in front of the small branch and Figure 30b presents the local map at the same timestamp.

For the fourth flight, the starting location was moved further from the goal area.

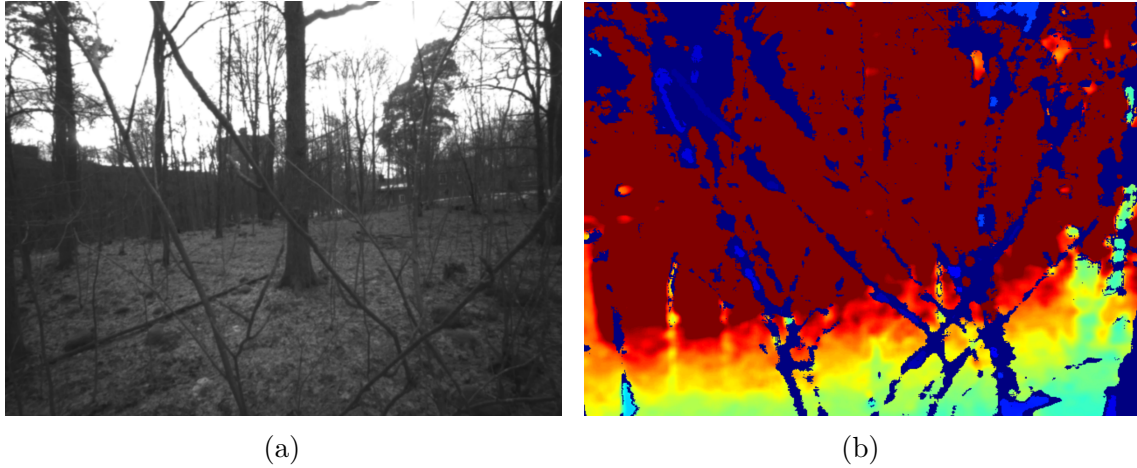


Figure 29: (a) Left stereo image at the time when the first emergency stop of the second flight was activated (b) Depth image at the time when the first emergency stop of the second flight was activated.

The system performed three emergency stops due to the late detection of branches. One of the stops was performed so late, that the drone firmly touched the branch, but the system was able to successfully navigate to the goal despite the longer flight distance and the sunlight against the camera.

The last five flights were all unsuccessful. During the fifth flight, the system performed in total four emergency stops. One of the stops was performed so late that the drone had contact with thin and flexible leafless branches, but otherwise, the drone avoided all obstacles well. However, the system did not manage to navigate to the goal, since the system considered the goal point to be under the ground, even though the start position and the given goal were the same as in the fourth flight. Since there was no exact positioning system available, it is not possible to determine in which of the flights the VIO estimate in the z-direction was more accurate, but there clearly was a notable deviation among the estimates.

At the beginning of the sixth flight, the system flew through small branches without detecting them, and after that performed one successful emergency stop. In the mid-flight, the system collided with a relatively thick tree. The planner stopped working after the collision, the system moved to auto-hover mode and was then manually landed. The system tried to perform an emergency stop before the collision, but the tree had been detected too late. Figures 31a and 31b present the left stereo image and the depth image approximately 0.8 s before the collision respectively. At that timestamp, the trees at the left were detected on the map already, but the trees right in front of the drone were not.

The seventh flight started well, and the system managed to navigate to the end with two successful emergency stops and only one touch to a thin and flexible branch. However, in the end, the system considered again the goal point to be under the ground, and the system did not reach the goal.

In the eighth flight, the system started flying towards a very dense leafless tree. The system detected the tree too late and performed an emergency stop, but still hit

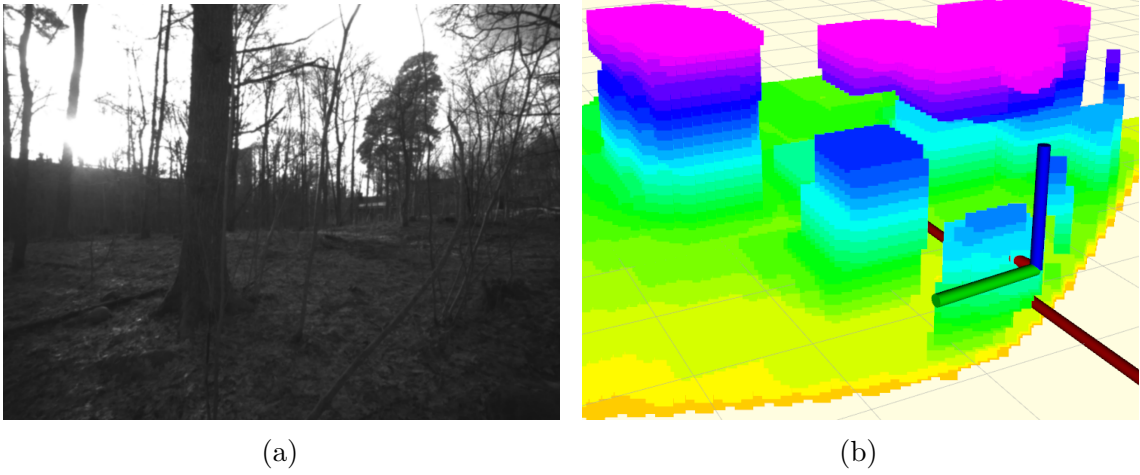


Figure 30: (a) Left stereo image at the time the system got stuck in the third flight. In the forefront the leafless branch that blocked the path. On the left the sun, that was shining against the camera. (b) The local map at the time the system got stuck on the third flight. The system was not able to find a suitable path past the branch that was in front of it.

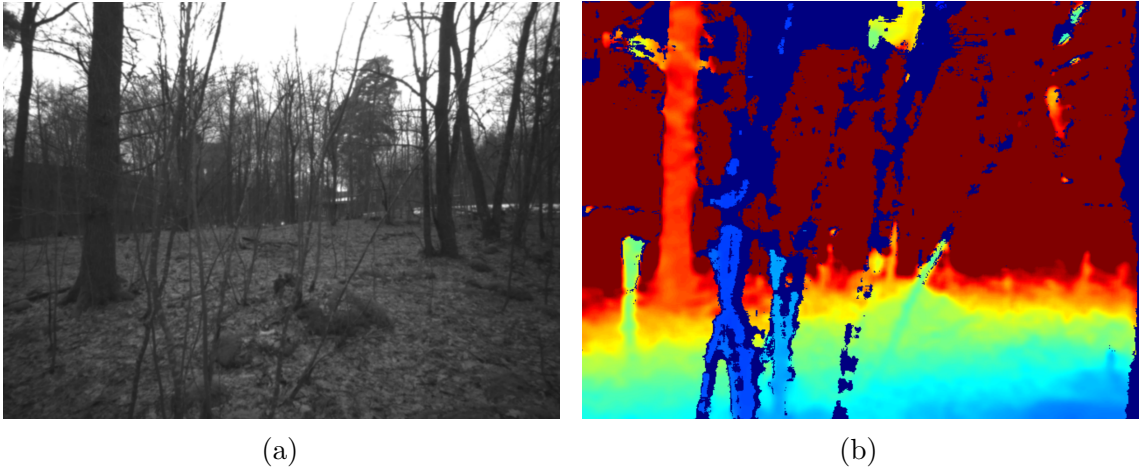


Figure 31: (a) Left stereo image approximately 0.8 s before the collision in the sixth flight. (b) The depth image at the same timestamp.

thin and flexible branches. The system got stuck in front of the tree. After hovering for approximately eight seconds, the system found a new path but performed a new emergency stop right after. In the end, approximately 15 s after the first emergency stop, the system was manually landed since one of the propellers was constantly hitting a branch.

In the ninth flight, the system started flying towards the same dense tree as in the eighth flight. The system performed the emergency stop too late again and hit thin branches. In this flight, the system managed to find a suitable path over that tree after the emergency stop. However, after multiple contacts with undetected branches and one successful emergency stop, the system performed a late emergency

stop in front of a leafless tree and started hovering and replanning. After hovering for approximately 18 s the system was manually landed.

A short summary of the test flights performed in a park woodland environment is presented in Table 5.

The results showed that the reliability of the obstacle avoidance was not good in the dense leafless understory vegetation, and most of the test flights failed. The system had problems detecting thin branches and even in the successful flights, the system had contacts with branches due to late detection. Sometimes the small branches were not detected at all, and the system straightly collided with them, and many times the branches were detected so late that even an emergency stop was not enough to prevent the collision.

6.4 Flying in a snowy spruce forest

6.4.1 Environment and the experimental setup

The last flight tests were performed in a dense spruce forest. The test area was located in Paloheinä, Helsinki (60°15'28.4"N 24°55'19.9"E). The tests were performed on two different days. The environmental conditions were similar on both days. The weather was mostly cloudy, and the dense forest mostly covered the sun also during the sunny moments. The ground was covered with snow, and the wind speed was moderate. The wind speeds during the flight tests are presented in Appendix A.

In total, 10 test flights were performed during the first test day. The takeoff location was the same in all flights, but there were three different goal areas. The first goal was in a small open area. The coordinates of the goal were again obtained by walking with the drone from the takeoff location to the goal and recording the VIO estimate. In the first four flights, the given goal was 37.0 m forward, 1.5 m left, and 1.5 m up from the start position. After four flights the goal was changed to 35.0 m forward, 1.5 m left, and 1.5 m up from the start position since the drone was flying past the open area.

After the fifth flight, the goal was changed to an adjacent small open area. The goal coordinates were kept as same, but the initial heading before the takeoff was rotated so that the drone was heading toward the new goal. The path to the second goal contained denser trees with more low branches.

For the last test flight of the first day, the goal was set to 60.0 m forward, 1.5 m left, and 1.5 m up from the start position. The initial heading before the takeoff was rotated back to the original one.

To estimate the forest density in the test area, the environment was scanned with a backpack 3D LiDAR. The used LiDAR model was GeoSLAM ZEB Horizon [107], and the scanning results were processed to point cloud with GeoSLAM Connect software [108]. The density was then estimated from the obtained point cloud by defining a rectangular area and calculating the number of trees in that area. The estimated forest density between the takeoff location and the first and the second goal was ≈ 0.238 trees/m². The trees in that area were mainly young spruces containing many dry low branches. The forest behind the first two goals, towards the last goal,

Table 5: Summary of the test flights performed in the park woodland with dense understory vegetation.

Flight No.	Takeoff location	Goal (xyz)	Max. Vel.	Sunny	Flight successful
1.	First	[21.0, 1.0, 1.25]	1.0 m/s	No	Yes
2.	First	[21.0, 1.0, 1.25]	1.0 m/s	No	Yes
3.	First	[21.0, 1.0, 1.25]	1.0 m/s	Yes	No. The system failed to find a path after an emergency stop and was manually landed.
4.	Second	[40.0, 3.0, 2.25]	1.0 m/s	Yes	Yes
5.	Second	[40.0, 3.0, 2.25]	1.5 m/s	Yes	No. The goal was under the ground due to the drifting of VIO.
6.	Second	[40.0, 3.0, 2.25]	1.5 m/s	No	No. The system collided, and the planner stopped working after that.
7.	Second	[40.0, 3.0, 2.25]	1.25 m/s	No	No. The goal was under the ground due to the drifting of VIO.
8.	Second	[40.0, 3.0, 2.25]	1.0 m/s	No	No. The system failed to find a path after an emergency stop and was manually landed.
9.	Second	[40.0, 3.0, 2.25]	1.0 m/s	No	No. The system failed to find a path after an emergency stop and was manually landed.

was sparser and consisted mainly of older trees with fewer low branches. The forest density in that area was ≈ 0.165 trees/m².

Figure 32 presents the obtained point cloud from up. The tree tops are cut from

the model. The views towards the takeoff location from the first two goal areas are presented in Figures 33a and 33b.

During the second test day, nine test flights were performed. The takeoff location was approximately the same as on the first test day and the takeoff heading was approximately the same as on the first flights of the first test day. The aim of the test flights on the second test day was in performing longer test flights. Two goals were used. The nearer goal was set to 60.0 m forward, 1.5 m left, and 1.5 m up from the start position, and the further goal was set to 80.0 m forward, 1.5 m left, and 1.5 m up from the start position.

Maximum flying velocity was set to 1.0 m/s on all flights during both days.

6.4.2 Results

The first flight test failed and ended on a manual emergency landing. The system performed a successful emergency landing after a late detection of a small dry branch, but after the stop the position estimate produced by VINS become unstable. Due to the unstable position estimate the drone started to shake and was manually landed. The reason for the sudden instability of the position estimate was not found in the bagfile recorded during the flight.

The second test flight failed during the takeoff. The position estimate from VINS failed so the estimate moved down when the system moved up. The system performed a fast acceleration toward the sky and the system was manually stopped with a killswitch. VINS reported numerical instability in preintegration before the takeoff.

The third flight was the first successful one. The drone performed four successful emergency stops due to the late detection of small branches but managed to navigate to the goal without collisions. The fourth test flight was also successful and the system did not need any emergency stops during the flight.

For the fifth test flight, the x-coordinate of the goal was moved 2 meters nearer, since in the previous flights the system had flown past the open goal area. The flight was again successful. The drone performed one emergency stop too late and had a touch with a small branch, but managed to continue the navigation.

For the sixth test flight, the initial heading of the system was rotated towards the second goal. The system was able to successfully navigate to the goal with three successful emergency stops. The system also flew through one needleless branch without detecting it.

The seventh flight failed. The system had difficulties getting past the densest part of the path, and one of the propellers loosened up due to a hit with a tree causing a need for a manual emergency landing.

The eighth and the ninth test flights both failed. On the eighth flight, the reason was dying of the planner after a collision with a small branch that was detected too late. The system performed an emergency stop but too late. After the death of the planner, the system entered into hovering mode and it was manually landed. On the ninth flight, the system performed an emergency stop in front of a dry branch. The emergency stop was successful, but the system did not find a new path and entered

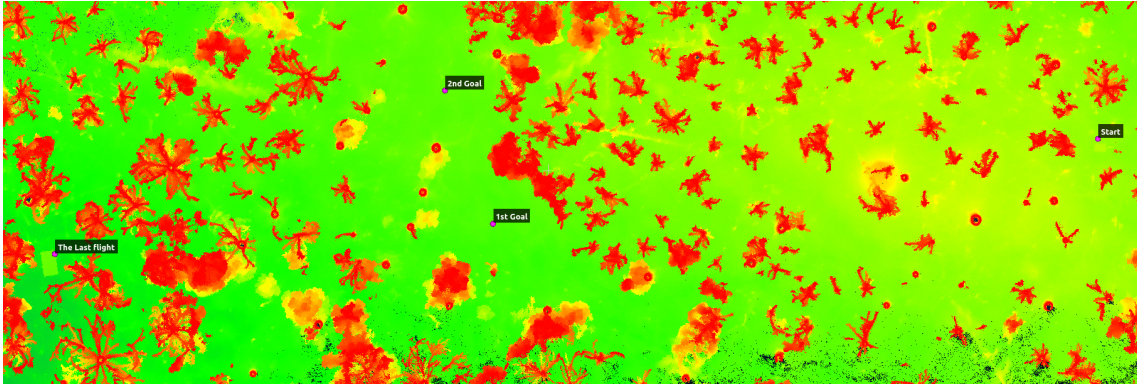


Figure 32: The point cloud of the third test area. The approximate locations of the takeoff location, two first goal areas, and the landing point of the last flight of the first test day are marked to the cloud. The cloud is cut so that the highest points visible are 2 m higher than the takeoff location.

into the hovering mode to continue path searching. The system was manually landed since one of the propellers was hitting a branch during the hovering.

For the tenth flight, the goal was moved further away and the takeoff heading was turned to the same that it was in the first flights. The system collided multiple times with branches during the flight but managed to continue flying and navigating after every collision. However approximately 6 m before the goal the system landed automatically. The ROS side of the system did not give the landing command, so the landing command came from the PX4 Autopilot software. The most probable reason for the landing is that system detected the battery level to be critically low and performed an automatic emergency landing. The battery was low when it was manually measured after the flight.

The test flights of the first day in the snowy spruce forest are summarized in Table 6.

In the first flight test of the second day, the given goal was the same as in the last flight test of the first test day. The system performed five successful emergency stops during the flight and managed to navigate to the goal. The goal was inside a tree. When the system detected that, it automatically changed the position of the goal ≈ 70 cm so that the goal was in the obstacle-free area.

The second flight failed already in the takeoff. The drone started to oscillate right after taking off and the killswitch was manually triggered. Based on the recorded rosbag, the reason was again the instability of the VINS position estimate. However, rosbags have limitations in their ability to exactly duplicate the timing of messages [109]. For that reason, there exists also a minor possibility that the reason for the oscillation of the system was somewhere else, and the instability of the position estimate was caused by that oscillation itself.

For the third flight the goal was changed to the further one, but the flight failed soon after the takeoff. The system performed a successful emergency stop in front of a tree, but when the system rotated after finding a new path, the propellers hit multiple times to branches and the system was manually landed.



Figure 33: (a) The view from the first goal area towards the takeoff location. (b) The view from the second goal area. The takeoff location is on the horizon on the right side of the picture.

The fourth flight failed right after the takeoff due to a poor global path. The system planned the initial global path so that it did try to go below the ground. Planning the global path under the ground could have been prevented by setting the virtual floor of the map to the real ground level. However, the virtual floor was set to -1 m to allow the drone to plan paths below the takeoff altitude since the altitude of the ground near the goal was slightly lower than in the takeoff area. The reason for the failure is illustrated in Figure 34a.

For the fifth flight, the goal was set back to the nearer one due to a low battery level. The flight failed when the system was flying toward a spruce that had multiple dry low branches. The system detected the branches late and planned a new path, but there was another branch right next to the system. When the system rotated to follow the new path, one of the propellers hit the branch so that one of its blades broke, and the system was manually landed. The tree is shown in Figure 34b.

In the sixth test flight, the system navigated to the nearer goal successfully. The system performed three emergency stops during the flight, of which two were successful. One emergency stop was performed so late that the system had a touch with a small branch, but managed to continue the mission after that.

For the seventh test flight, the battery was changed to a new one, and the goal was changed back to the farther one. The system performed an emergency stop

Table 6: Summary of the test flights performed at the first day in the snowy spruce forest.

Flight No.	Takeoff heading	Goal (xyz)	Max. Vel.	Flight successful
1.	First	[37.0, 1.5, 1.5]	1.0 m/s	No. The system was manually landed after the position estimate became unstable.
2.	First	-	1.0 m/s	No. The initialization of VINS failed leading to a failure in the takeoff.
3.	First	[37.0, 1.5, 1.5]	1.0 m/s	Yes
4.	First	[37.0, 1.5, 1.5]	1.0 m/s	Yes
5.	First	[35.0, 1.5, 1.5]	1.0 m/s	Yes
6.	Second	[35.0, 1.5, 1.5]	1.0 m/s	Yes
7.	Second	[35.0, 1.5, 1.5]	1.0 m/s	No. one of the propellers loosened up after hitting a branch and the system was manually landed.
8.	Second	[35.0, 1.5, 1.5]	1.0 m/s	No. The system collided with a branch, and the planner stopped working after that.
9.	Second	[35.0, 1.5, 1.5]	1.0 m/s	No. The system failed to find a path after an emergency stop and was manually landed.
10.	First	[60.0, 1.5, 1.5]	1.0 m/s	Yes and No. The PX4 performed an emergency landing 6 m before the goal due to a low battery level.

halfway through the flight and entered into hovering mode to search for a new path. During the hovering, the propellers hit multiple times to branches, so the system was manually landed.

For the eighth flight, the goal was again set back to the nearer one. The flight was successful. The system performed four emergency stops during the flight without collisions.

For the last test flight, the goal was set to the farther one. The system managed to successfully navigate to the goal. During the flight, it performed six successful

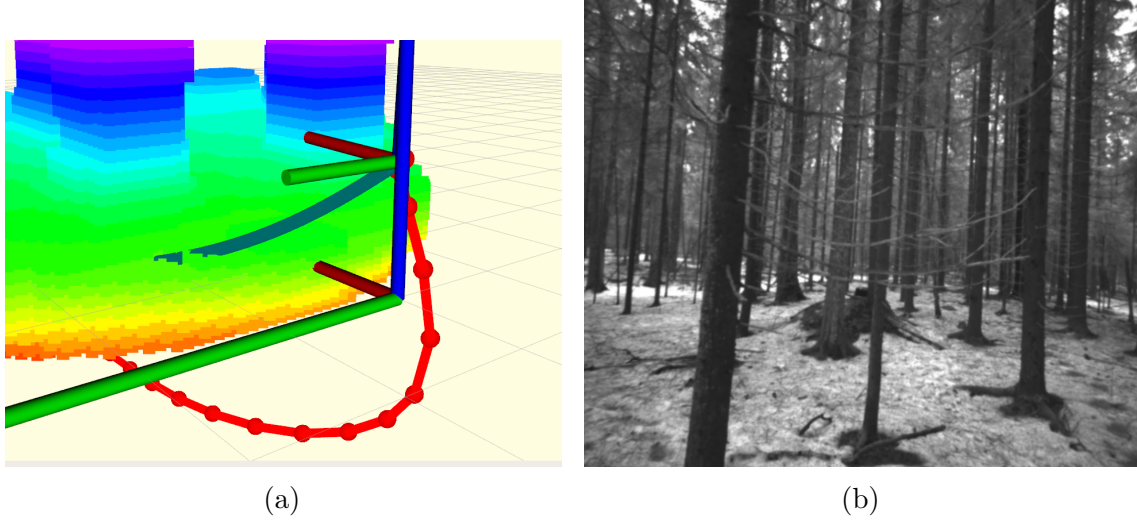


Figure 34: (a) The failure of the fourth flight. The system planned the global path (grey) to go below the ground. The forward pointing camera did not see the area below the drone, so the system assumed that to be a free region, and returned an optimal path (red) that also went below the ground. (b) The tree that caused the failure in the fifth flight of the second test day.

emergency stops.

The test flights of the second day in the snowy spruce forest are summarized in Table 7.

For the flights performed during the last test day, also the precision and the accuracy of the VINS were roughly estimated with a laser range meter and measuring tape. Estimating the flying distance with a laser range meter gave only rough estimates since the straight path from the takeoff location to the landing position contained branches and trees. For that reason, the laser range meter was held on the position next to the takeoff position, instead of the exact takeoff position.

The measured distance from the takeoff position to the landing position of the last flight was 78.6 m. The xy-position reported by VINS after the landing was approximately (79.57 m, 1.52 m), so the flying distance reported by VINS was $\sqrt{(79.2 \text{ m})^2 + (1.52 \text{ m})^2} = 79.58 \text{ m}$.

For the three successful flights to the nearer goal position, there were three landing positions. The maximum distance between these landing positions was measured with a measuring tape and the approximate distance was 2.2 m. The deviation between the position was mainly in the y-direction. The distance measured with a laser range meter to the approximate center of the three landing positions was 58.5 m. The deviation in the y-direction was also expected since the initial heading of the system was set manually without any exact measurements. With a 60 m flying distance, a 1° difference in the takeoff heading results in approximately a 1 m difference to the goal position with perfect localization.

The results showed that the reliability of the navigation in this environment was not good, but better than in the park woodland. Nine of 19 test flights can be

Table 7: Summary of the test flights performed at the second day in the snowy spruce forest.

Flight No.	Goal (xyz)	Max. Vel.	Flight successful
1.	[60.0, 1.5, 1.5]	1.0 m/s	Yes
2.	-	1.0 m/s	No. The system was manually landed after the position estimate became unstable during the takeoff.
3.	[80.0, 1.5, 1.5]	1.0 m/s	No. The system was manually landed since the propeller was hitting a branch during replanning.
4.	[80.0, 1.5, 1.5]	1.0 m/s	No. The system tried to go under the ground due to a poor global path.
5.	[60.0, 1.5, 1.5]	1.0 m/s	No. The system broke one of its propellers by hitting a branch.
6.	[60.0, 1.5, 1.5]	1.0 m/s	Yes
7.	[80.0, 1.5, 1.5]	1.0 m/s	No. The system was manually landed since the propeller was hitting a branch during replanning.
8.	[60.0, 1.5, 1.5]	1.0 m/s	Yes
9.	[80.0, 1.5, 1.5]	1.0 m/s	Yes

considered successful, so less than half of the flight tests succeeded. The system had again problems with the detection of small needleless branches, and late or completely missed detection of those caused again emergency stops and collisions. However, all failures occurred in the first part of the forest, where the forest density was very high (≈ 0.238 trees/m²). There were no failures in the later part of the forest even though the forest density was relatively dense also there (≈ 0.165 trees/m²). When the system navigated past the densest part of the forest, it managed to navigate to the goal with a 100 % success rate, even though it had firm contacts with small branches in some flights also in the sparser area.

The results also showed that the accuracy and the precision of the VINS-Fusion estimate were good, but the deviation and error were bigger than in the shorter flights in Chapter 6.2. The accuracy of the estimation was sufficient for the task even in the longest flight. However, there were three test flights that failed due to the unstability of the estimation or estimate initialization failure. These problems did not occur in other environments, and the reasons behind them remained unknown.

6.5 Discussion

The literature survey in Chapter 3 showed that although autonomous flying without GNSS positioning has been an actively researched topic, there exist only a few solutions with an open source code. From the perspective of boreal forest research, most of the existing open-source systems had been flight tested in significantly different forest environments. Also in most cases, the papers mentioned only one or two successful test flights through a forest without specifying the density of the forest or the robustness of operation. The method by Loquercio et al. [49] was the only solution presented in the survey, whose reliability was evaluated in the original paper by flying multiple tests with the same path. For these reasons, the original articles presented in the literature survey contain very few results against which the results of this work could have been compared.

In the original paper by Zhou et al. [63], the system was tested in swarm operations. The paper presented an experiment where a swarm of drones was flying through a dense bamboo forest, but the paper did not report an evaluation of the reliability of the system by performing the test multiple times or in different forest environments. In this thesis, the system was evaluated with one drone by flying multiple flight tests in different types of forests.

Based on the flight tests performed in chapter 6, the weakness of the system is the depth map generation. The system has only one forward-looking stereo depth camera. That makes it possible to run the system in very light drones, as shown in the original article, but in dense forests containing small dry branches relying on only one camera is not enough for reliable operation. As shown in the experiments of chapter 6.4, the system can successfully navigate even through very dense forests, but the performance is not reliable. The results showed, that sometimes the system was able to avoid dense small branches with very smooth trajectories, but sometimes the system failed to fly to the goal.

The problems in the sensing were related to both camera resolution and the field of view. Since the system had only one forward-looking camera, it sensed only the obstacles right in front of it. That led to situations, where the propellers hit branches or trees when the system had entered into hovering mode after an emergency stop. Since the system maintains the map only from the currently sensed objects, the objects next to the system are not on the map. The system also does not detect the small leafless branches and trees early enough. In some flights, the system did not detect the branches until they were so close that it was not possible to triangulate them, and the system collided without even detecting the branch before the collision. Increasing the resolution of the depth images could improve the ability to detect the small branches. The maximum depth image resolution of the RealSense D435 is 1280×720 . However, using the same camera for mapping and VIO prevents the possibility of increasing the resolution. The camera driver needs depth images and infrared stereo images to have identical resolutions and identical publishing rates, and VINS-Fusion was not able to run in real-time with the increased resolution with the onboard computer ASUS PN51-E1.

The system had problems with the detection of needless and leafless branches

and trees in chapters 6.3 and 6.4, but the performance of the system in the sparse forest in chapter 6.2 was good. The flying distances in that forest were short, but the system failed only once. In the only failed flight, the location of the goal was difficult. The system failed to plan a suitable path to the goal even though there were free regions on the map. The problem was that the system failed to generate a path that would fulfill the limits for maximum velocity, acceleration, and jerk. By increasing the limit values for acceleration and jerk, the system could have found a suitable path to the goal also in the only failed flight. By increasing those values, the system could also have planned more aggressive paths in other flights. However, the limits of the drone platform or the trajectory tracking controller were not evaluated in this study, so it remains unknown how much those values could have been increased without losing the ability to follow the planned path.

One flight test in chapter 6.4 ended right at the beginning of the flight since the planner returned an initial global path that went aggressively below the ground level. The planner generates the global path so that it is between the virtual floor and virtual ceiling, which restricts the flying altitude. However, if the altitude of the ground during the path is not flat, the floor and the ceiling need to be set so that even a single waypoint during the path is not below the floor or over the ceiling. Therefore, in non-flat areas, the ceiling has to be set on a very high level or the floor to a very low level, depending on the altitude of the takeoff location. In the worst case, that can lead to a poor initial global path as demonstrated in chapter 6.4, but it also makes it hard to restrict the flying altitude of the drone in non-flat areas. The system is clearly designed for flat areas, which is problematic in typical boreal forests, and requires further development.

Despite the mistakes in the calibration of the VINS-Fusion, the performance of the localization was relatively good during the flight tests. In chapters 6.2 and 6.4 there were multiple test flights with the same goal, and the precision of the system seemed to be good. The deviation in the x-direction was very low. The deviation in the y-direction was bigger, but still relatively low. From these chapters, the test flights in chapter 6.4 were longer. In approximately 60 m long flights, the maximum distance between the landing points was 2.2 m, and the deviation was mainly in the y-direction. Differences in the initial heading of the drone are a possible explanation for the deviation in the y-direction since the initial heading was not measured with any exact method. Based on the measured distances between the takeoff location and the landing position, the accuracy of VINS-Fusion was also relatively good. In the longest flight performed in this thesis, the difference between the flying distance reported by VINS-Fusion and the laser range meter value was approximately one meter.

The deviation in the z-direction was not measured, but in the flights performed in chapter 6.3 there was a problem that the goal was below the ground in two of the flights. One flight with the exactly same takeoff location and goal was still successful, indicating that there was a deviation in the z-direction. Using the non-optimal calibration parameters in the configuration of VINS-Fusion is a likely reason for the deviation.

In chapter 6.4 the system failed two times right after the takeoff due to failed

initialization of VINS-Fusion and once during the flight due to an unstable VINS-Fusion estimate. The reasons for the failed initialization and suddenly unstabilizing estimate remained unknown, but the non-optimal calibration is a possible explanation for these failures also. Another possible explanation is some failure in the IMU data due to cold weather or hardware break. Since these failures happened during the last test days, the system was not tested after that, so it is unknown if the problem will occur again.

Evaluating the performance of the VINS-Fusion is difficult due to non-optimal configuration parameters. It is not known, how good the performance would have been without mistakes during the calibration of the sensor suite. However, even with the non-optimal configuration, the performance of VINS-Fusion was good enough for over 80 m long flights. In longer flying missions, drift is always unavoidable, and correcting the position estimate with loop closures, collaborative sensing systems (e.g. the UWB used in the original article [63]) or GNSS is necessary.

The test flights aimed to evaluate the reliability of the navigation in various environments, study what kind of obstacles cause collisions or emergency stops, and evaluate the accuracy of localization. For the two first questions, the obtained results were comprehensive and it can be quite reliably concluded that the reliability was satisfying in a sparse forest but decreased when the forest density or amount of understory vegetation increased. It was also widely shown that needleless or leafless thin branches are hard to detect and hence cause emergency stops or even collisions for the system. However, the evaluation of the accuracy or precision of the VINS-Fusion could have been more comprehensive or accurate. The results obtained here were quite rough, and both a better calibration of the sensor suite and more precise measurement methods would be required for making more precise evaluations.

7 Discussion

This thesis started with a literature survey of existing open-source solutions for autonomous flying in GNSS-denied forest environments. Based on the literature survey, EGO-Planner-v2 proposed by Zhou et al. [63] was chosen for an experimental evaluation. The experimental tests were performed both in a simulator and in real forest environments with a custom drone platform, and the results of those tests were already discussed in Chapters 5.5 and 6.5 respectively. This chapter concentrates on discussing reached level of automation, potential improvements and applications for the system, and also the limitations of this study.

7.1 Limitations of the study

The evaluations of the performance in this study contained certain limitations. The mistakes in the calibration of the sensor suite made it hard to make conclusions about the real performance of VINS-Fusion since non-optimal configuration parameters inevitably affect the tracking result. In the snowy spruce forest, obtaining the ground truth value for flying distance with a laser range meter was a non-optimal method since there was no clear path from the takeoff location to the landing location. The better solution would have been marking the precise landing locations in the forest and recording the point cloud after the flights. From the processed point cloud, the true flying distance could have been determined better for every flight individually. It would also have been beneficial to determine the takeoff headings more precisely.

In the simulated experiments, computational burden was a limiting factor. Even though the simulations were performed on a powerful desktop computer and the tops of the tree models were cut off, the forest sizes were small. The small size of forests limits the usefulness of the simulated data in the evaluation of the performance of the system. The Gazebo simulator was an essential tool during the commissioning and initial testing of the system, but its usefulness in evaluating the performance of the system was questionable, and using another simulator, for example, Microsoft AirSim [110], could have been a better option.

This study did evaluate the effect of the flying speed on the performance only in the simulation. To get a more comprehensive evaluation of the performance of the system, it would have been better to make real-world test flights with slower flying speeds also. Based on the results of the simulated tests, the lower flying speed would probably have improved the ability of the system to avoid collisions. Especially in the dense spruce forest in chapter 6.4, a lower flying speed would probably have increased the success rate and decreased the number of emergency stops since the system would have had more time to react to small branches that were detected late. The reason for not using slower flying speeds was the limited amount of flying batteries. Flying at a slower speed increases the flying time and hence the battery consumption of the flight. However, it could have been better to have one more test day and perform test flights with slower flying speeds also.

7.2 Ways to improve the system and potential applications

Autonomous vehicles are typically categorized into different categories based on the level of automation. Many researchers have also discussed the autonomy levels of autonomous drones. Lee et al. [111] have proposed criteria for six levels of drone autonomy. The scale starts from Level 0, where the system has no automated assisting systems, and ends to level 5, where the system can operate autonomously in all possible scenarios making the human operator unnecessary. The system evaluated in this thesis falls between levels 3 and 4 on that scale. The system was able to move autonomously, avoid collisions, and generate three-dimensional paths autonomously. However, it did not fulfill the criteria of autonomous safe landings required in level 4. In addition, criteria of the level 4 define the human operator as optional. The reliability of the system was not high enough, and the operator had to monitor the system and be ready for emergency landings, even though the system could fly without an operator.

The performance of the system was promising, but to achieve a higher level of autonomy, the reliability of the system needs to be improved. The easiest way to improve the system without modifying the hardware is to re-calibrate the used sensor suite. Hardware synchronization between the camera and the IMU could also improve the performance of VINS-Fusion. The trajectory tracking controller could also be re-tuned to track the planned trajectories more precisely, even though the tracking performance was already satisfactory with current parameters.

Another natural way to improve the system would be using smaller hardware. In this thesis, the hardware was still relatively big and heavy compared to the drones in the original article. Smaller drones can more easily find suitable paths in dense forests, and decreasing the weight increases the flying time. On the other side, in case of a collision with small branches, a lighter drone would have more problems. Hence a better and more reliable ability to sense small branches is needed before the hardware can be replaced with a smaller one. A natural way to improve the sensing would be increasing the resolution of the depth camera, but also complementary sensors could be considered.

Using the same camera for mapping and VIO makes it possible to have only one camera, which makes the sensor suite lighter, but it also restricts the resolution used for mapping. One way to improve the sensing of the small branches would be to forgo the one-camera system and use different cameras for VIO and mapping. For example, Intel RealSense T265 [47] weighs 55 g and runs a computationally efficient built-in VIO algorithm. Replacing VINS-Fusion with that would make it possible to use the full resolution of RealSense D435 for mapping, which should improve the detection of small branches. With the current one-camera setup, a potential solution would be increasing the resolution of D435 to the maximum and then downsampling the grayscale stereo-images to VINS-Fusion. However, the downsampling process would also increase the computational burden of the system, and performing downsampling before forwarding the images to VINS-Fusion would increase the latency between image capture and location estimation update.

Another natural way to improve the system would be swarm operations. The

swarm operation ability was already included in the source code of the system, so expanding the system to those should be a straightforward operation. Having a swarm of drones would make it possible to efficiently explore large areas and also make it possible to use the systems drift-correction algorithm to increase the accuracy of VIO estimates.

The mapping algorithm of the system could also be improved. Virtual floor and virtual ceiling are simple ways to restrict the flying altitude. That solution works in flat areas, but if there are hills on the path, the flying altitude between the floor and the ceiling has to be set large. From point of view of potential future research applications, it would be better, if the permitted flying altitude would be with respect to the ground level so that the system would fly at the approximately same altitude even in hilly forests. This could be achieved e.g. by using altitude sensors or utilizing existing terrain models. The reliability of the system could also be improved by modifying the mapping of the system. By default, the map of the system covers only areas that are currently visible to the depth camera. That makes the mapping computationally lighter but has also a drawback that the system does not remember the branches next to it even though it has successfully mapped them earlier. That caused multiple times problems when the system entered hovering mode after emergency stops, and the propellers hit branches next to the system since the branches were not on the map.

In the future, the system could also be integrated into a high-level planner performing the area coverage. That would make longer flying missions possible and increase the usability of the system. However, in long flying missions correcting the VIO estimates would be necessary. Therefore, one possible improvement could also be a sensor fusion between GNSS and VIO. After that, the system would be able to navigate inside dense forests by relying on the VIO and correct the drift in the VIO estimate on open areas. VINS-Fusion already includes an option to fuse GNSS position estimates to the VIO estimates [112].

The system evaluated in this study still has limited operational usefulness. Since the system is not able to perform longer missions covering areas, it is not very useful for data collection yet. However, it is a good prototype of an autonomously navigating drone, and the system can be further developed on top of it. After the improvements in the reliability and the duration of missions, the system would have many potential applications in forest research.

The literature contains very few studies where under-canopy image data is collected. However, having the ability to fly under the canopy autonomously would make it possible to easily collect image data of understory vegetation in large forest areas. That data could have potential applications in biodiversity mapping as well as in detecting alien species endangering natural species. Flying under the canopy would also facilitate the collection of image data of tree trunks. There exist many studies estimating forest health from aerial images, but collecting data on the bark of the trees would open new options, for example, in bark beetle research.

Drones flying inside forests offer a platform for both scientific research and practical applications. Inside forest flying can offer a low-cost and efficient technique for forest inventory, such as measuring diameters on breast height, stem curvature, and trunk

anomalies. Currently, some indicators of forest inventory require humans to measure forest plots. Drones flying inside forests could collect this information autonomously, enabling fully digital forest information capture. In tropical plantations detecting ants killing the trees is one example of potential applications. Another interesting application area would be wild-food, where yields of berries and mushrooms could be found by flying inside the forest. In wet forests, the knowledge about areas covered by water is limited and difficult to observe from the top of the canopy. The low-cost drones flying inside forests could offer efficient tools also for that research area. In scientific research, observations collected by field inspections form the basis for understanding forest ecosystems. Inside flying drones will enable larger study areas and higher temporal resolutions than the classical methods, which is expected to improve forest environment understanding. Processing and analyzing under-canopy datasets also sets new challenges e.g. for remote sensing and machine learning researchers.

Once the hardware for inside forest flying has been implemented, the system will enable new research topics at Finnish Geospatial Research Institute. Different improvements will be implemented in the current system. Potentially new higher-performance drone hardware will be taken into use to enable longer flight times. In summer 2023, the performance of the system will be evaluated in different applications, such as bark beetle research and wild food.

8 Conclusion

In this thesis, a prototype of an autonomous drone for forest research purposes was implemented, and its performance and suitability to boreal forest environments were evaluated.

Based on the literature survey, EGO-Planner-v2 with VINS-Fusion VIO and depth camera-based mapping was proposed as the base of the implemented prototype. The performance of the proposed system was evaluated both in Gazebo simulation and in real forests with custom drone hardware.

In the simulated experiments, the two main components of the system, obstacle avoidance and VIO, were evaluated independently. Based on the results, increasing the forest density or flying velocity decreased the success rate of obstacle avoidance. With a maximum flying velocity of 1.0 m/s, the system was able to navigate without collisions in forests with a density of 0.15 trees/m². The average position error at the end of the flight was 0.4893 m in the 25 m flights and 1.3189 m in the 90 m flights.

The experiments with the drone hardware were performed in three different forest environments: a sparse mixed forest, a park woodland with dense understory vegetation, and a dense spruce forest. In a sparse forest, the system performed well, but in the park woodland and the dense spruce forest, the system had difficulties in sensing the small and dry branches which caused emergency stops, collisions, and failures. The performance of the VIO was satisfactory, and the error of the estimate of the trajectory length was approximately 1 m in the longest, approximately 80 m long, flight.

The performance of the system was promising, but more development is needed to increase the reliability in dense boreal forests. Especially the sensing of small needleless branches has to be improved either by increasing the camera resolution or by complementary sensors. Generally, detecting thin objects with stereo-vision is challenging, and that requires the cameras to have high resolution.

The implemented prototype can be used for example for further research on improving the performance of the system and developing applications on top of it. Potential ways to further improve the system are, for example, combining a high-level path planner performing area coverage to the system or improving the VIO estimate with loop closures and GNSS-fusion. Potential future applications of the system in forest research include image data collection of understory vegetation biodiversity research and detection of damages in the bark of the trees due to insect pests, to mention but a few.

References

- [1] M. Merz, D. Pedro, V. Skliros, C. Bergenhem, M. Himanka, T. Houge, J. P. Matos-Carvalho, H. Lundkvist, B. Cürüklü, R. Hamrén, A. E. Ameri, C. Ahlberg, and G. Johansen, “Autonomous UAS-Based Agriculture Applications: General Overview and Relevant European Case Studies,” *Drones*, vol. 6, no. 5, 2022.
- [2] A. Konert and T. Balcerzak, “Military autonomous drones (UAVs) - from fantasy to reality. Legal and Ethical implications.,” *Transportation Research Procedia*, vol. 59, pp. 292–299, 2021. 10th International Conference on Air Transport – INAIR 2021, TOWARDS AVIATION REVIVAL.
- [3] J. Scherer, S. Yahyanejad, S. Hayat, E. Yanmaz, T. Andre, A. Khan, V. Vukadinovic, C. Bettstetter, H. Hellwagner, and B. Rinner, “An autonomous multi-UAV system for search and rescue,” in *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pp. 33–38, 2015.
- [4] K. N. Tahar, A. Ahmad, W. A. A. W. M. Akib, and W. M. N. W. Mohd, “Aerial mapping using autonomous fixed-wing unmanned aerial vehicle,” in *2012 IEEE 8th International Colloquium on Signal Processing and its Applications*, pp. 164–168, IEEE, 2012.
- [5] J. Grzybowski, K. Latos, and R. Czyba, “Low-Cost Autonomous UAV-Based Solutions to Package Delivery Logistics,” in *Advanced, Contemporary Control* (A. Bartoszewicz, J. Kabziński, and J. Kacprzyk, eds.), (Cham), pp. 500–507, Springer International Publishing, 2020.
- [6] “Autonomous UAS Delivers Blood & Medical Supplies.” <https://www.unmannedsystemstechnology.com/2021/11/autonomous-uas-delivers-blood-medical-supplies/>. Accessed: 2022-06-23.
- [7] E. Hyypä, X. Yu, H. Kaartinen, T. Hakala, A. Kukko, M. Vastaranta, and J. Hyypä, “Comparison of Backpack, Handheld, Under-Canopy UAV, and Above-Canopy UAV Laser Scanning for Field Reference Data Collection in Boreal Forests,” *Remote Sensing*, vol. 12, no. 20, 2020.
- [8] J. Hyypä, X. Yu, T. Hakala, H. Kaartinen, A. Kukko, H. Hyyti, J. Muhojoki, and E. Hyypä, “Under-Canopy UAV Laser Scanning Providing Canopy Height and Stem Volume Accurately,” *Forests*, vol. 12, no. 7, 2021.
- [9] J. Choi and H. Myung, “BRM Localization: UAV Localization in GNSS-Denied Environments Based on Matching of Numerical Map and UAV Images,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4537–4544, 2020.

- [10] T. Feng, S. Chen, Z. Feng, C. Shen, and Y. Tian, “Effects of Canopy and Multi-Epoch Observations on Single-Point Positioning Errors of a GNSS in Coniferous and Broadleaved Forests,” *Remote Sensing*, vol. 13, no. 12, 2021.
- [11] R. Opromolla, G. Fasano, G. Rufino, M. Grassi, and A. Savvaris, “LIDAR-inertial integration for UAV localization and mapping in complex environments,” in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 649–656, 2016.
- [12] M. Karimi, M. Oelsch, O. Stengel, E. Babaian, and E. Steinbach, “LoLa-SLAM: Low-Latency LiDAR SLAM Using Continuous Scan Slicing,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2248–2255, 2021.
- [13] M. Hammer, B. Borgmann, M. Hebel, and M. Arens, “LiDAR-based localization and automatic control of UAVs for mobile remote reconnaissance,” in *Electro-Optical and Infrared Systems: Technology and Applications XVIII and Electro-Optical Remote Sensing XV* (D. L. Hickman, H. Bürsing, G. W. Kamerman, and O. Steinvall, eds.), vol. 11866, pp. 186 – 194, International Society for Optics and Photonics, SPIE, 2021.
- [14] D. Nister, O. Naroditsky, and J. Bergen, “Visual odometry,” in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I–I, 2004.
- [15] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, “Robust visual inertial odometry using a direct EKF-based approach,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 298–304, 2015.
- [16] M. Servières, V. Renaudin, A. Dupuis, and N. Antigny, “Visual and Visual-Inertial SLAM: State of the Art, Classification, and Experimental Benchmarking,” *Journal of Sensors*, 2021.
- [17] P. Florence, J. Carter, and R. Tedrake, *Integrated Perception and Control at High Speed: Evaluating Collision Avoidance Maneuvers Without Maps*, pp. 304–319. Cham: Springer International Publishing, 2020.
- [18] S. Giancola, M. Valenti, and R. Sala, *A survey on 3D cameras: Metrological comparison of time-of-flight, structured-light and active stereoscopy technologies*. Springer, 2018.
- [19] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [20] Y. W. Kuan, N. O. Ee, and L. S. Wei, “Comparative study of intel r200, kinect v2, and primesense rgb-d sensors performance outdoors,” *IEEE Sensors Journal*, vol. 19, no. 19, pp. 8741–8750, 2019.

- [21] D. Scaramuzza and F. Fraundorfer, “Visual odometry [tutorial],” *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.
- [22] F. Fraundorfer and D. Scaramuzza, “Visual odometry : Part ii: Matching, robustness, optimization, and applications,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012.
- [23] C. G. Harris and J. Pike, “3d positional integration from image sequences,” *Image and Vision Computing*, vol. 6, no. 2, pp. 87–90, 1988.
- [24] J. Shi *et al.*, “Good features to track,” in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pp. 593–600, IEEE, 1994.
- [25] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pp. 430–443, Springer, 2006.
- [26] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, pp. 91–110, 2004.
- [27] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 404–417, Springer Berlin Heidelberg, 2006.
- [28] D. Nister, “An efficient solution to the five-point relative pose problem,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004.
- [29] H. C. Longuet-Higgins, “A computer algorithm for reconstructing a scene from two projections,” *Nature*, vol. 293, no. 5828, pp. 133–135, 1981.
- [30] D. Scaramuzza and Z. Zhang, *Aerial Robots, Visual-Inertial Odometry of*, pp. 1–9. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020.
- [31] P. Corke, J. Lobo, and J. Dias, “An introduction to inertial and visual sensing,” 2007.
- [32] H. Moravec and A. Elfes, “High resolution maps from wide angle sonar,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 116–121, 1985.
- [33] S. Thrun, “Probabilistic robotics,” *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.
- [34] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: An efficient probabilistic 3D mapping framework based on octrees,” *Autonomous Robots*, 2013. Software available at <https://octomap.github.io>.

- [35] I. Kostavelis and A. Gasteratos, “Semantic mapping for mobile robotics tasks: A survey,” *Robotics and Autonomous Systems*, vol. 66, pp. 86–103, 2015.
- [36] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge University Press, 2017.
- [37] S. M. LaValle *et al.*, “Rapidly-exploring random trees: A new tool for path planning,” 1998.
- [38] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, “Learning by cheating,” in *Conference on Robot Learning*, pp. 66–75, PMLR, 2020.
- [39] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, *et al.*, “An algorithmic perspective on imitation learning,” *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.
- [40] P. Cunningham, M. Cord, and S. J. Delany, “Supervised learning,” in *Machine learning techniques for multimedia*, pp. 21–49, Springer, 2008.
- [41] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [42] 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,” *Journal of Field Robotics*, vol. 38, no. 2, pp. 307–326, 2021.
- [43] J. Kuffner and S. LaValle, “RRT-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, pp. 995–1001 vol.2, 2000.
- [44] 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)*, pp. 344–351, 2018.
- [45] V. Usenko, L. von Stumberg, A. Pangercic, and D. Cremers, “Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 215–222, 2017.
- [46] “Depth Camera D435.” <https://www.intelrealsense.com/depth-camera-d435/>. Accessed: 2023-2-8.
- [47] “Intel® RealSense™ Tracking Camera T265.” <https://www.intelrealsense.com/tracking-camera-t265/>. Accessed: 2023-2-8.
- [48] “Intel Aero Drone.” <https://botland.store/withdrawn-products/9808-intel-aero-drone-5032037106252.html>. Accessed: 2023-3-24.

- [49] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, “Learning high-speed flight in the wild,” *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021.
- [50] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, “PAMPC: Perception-Aware Model Predictive Control for Quadrotors,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8, 2018.
- [51] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, “Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.
- [52] “Chameleon 6.” <https://armattanquads.com/Chameleon-6-inch/>. Accessed: 2023-3-24.
- [53] “Skydio R1.” <https://support.skydio.com/hc/en-us/categories/115000372033-Skydio-R1>. Accessed: 2023-3-2.
- [54] X. Liu, G. V. Nardari, F. C. Ojeda, Y. Tao, A. Zhou, T. Donnelly, C. Qu, S. W. Chen, R. A. F. Romero, C. J. Taylor, and V. Kumar, “Large-Scale Autonomous Flight With Real-Time Semantic SLAM Under Dense Forest Canopy,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5512–5519, 2022.
- [55] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makeneni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, D. Thakur, K. Karydis, N. Atanasov, G. Loianno, D. Scaramuzza, K. Daniilidis, C. J. Taylor, and V. Kumar, “Fast, autonomous flight in GPS-denied and cluttered environments,” *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.
- [56] K. Mohta, K. Sun, S. Liu, M. Watterson, B. Pfrommer, J. Svacha, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, “Experiments in fast, autonomous, gps-denied quadrotor flight,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7832–7839, 2018.
- [57] S. W. Chen, G. V. Nardari, E. S. Lee, C. Qu, X. Liu, R. A. F. Romero, and V. Kumar, “Sloam: Semantic lidar odometry and mapping for forest inventory,” in *IEEE Robotics and Automation Letters (RA-L)*, 2020.
- [58] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, “RangeNet ++: Fast and Accurate LiDAR Semantic Segmentation,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4213–4220, 2019.
- [59] 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.

- [60] D. Harabor and A. Grastien, “Online graph pruning for pathfinding on grid maps,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, pp. 1114–1119, 2011.
- [61] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, “Search-based motion planning for aggressive flight in $se(3)$,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.
- [62] T. Lee, M. Leok, and N. H. McClamroch, “Geometric tracking control of a quadrotor UAV on $SE(3)$,” in *49th IEEE Conference on Decision and Control (CDC)*, pp. 5420–5425, 2010.
- [63] X. Zhou, X. Wen, Z. Wang, Y. Gao, H. Li, Q. Wang, T. Yang, H. Lu, Y. Cao, C. Xu, *et al.*, “Swarm of micro flying robots in the wild,” *Science Robotics*, vol. 7, no. 66, p. eabm5954, 2022.
- [64] Z. Wang, X. Zhou, C. Xu, and F. Gao, “Geometrically constrained trajectory optimization for multicopters,” *IEEE Transactions on Robotics*, pp. 1–10, 2022.
- [65] 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.
- [66] T. Qin, J. Pan, S. Cao, and S. Shen, “A general optimization-based framework for local odometry estimation with multiple sensors,” 2019.
- [67] J. Tordesillas and J. P. How, “Mader: Trajectory planner in multiagent and dynamic environments,” *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 463–476, 2022.
- [68] X. Zhou, J. Zhu, H. Zhou, C. Xu, and F. Gao, “EGO-Swarm: A Fully Autonomous and Decentralized Quadrotor Swarm System in Cluttered Environments,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4101–4107, 2021.
- [69] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, “EGO-Planner: An ESDF-Free Gradient-Based Local Planner for Quadrotors,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2021.
- [70] “Intel® RealSense™ Depth Module D430.” <https://www.intel.com/content/www/us/en/products/sku/98320/intel-realsense-depth-module-d430/specifications.html>. Accessed: 2023-2-8.
- [71] “Hardware Requirements.” https://github.com/KumarRobotics/kr_autonomous_flight/wiki/Hardware-Requirements. Accessed: 2022-07-13.
- [72] “ROS/Introduction.” <http://wiki.ros.org/ROS/Introduction>. Accessed: 2022-12-7.

- [73] “ROS/Concepts.” <http://wiki.ros.org/ROS/Concepts>. Accessed: 2022-12-7.
- [74] “Bags.” <http://wiki.ros.org/Bags>. Accessed: 2022-12-7.
- [75] “mavros.” <http://wiki.ros.org/mavros>. Accessed: 2023-2-28.
- [76] “MAVLink Developer Guide.” <https://mavlink.io/en/>. Accessed: 2023-2-28.
- [77] “Software Overview.” <https://px4.io/software/software-overview/>. Accessed: 2023-2-28.
- [78] “PX4 Architectural Overview.” <https://docs.px4.io/main/en/concept/architecture.html>. Accessed: 2023-2-28.
- [79] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *IJCAI’81: 7th international joint conference on Artificial intelligence*, vol. 2, pp. 674–679, 1981.
- [80] S. Agarwal, K. Mierle, and Others, “Ceres Solver.” <https://github.com/ceres-solver/ceres-solver>.
- [81] T. Qin and S. Shen, “Online temporal calibration for monocular visual-inertial systems,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3662–3669, 2018.
- [82] V. Usenko, L. von Stumberg, A. Pangercic, and D. Cremers, “Real-time trajectory replanning for mavs using uniform b-splines and a 3d circular buffer,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 215–222, 2017.
- [83] 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, 2011.
- [84] J. M. Mendel, *Lessons in Estimation Theory for Signal Processing, Communications, and Control, Second Edition*. Pearson, 2nd edition ed., 1995.
- [85] “Controller Diagrams.” https://docs.px4.io/main/en/flight_stack/controller_diagrams.html. Accessed: 2023-3-1.
- [86] “Multicopter PID Tuning Guide (Manual/Advanced).” https://docs.px4.io/main/en/config_mc/pid_tuning_guide_multicopter.html. Accessed: 2023-3-1.
- [87] “Simulation.” <https://docs.px4.io/main/en/simulation/>. Accessed: 2022-11-29.

- [88] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149–2154 vol.3, 2004.
- [89] “Gazebo Architecture.” https://classic.gazebosim.org/tutorials?tut=architecture&cat=get_started/. Accessed: 2022-11-29.
- [90] “Physics Parameters.” https://classic.gazebosim.org/tutorials?tut=physics_params&cat=physics/. Accessed: 2022-11-29.
- [91] “Using the ECL EKF.” https://docs.px4.io/main/en/advanced_config/tuning_the_ecl_ekf.html. Accessed: 2023-1-12.
- [92] X. Liang, Y. Wang, J. Pyörälä, M. Lehtomäki, X. Yu, H. Kaartinen, A. Kukko, E. Honkavaara, A. E. Issaoui, O. Nevalainen, M. Vaaaja, J.-P. Virtanen, M. Katoh, and S. Deng, “Forest in situ observations using unmanned aerial vehicle as an alternative of terrestrial measurements,” *Forest ecosystems*, vol. 6, no. 1, pp. 1–16, 2019.
- [93] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, “Continuous-time trajectory optimization for online uav replanning,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5332–5339, 2016.
- [94] Blender Online Community, “Blender - a 3D modelling and rendering package,” 2018.
- [95] “Groud Truth Position Pose and Rates Interface.” http://docs.ros.org/en/electric/api/gazebo_plugins/html/group__GazeboRosP3D.html. Accessed: 2023-1-25.
- [96] “ZENMUSE P1.” <https://www.dji.com/fi/zenmuse-p1>. Accessed: 2023-2-17.
- [97] “Agisoft Metashape User Manual Professional Edition, Version 1.7.” https://www.agisoft.com/pdf/metashape-pro_1_7_en.pdf. Accessed: 2023-2-17.
- [98] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, 2012.
- [99] “Mini PC PN51 .” <https://www.asus.com/fi/displays-desktops/mini-pcs/pn-series/mini-pc-pn51/>. Accessed: 2023-2-8.
- [100] “Hex Cube Black Flight Controller.” https://docs.px4.io/main/en/flight_controller/pixhawk-2.html. Accessed: 2023-2-8.

- [101] J. Maye, P. Furgale, and R. Siegwart, “Self-supervised calibration for robotic systems,” in *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 473–480, 2013.
- [102] 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, 2013.
- [103] P. Furgale, T. D. Barfoot, and G. Sibley, “Continuous-time batch estimation using temporal basis functions,” in *2012 IEEE International Conference on Robotics and Automation*, pp. 2088–2095, 2012.
- [104] E. Olson, “Apriltag: A robust and flexible visual fiducial system,” in *2011 IEEE International Conference on Robotics and Automation*, pp. 3400–3407, 2011.
- [105] “QGroundControl.” <http://qgroundcontrol.com/>. Accessed: 2023-3-13.
- [106] “IMU Noise Model.” <https://github.com/ethz-asl/kalibr/wiki/IMU-Noise-Model>. Accessed: 2023-2-9.
- [107] “ZEB HORIZON.” <https://geoslam.com/solutions/zeb-horizon/>. Accessed: 2023-3-14.
- [108] “GeoSLAM Connect.” <https://geoslam.com/solutions/connect/>. Accessed: 2023-3-14.
- [109] “Recording and playing back data.” <http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data>. Accessed: 2023-3-7.
- [110] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “Airsim: High-fidelity visual and physical simulation for autonomous vehicles,” in *Field and Service Robotics* (M. Hutter and R. Siegwart, eds.), (Cham), pp. 621–635, Springer International Publishing, 2018.
- [111] T. Lee, S. McKeever, and J. Courtney, “Flying free: A research overview of deep learning in drone navigation autonomy,” *Drones*, vol. 5, no. 2, 2021.
- [112] T. Qin, S. Cao, J. Pan, and S. Shen, “A general optimization-based framework for global pose estimation with multiple sensors,” 2019.

A Wind Speeds during the flight tests

The wind speeds were collected from the historical data of Finnish Meteorological Institute²⁸. The wind speeds presented in the following tables are from the observation center that was nearest to the test location. The wind speed during the flight tests presented in the chapters 6.2 and 6.3 are presented in Tables A1 and A2 respectively. These tests were performed in Espoo Otaniemi, so the nearest observation center was Espoo Tapiola.

Table A1: Windspeeds during the flight tests in sparse forest presented in the chapter 6.2.

Time	Gust speed	Wind speed
2023-1-27 13:00	3.2	2.2
2023-1-27 13:10	2.3	1.1
2023-1-27 13:20	1.7	1.0
2023-1-27 13:30	2.0	1.1
2023-1-27 13:40	2.1	1.2
2023-1-27 13:50	3.2	1.3
2023-1-27 14:00	3.6	2.4
2023-1-27 14:10	3.7	2.0

Table A2: Windspeeds during the flight tests in park woodland presented in the chapter 6.3.

Time	Gust speed	Wind speed
2023-1-31 10:30	8.5	5.7
2023-1-31 10:40	8.0	5.4
2023-1-31 10:50	10.7	5.7
2023-1-31 11:00	11.3	6.4
2023-1-31 11:10	10.7	7.1
2023-1-31 11:20	9.6	6.6
2023-1-31 11:30	10.7	6.7
2023-1-31 11:40	10.6	6.4
2023-1-31 11:50	10.8	6.4
2023-1-31 12:00	8.2	5.2
2023-1-31 12:10	7.9	5.3
2023-1-31 12:20	7.8	4.8

²⁸<https://en.ilmatieteenlaitos.fi/download-observations>

The flight tests presented in the chapter 6.4 were performed during two different test days, and the wind speeds are presented in the tables A3 and A4. The test location was Helsinki Paloheinä, and the nearest observation station was Helsinki Kumpula.

Table A3: Windspeeds during the first day of the flight tests in snowy spruce forest presented in the chapter 6.4.

Time	Gust speed	Wind speed
2023-2-2 13:00	5.7	2.8
2023-2-2 13:10	6.7	3.9
2023-2-2 13:20	6.2	3.7
2023-2-2 13:30	6.1	3.8
2023-2-2 13:40	5.8	3.7
2023-2-2 13:50	5.3	3.0
2023-2-2 14:00	5.8	3.4
2023-2-2 14:10	6.4	3.7
2023-2-2 14:20	4.8	2.9

Table A4: Windspeeds during the second day of the flight tests in snowy spruce forest presented in the chapter 6.4.

Time	Gust speed	Wind speed
2023-2-8 11:10	6.5	3.7
2023-2-8 11:10	6.2	3.7
2023-2-8 11:10	6.5	4.3
2023-2-8 11:10	6.5	4.4
2023-2-8 11:10	6.4	3.9
2023-2-8 11:10	6.5	4.3
2023-2-8 11:10	7.7	4.8