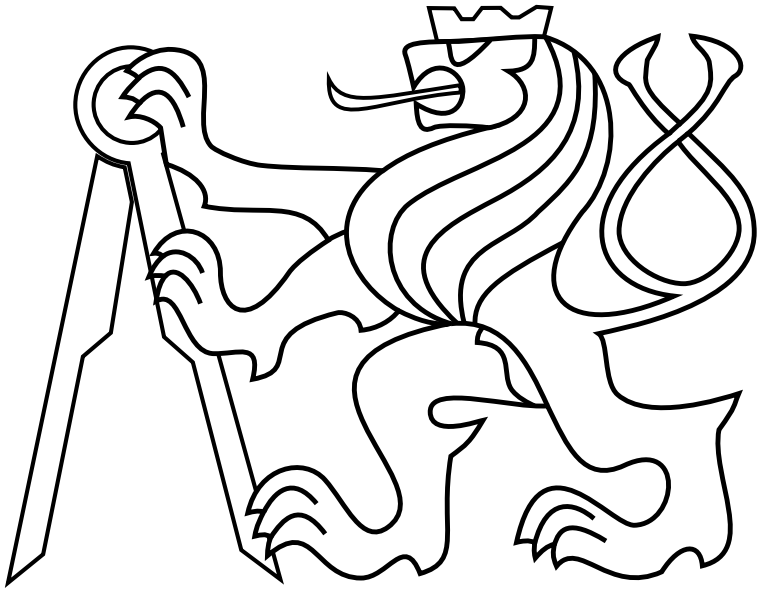
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Václav Pritzl

UAV Guidance for Fire Challenge of MBZIRC Contest

Department of Control Engineering

Thesis supervisor: **RNDr. Petr Štěpán, Ph.D.**

Declaration of Authorship

I hereby declare that I wrote the presented thesis on my own and that I cited all the used information sources in compliance with the Methodical instructions about the ethical principles for writing an academic thesis.

Prague, date..... .....

signature

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Pritzl Václav** Personal ID number: **456894**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

UAV guidance for fire challenge of MBZIRC contest

Master's thesis title in Czech:

Navigace UAV pro úlohu hašení v soutěži MBZIRC

Guidelines:

The goal of the thesis is to design, implement and experimentally verify algorithms for window detection and trajectory planning for flying through a window and flying inside a building as subtasks of the fire challenge of the MBZIRC competition. The following tasks will be solved:
1) To get familiar with sensors used for the fire challenge and with UAV control mechanisms.
2) To design and to implement algorithm for window detection from lidar fused with detections from Realsense depth camera (detection algorithm for Realsense is not developed by the student).
3) To design and to implement algorithm for planning trajectories for flying through window and flying inside the building to locate the fire.
4) To test and to analyze developed algorithms in a simulated environment.
5) To verify the solution in real experiments and to evaluate its performance under normal conditions as well as in a smoke environment.

Bibliography / sources:

[1] Wang, R., Bach, J., & Ferrie, F. P. (2011, January). Window detection from mobile LiDAR data. In 2011 IEEE Workshop on Applications of Computer Vision (WACV) (pp. 58-65). IEEE.
[2] Jutzi, B., Weinmann, M., & Meidow, J. (2014). Weighted data fusion for UAV-borne 3D mapping with camera and line laser scanner. International Journal of Image and Data Fusion, 5(3), 226-243.
[3] Tulldahl, H. M., & Larsson, H. (2014, October). Lidar on small UAV for 3D mapping. In Electro-Optical Remote Sensing, Photonic Technologies, and Applications VIII; and Military Applications in Hyperspectral Imaging and High Spatial Resolution Sensing II (Vol. 9250, p. 925009). International Society for Optics and Photonics.

Name and workplace of master's thesis supervisor:

RNDr. Petr Štěpán, Ph.D., Multi-robot Systems, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **31.01.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until:
by the end of summer semester 2020/2021

RNDr. Petr Štěpán, Ph.D. Supervisor's signature	prof. Ing. Michael Šebek, DrSc. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

I would like to thank my thesis advisor RNDr. Petr Štěpán, Ph.D. for his guidance during the work on my thesis. Furthermore, I would like to thank all the members of the Multi-robot Systems group of which there are too many to mention individually and I would surely forgot about somebody. This work would not have been possible without such an amazing team.

Next, I would like to thank my family for their endless support over the course of my life and my studies. Without their support, I would not have managed this.

Finally, I would like to thank all my friends who kept me from going crazy during my studies and my work on this thesis. Without them, I would never finish this. Especially, I wish to thank my friend Adam, because he feels that he deserves a special mention and after careful consideration I must say that he is right.

Abstract

This thesis deals with the guidance of an unmanned multirotor aircraft for autonomous flight into buildings and autonomous indoor exploration. This research was motivated by the firefighting challenge of the Mohamed Bin Zayed International Robotic Challenge 2020. The main focus of this work is placed on window detection from 2D LiDAR data, fusion of the obtained detections with 3D depth camera data, and path planning for safe flight through the detected window and inside the building. A combination of line extraction algorithms is used for window detection from the LiDAR data. A Kalman filter-based estimator is used for filtering and fusion of the individual detections. A state machine generating line trajectories with constant velocity is used for guiding the UAV through the detected window. A wall following algorithm producing local trajectories based on a single laser scan is used for guiding the UAV inside the building. The proposed algorithms were extensively verified in simulations and real-world experiments under normal visibility conditions and in a smoke-filled environment.

Keywords: unmanned aerial vehicle, window detection, firefighting, indoor-outdoor transition, indoor exploration, Kalman filtering, LiDAR, depth

Abstrakt

Tato práce se zabývá navigací bezpilotní helikoptéry pro účel autonomního vletu do budovy a jejího autonomního průzkumu. Tento výzkum byl motivován účastí na robotické soutěži Mohamed Bin Zayed International Robotic Challenge 2020, respektive její části zabývající se hašením požárů ve výškových budovách. Práce se soustředí zejména na detekci oken z dat naměřených pomocí 2D LiDARu, fúzi těchto detekcí s detekcemi z 3D hloubkové kamery a plánování bezpečného letu skrz detekované okno a vnitřek budovy. Okna jsou z LiDARových dat detekována pomocí kombinace algoritmů na extrakci přímek. Na filtrování a fúzi jednotlivých detekcí je použit lineární Kalmanův filtr. Stavový automat generující referenční trajektorie v podobě přímek s konstantní rychlostí je poté použit pro plánování průletu cílovým oknem. Algoritmus na sledování zdí, generující trajektorie na základě dat z LiDARových měření, je použit pro plánování letu uvnitř budovy. Všechny navržené algoritmy byly podrobně otestovány v simulacích a reálných experimentech za normálních podmínek i v prostředí s kouřem.

Klíčová slova: bezpilotní helikoptéry, detekce oken, hašení požárů, průlet do budovy, průzkum budov, Kalmanův filtr, LiDAR, hloubka

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

A fire breaking out inside buildings poses an ever-present threat for everybody, and therefore it is crucial to research new ways of detecting and extinguishing it as quickly as possible. The fire in high-rise buildings is especially dangerous because it can very quickly spread throughout the building and endanger the life of an enormous amount of people who can be deprived of their only way to escape.

Autonomous Unmanned Aerial Vehicles (UAVs) or drones have recently undergone rapid development, and an excellent possibility to employ them in early detection and extinguishing of fires has arisen. After a fire appears, the UAV can very quickly reach the fire's location even if it is located very high on the building or inside a hard-to-reach space and can start extinguishing it much earlier than it could be reached by human firefighters.

The possibility to employ UAVs in firefighting has already attracted the attention of many researchers. An obvious example of a situation where UAVs can prove beneficial is outdoor fire detection and monitoring. As reported in [1], a system of multiple UAVs could be used for automatic forest fire monitoring using visual and infrared cameras. Real experiments with forest fire monitoring in a national park have already been conducted by the Hungarian fire department[2]. The authors of [3] describe a task allocation strategy for distributed cooperation of ground and aerial robot teams in fire detection and extinguishing. In [4], a UAV system is designed for the delivery of fire-extinguishing bombs to a target area.

UAVs fighting fires have already been a topic of robotic competitions as well. Paper [5] describes the design and implementation of a firefighting UAV for outdoor applications designed specifically for the IMAV 2015 competition.

Moreover, the employment of UAVs could prove to be beneficial and life-saving in urban environments. Studies have already been done on fire detection in urban areas using a thermal camera carried by a UAV[6]. UAVs capable of entering buildings through doors and windows will be especially helpful because of their ability to reach the target location much earlier than human firefighters.

Work [7] contains the design of a semi-autonomous indoor firefighting UAV. The authors designed a fireproof, thermoelectrically cooled UAV equipped with visual and thermal cameras, collision avoidance module, and first-person view system. However, to fully exploit the potential of UAVs in firefighting and achieve reliable operation, the drones need to be autonomous. In the case of indoor firefighting, that means the UAV needs the capability of

autonomous detection of free entrances to the building, the capability of flying inside, exploring the building, detecting the fire, extinguishing the fire, and returning back to the starting location.

A similar very complex task was the topic of the fire challenge of the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) 2020.[1] The MBZIRC contest aims to demonstrate the current state of the art in robotics, to extend what is possible, and to bring the scientific theories closer to real-world applications. The contest provides an ambitious set of challenges motivated by contemporary world needs. The fire challenge was meant to utilize autonomous drones in extinguishing simulated fires outside and inside high-rise buildings.

This thesis focuses mainly on the following subtasks of the fire challenge:

- Detection of windows usable as building entrances
- Safe guidance of the UAV through the window into the building
- Indoor path planning in order to look for the simulated fire

1.1 State of the art

The guidance of autonomous UAVs for the task of flying through a window to enter a building has already been partially explored in literature several times using different approaches and with varying levels of experimental verification.

The specific case of a UAV flying through a window has been researched in [8] and [9]. The authors used RBG camera images and 2D LiDAR data for navigation, visual servoing while approaching the window, and potential field-based planning for the flythrough itself. They performed real experiments with flying through a wooden frame representing a window. Article [10] deals with the detection and estimation of a window that can be used for entering a building by a Micro Aerial Vehicle (MAV). The authors use a stereo camera rig for obtaining the data. Then, they use a haar-like cascade classifier for window recognition. They extract various features on the edges of the window and estimate the 3D center of the window from the left and right images. Frame to frame motion is then estimated for the navigation of the MAV. The authors evaluate the performance of the algorithm on prerecorded images of real windows. Work [11] deals with window detection from an RGB-D camera along with the generation of an optimal trajectory to a point in front of the window. The developed approach was verified in simulations. Paper [12] focuses on the control of an ornithopter MAV using visual servoing for narrow passage traversal. They demonstrate their approach in a real experiment with a small MAV flying through a narrow wooden frame. Article [13] describes visual servoing of an autonomous helicopter in urban areas, while using windows as features. The windows are detected from an RGB camera onboard the helicopter.

The research of window detection has also focused on detection from ground-based LiDAR or camera data for the purpose of urban planning, cadastral mapping, or navigation. Work [14] focuses on building facade separation and window detection from ground-based 3D LiDAR point clouds. The window detection is done by fitting planes to segments of point

[1] `http://mbzirc.com` (Accessed: 21 May 2020)

cloud data and searching for holes in the fitted planes. These holes are then filtered based on their location and their rectangular bounding boxes.

Similarly, paper [15] deals with facade and window detection from 3D LiDAR data collected from a moving vehicle in an urban environment. Their approach consists of ground point separation, fitting a plane to the facade using Random Sample Consensus (RANSAC), and detecting potential window edge points based on searching for neighboring points. The windows are then localized by projecting the potential window points in horizontal and vertical directions to two projection profiles, calculating histograms of the profiles, and localizing windows as peaks in the histograms.

In [16], window detection from RGB camera data is described. First, semantic segmentation is used to identify the facade of the building. An iterative algorithm then separates the data into blocks labeled as either facade or window based on vertical and horizontal gradients in the image.

Article [17] focuses on window detection and classification from visual data using a graph of contours. This approach consists of extracting edges from an image and representing sets of edges by an Attributed Relational Graph. Graph matching is used for window detection.

This thesis focuses more on the issue of detecting a single window, which will be used as a safe entrance to the building. This task is very similar to the task of door detection for ground robots. In [18], a door detection algorithm, which utilizes 3D depth data from the Microsoft Kinect sensor, is described. The algorithm searches for the largest hole (a place where data are missing) in the depth image. Then, it detects the door corners by iterating along the edges of the detected hole and calculates the width to length ratio, which is then used for confirming the detection and gaining information about the state of the door. Work [19] focuses on detection of closed door from 3D point cloud produced by PrimeSense depth sensor. It combines a region growing algorithm to detect the plane of the door and detection of the door handle as a hole in the plane. Article [20] approaches door detection by using a mobile robot equipped with a 2D LiDAR and a panoramic camera. An expectation-maximization algorithm is used for segmentation of the environment into door and wall objects based on their shape, color, and motion properties. In [21], a method for corridor and door detection from RGB video images is proposed. The doors are found by detecting their left, right, and top edges from the change of color in the image and verifying their required geometric properties. The approach proposed in [22] aims at the detection of doors from an RGB camera and 2D LiDAR mounted on a moving robot. The detection uses several weak classifiers that detect features such as door frame, width, color, and knob, and combines them using the AdaBoost algorithm. Paper [23] presents an image-based door detection algorithm based on the geometric model of doors combining edge and corner features obtained using the Canny edge detector. In [24], a door detection algorithm processing 2D LiDAR data from a moving ground robot is proposed. The algorithm detects the door by searching for space between 2 short segments and comparing it with a predefined door width. Article [25] proposes a door detection and tracking technique that uses a single RGB camera. It consists of extracting lines from the image, detecting the 3D wall planes in the image, and finding a rectangular shape with correct width to height ratio corresponding to the door. The door is then tracked using a 2D edge tracker applied to doorposts only.

Once the entrance into the building is detected, it is necessary to generate a safe trajectory through this entrance and execute it. Article [26] deals with state estimation, control, and

planning for an aggressive flight of a Micro Aerial Vehicle (MAV) through a narrow window tilted at various angles. The position of the window is known beforehand, and an aggressive trajectory through the window gap is generated. State estimation is done based on visual camera images and an Inertial Measurement Unit (IMU). Similarly, work [27] focuses on an aggressive flight of MAVs through narrow gaps tilted at various angles. The gap is detected from a forward-facing fisheye camera using a combination of Canny edge detection and search for quadrangular shapes. Then, a trajectory through the gaps is generated and executed.

Furthermore, flying into a building poses an interesting challenge because the transfer between indoor and outdoor environment leads to the possibility to combine different localization methods (e.g., Global Positioning System (GPS) outside the building and Simultaneous Localization and Mapping (SLAM) inside). Article [28] describes a system combining visual and laser odometry with IMU using an Extended Kalman Filter (EKF) for flight in both indoor and outdoor environments. Paper [29] describes an approach that used depth image processing for visual odometry capable of navigating MAVs during both indoor and outdoor flight and transfers from one to the other.

After entering the building, there are many different options for path planning inside the indoor space. To ensure that the space inside the building will be completely explored, it is beneficial to use a frontier exploration-based approach, e.g., [30] or [31]. However, this approach would require either having a map of the environment available or building a map of the environment online during flight. An example of algorithms usable for building such a map and planning on it can be found in [32]. While this approach ensures complete exploration of the target space, it is too computationally demanding, and depending on the nature of obstacles in the environment, a much simpler planning algorithm is sufficient. Another option, the one explored in this thesis, is to plan trajectories only based on current sensor data available to the UAV. An example of this approach is written in [33].

In order to reliably extinguish indoor fires, the guidance system must be able to work correctly under low visibility conditions in the presence of smoke. A comprehensive survey of the effects of smoke on sensor performance can be found in [34].

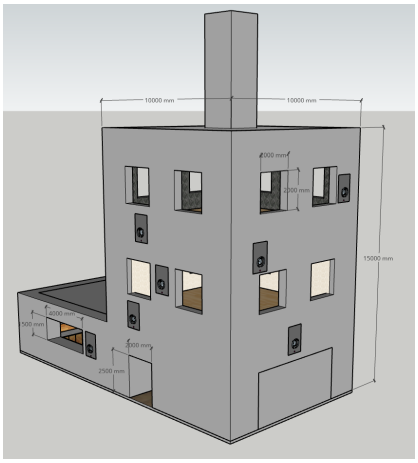
This thesis combines the subtasks of window detection, window position estimation, guidance for autonomous flight through the window, and for the autonomous flight along the walls inside the building. The proposed algorithms run completely online onboard the UAV hardware, and their performance has been extensively verified in both simulations and real-world deployments under conditions very close to a real firefighting scenario.

1.2 Problem Statement

The fire challenge of the MBZIRC competition consists of multiple tasks. These are extinguishing simulated fires inside a building, extinguishing outdoor fires on the facade of the building, and extinguishing outdoor fires on the ground next to the building.

The building, built specifically for the purpose of the fire challenge, contains three floors, with a single 2 by 2 meters large window opened on each floor with the rest of the windows closed. The model of the building provided by the organizers can be seen in Figure 1.1. The figure also contains a picture of the windows from the competition itself. This thesis focuses

(a) UAV searching for an open window



(b) Building model

(c) UAV extinguishing an outdoor fire on the facade of the building

Figure 1.1: UAV at the entrance to the building, building specifications provided by organizers and UAV extinguishing outside fires

on solving several subtasks necessary for the successful extinguishing of the inside fires. These subtasks include:

1. *Window detection and position estimation*: Detection of window edges from 2D LiDAR, fusing them with detections from RealSense 3D camera (the algorithm for detection from depth data is not a part of this thesis) and continuous estimation of the window's position during flying through it.

2. *Motion planning for flying through the detected window*: Planning and executing a safe maneuver for entering the building through the detected window.

3. *Path planning inside the building*: Continuous planning of a collision-free path inside the building. The rooms inside the building were assumed to contain obstacles on the ground and next to the walls of the room.

It was assumed that one room inside the building can contain smoke generated by the simulated fires.

1.3 Outline

This thesis is organized as follows. Chapter 1 describes the motivation behind the research performed in this thesis, an overview of related work, and a description of the problems to be solved.

Chapter 2 describes the system used to test the algorithms developed in this thesis. Section 2.1 contains information about the UAV platform and the specific sensors aimed to be used at the competition. Section 2.2 describes the software used for simulations and the interface with the hardware. The basic framework employed for controlling the drones is described.

Chapter 3 contains a description of the algorithms used for the detection of windows from the depth and LiDAR data, algorithms for fusion of these detections together, and for continuous estimation of the window position, orientation, and size for navigation while flying into the target building.

Chapter 4 contains path planning algorithms used for generating the desired UAV trajectory to be passed to the lower levels of the UAV control pipeline. Section 4.1 describes control algorithms for flying into the building through a window detected by the algorithms described in the previous chapter. Section 4.2 describes a path planning algorithm used for flying inside the target building in order to locate the fire. Section 4.3 describes an algorithm for filtering of data used for estimation of UAV altitude while flying through the window and inside the building.

Chapter 5 contains verification and tests of the algorithm performance in simulations before deployment to real UAV hardware.

Chapter 6 describes the performance of the proposed solutions in real-world experiments. The experimental evaluation includes tests at a small-scale testing building at the university, further tests at the desert, and information about the deployment of the solution at the competition itself. The performance of the solutions is evaluated both under normal conditions and decreased visibility conditions.

Finally, Chapter 7 contains a summary of the work performed in this thesis and the achieved results.

Chapter 2

Description of Used System

This chapter contains a short description of the hardware and software used for the UAVs. In the hardware description, special emphasis is put on the RPLIDAR laser scanner and RealSense depth camera, as they are the primary sources of data for algorithms used in this thesis.

2.1 Hardware Description

The UAV platform designed for the competition can be seen in Figure 2.1. It is a quadrotor based on Tarot 650 frame. The UAV dimensions are approximately $80 \times 80 \times 45$ cm including propellers. The weight of the UAV is approximately 3.6 kilograms without water.

Low-level control is performed by Pixhawk 4 Flight Controller, which includes a built-in IMU with accelerometers, a magnetometer, and a barometer. The flight controller and sensors are all connected to the Intel NUC Kit NUC8i7BEH computer, which runs all the UAV software.

Localization in outdoor environment is performed based on data from Pixhawk GPS and compass module. Two Garmin LIDAR-Lite v3 rangefinders, one of them pointing upwards and the other pointing downwards, are mounted on the UAV. They provide measurements of the distance from the ground below and the ceiling above the UAV. The RPLIDAR-A3 laser scanner is mounted on top of the UAV. The RealSense D435 depth camera is placed at the front of the UAV. Next to the RealSense camera, there are 3 TeraRanger Thermal 33 cameras for fire detection. Finally, a downward-looking BlueFox camera is mounted on the UAV, which enables localization using optic flow.

The fire extinguishing device consists of a water bag connected to a water pump controllable by the computer. The water is sprayed through a nozzle at the target location.

2.1.1 RPLIDAR-A3

The RPLIDAR-A3 laser scanner is a Light Detection and Ranging (LiDAR) device. It is capable of measuring distance to obstacles in a 2D plane in 360-degree field of view by illuminating the target with infrared light and measuring the reflected light by a sensor.

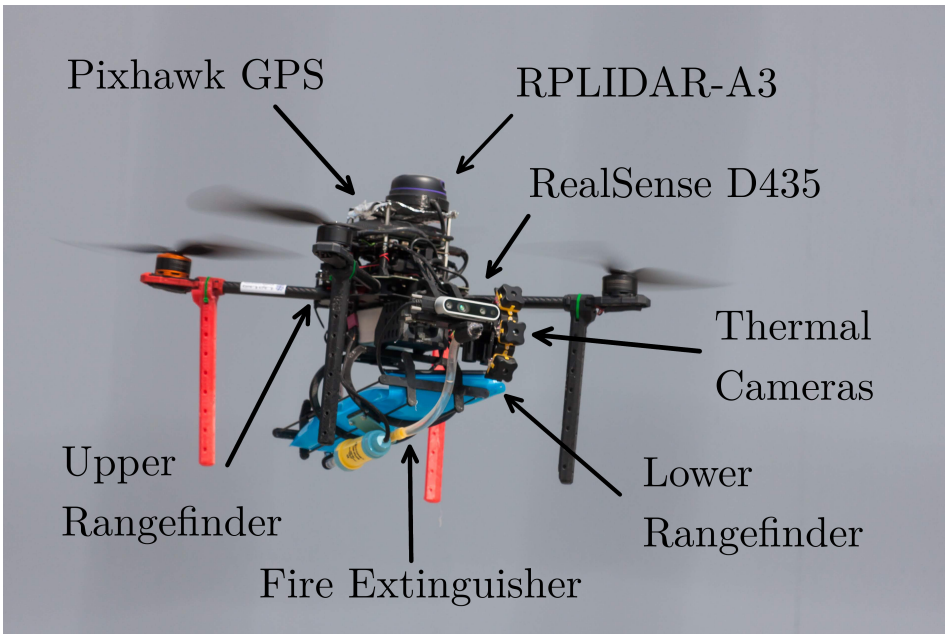


Figure 2.1: UAV platform used for the competition

Datasheet of the laser scanner can be found at [1] and some of its specifications can be found at [2]. Table 2.1 displays an extract of some of its basic parameters.

RPLIDAR offers the option to switch between different scan modes. The scan modes used in this thesis are called "sensitivity" and "stability"[3] and are dedicated for indoor and outdoor usage, respectively. These scan modes differ in the measurement frequency, maximal measurement distance, sensitivity of detection, and environment light elimination.

The *sensitivity* mode is optimized for longer range and better sensitivity but suffers from weak environment light elimination. As the experiment in Section 6.4 shows, the mode is able to function even in thick smoke and is capable of detecting black objects. However, it is unable to see objects illuminated by direct sunlight and therefore cannot be used outdoors.

The *stability* mode is optimized for environment light elimination performance at the cost of shorter range and lower sample rate. Furthermore, its performance drastically decreases in an environment containing smoke, and it is practically unable to detect black objects.

In this thesis, RPLIDAR data are utilized for indoor localization, window detection, and path planning while following walls inside the building. The RPLIDAR is used in its indoor mode while flying inside and in its outdoor mode while flying outside in direct sunlight.

[1] `http://bucket.download.slamtec.com/aaf96dddba2f6a9baa03261628c01af9fc2f866c/LD310_SLAMTEC_rplidar_datasheet_A3M1_v1.0_en.pdf` (Accessed: 18 May 2020)

[2] `https://www.slamtec.com/en/Lidar/A3Spec` (Accessed: 18 May 2020)

[3] `http://bucket.download.slamtec.com/ccb3c2fc1e66bb00bd4370e208b670217c8b55fa/LR001_SLAMTEC_rplidar_protocol_v2.1_en.pdf` (Accessed: 18 May 2020)

Parameter	Indoor Mode (Sensitivity)	Outdoor Mode (Stability)
Maximal Distance	White object: 25 m	White object: 20 m
	Dark object: 10 m	Dark object: ≈ 0 m
Sample Rate	16000 Hz	10000 Hz
Scan Rate	10-20 Hz (≈ 13 Hz used during experiments)	
Laser Wavelength	785 nm	
Angular Resolution	0.3375°	0.54°

Table 2.1: RPLIDAR-A3 specifications

Parameter	Depth	RGB
Maximal Depth Range	10 m	-
Maximal Depth Range	0.105 m	-
Field of View (H × V × D)	87° × 58° × 95°	69.4° × 42.5° × 77°
Maximal Resolution	1280 × 720 (at max 30 Hz)	1920 × 1080
Maximal Frame Rate	90 Hz	30 Hz

Table 2.2: Intel RealSense D435 specifications

2.1.2 RealSense D435

The Intel RealSense D435 is a stereo vision depth camera system. It consists of a pair of infrared (IR) cameras, an infrared projector, and an RGB camera. The IR projector optionally projects an invisible infrared pattern on the scene, improving the depth accuracy. The left and right IR cameras capture the scene, and the captured data are combined to obtain a depth image. The depth data can be passed on for further processing either in the form of a point cloud or in the form of a depth image where each pixel in the image has a value that corresponds to the depth of the object in the scene.

The RealSense depth camera is capable of working both in indoor and outdoor environments. However, as was found out during the real experiments, the quality of depth data is significantly influenced by smoke. Additionally, the accuracy of the data is decreased by direct sunlight. Furthermore, RealSense can act as a source of electromagnetic interference for other sensors, e.g., the GPS module. For usage in real-world applications, it needs to be properly shielded.

Basic specification of the camera can be found at [4] and its datasheet can be found at [5]. Table 2.2 contains its basic properties extracted from the official specifications. The field of view is specified as horizontal × vertical × diagonal values. In this thesis, RealSense depth camera provides data for one of the algorithms for window detection.

[4] `https://www.intelrealsense.com/depth-camera-d435/` (Accessed: 18 May 2020)

[5] `https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf` (Accessed: 18 May 2020)

2.2 Software Description

All the software runs on the Intel NUC Mini PC with the Ubuntu 18.04 operating system. Robot Operating System[6] (ROS) is used as a middleware for the development of the robotic software. It provides services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, and package management. ROS contains packages to intuitively handle transformations between different coordinate frames. It is possible to define specific transformations for each sensor or localization algorithm and transform coordinates from one frame of reference to another. Furthermore, it enables to use the same implementation of robotic algorithms for simulations and for reality, which is crucial for safe development of control software.

The Gazebo robotics simulator was used for simulations and testing of the software before deploying it to real UAV hardware. It is an open-source 3D robotics simulator with a robust physics engine enabling to simulate the real-time robot behavior in various complex environments.

2.2.1 Control pipeline

Figure 2.2 displays a diagram of the control pipeline running on the UAVs. It provides features such as precise trajectory tracking, localization, and collision avoidance and therefore enables the completion of various complex tasks by the UAVs. The control pipeline has been developed in the Multi-robot Systems group and is described in [35] and [36] in detail. A short description of the individual components follows.

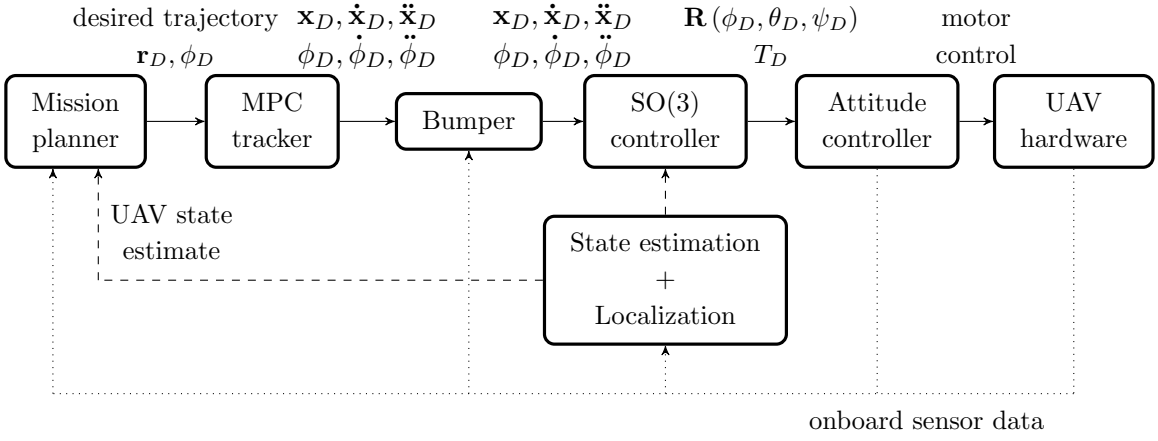


Figure 2.2: Diagram of the control pipeline

The *UAV hardware* represents the interface to the hardware part of the UAV, which provides sensor data and accepts motor commands.

The *Attitude controller* is the controller embedded in the Pixhawk 4 flight control unit. It accepts desired orientation of the UAV $\mathbf{R}$ and desired total thrust T_D and outputs commands to electronic speed controllers controlling the motors to achieve the desired attitude.

[6] `https://www.ros.org/` (Accessed: 18 May 2020)

The *SO(3) controller* is a nonlinear state feedback controller, described in [37]. Its input is a trajectory defined by particular positions $\mathbf{x}_D$, yaw angles ϕ_D, linear velocities $\dot{\mathbf{x}}_D$, angle velocities $\dot{\phi}_D$, linear accelerations $\ddot{\mathbf{x}}_D$, and angle accelerations $\ddot{\phi}_D$ and the current UAV state. Its output is the desired UAV attitude and total thrust, which is passed to the attitude controller.

The *Bumper* provides reactive obstacle avoidance. It accepts sensor data from the LiDAR, RealSense 3D camera, and rangefinders and the predicted UAV trajectory in order to detect potential collisions. All points included in the predicted trajectory are checked for possible collisions, and the trajectory is modified to avoid dangerous obstacles.

The *MPC tracker* represents a reference generator for the underlying state feedback. It consists of a linear Model Predictive Controller (MPC) and a linear time-invariant (LTI) model of the UAV system connected by a simulated control loop. The MPC tracker interface accepts a desired trajectory consisting of a list of set-points sampled with a constant time step Δt. Each set-point is defined by its Cartesian coordinates $\mathbf{r}_D = [x_D, y_D, z_D]^T$ and desired UAV yaw orientation ϕ_D. The coordinates can be specified in an arbitrary coordinate frame with an existing and known transformation to the coordinate frame currently used by the controller.

The MPC tracker then generates a smooth and feasible trajectory with a constant sampling step, including velocities and accelerations at each point, while considering the translational dynamics of the UAV and the input and state constraints. The trajectory generation assumes that the original set-points are sampled with a constant time step $\Delta t = 0.2$ s by default. Therefore, it is possible to change the velocity of the UAV while following the trajectory by changing the Euclidean distance between the set-points, considering the constant time step Δt. The generation of set-points for flying through the window and inside the rooms is described in Section 4.1.2.

The *Mission planner* provides navigation and higher-level planning of the UAV movement. It accepts the sensor data and current UAV state and calculates desired trajectories for the MPC tracker. The mission planner is a state machine which employs various navigation algorithms. The algorithms for flying through windows and flying inside the building described in this thesis represent parts of the mission planner.

2.2.2 State estimation and localization

The *State estimation and localization* manages estimation of the UAV state vector

$$\mathbf{x} = (\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}, \mathbf{R}, \dot{\mathbf{R}})^T, \tag{2.1}$$

where $\mathbf{r} = [x, y, z]^T$ denotes the Cartesian coordinates of the UAV position. $\dot{\mathbf{r}}$ and $\ddot{\mathbf{r}}$ are the linear velocity and acceleration vectors, respectively. Rotation matrix $\mathbf{R}(\phi, \theta, \psi)$ represents UAV orientation in the world coordinate frame and $\dot{\mathbf{R}}$ represents angular velocities, as described in [36].

The state vector is estimated using a linear Kalman filter (LKF), which fuses information from the UAV's inertial measurement unit and data from the currently selected localization method.

In the experiments performed in this thesis, localization in open outdoor environments is done using Global Navigation Satellite Systems (GNSS). The precision of this localization method depends on the number of satellites available, the presence of high obstacles in the area, electromagnetic interference generated both by various sources in the area and the UAV itself, atmospheric conditions, etc.

The localization in indoor environments is done based on LiDAR data passed to the Hector SLAM algorithm [38]. The development of Hector SLAM was motivated specifically by urban search and rescue (USAR) scenarios. The algorithm is designed to be used with 2D laser scanner data. It operates by constructing a 2D grid-based occupancy map and estimating the position of the robot in the map using a robust scan matching approach.

The localization in the vertical z-axis is performed by the fusion of barometer data from the Pixhawk unit and range data from the Garmin rangefinders. For the purpose of an indoor-outdoor transition, where the range data can change rapidly and can contain offset caused by flying to a higher floor, the range data are filtered, and current distance from the floor and the ceiling is combined to obtain a reliable height estimate as described in Section 4.3.

The state estimation framework also enables the option to employ other localization methods, such as optic flow or visual odometry.

Chapter 3

Window Detection and Position Estimation

This chapter describes the system that detects potential entrances into the target building and consequently estimates their positions and parameters, which are used for guidance of the UAV flying through the window into the building.

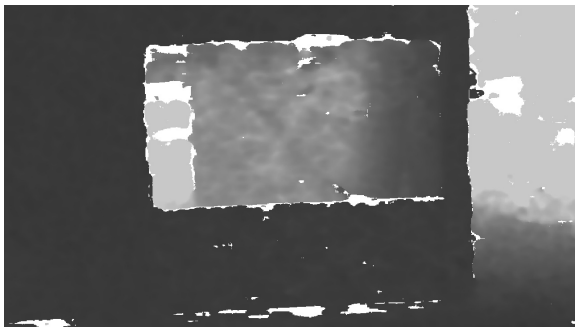
The system consists of a depth detector, LiDAR detector, and window position estimator. The depth detector, described in 3.1, receives raw depth data from the RealSense RGB-D camera and detects rectangular holes in the data. The LiDAR detector, described in 3.2, detects window candidates from 2D laser scanner data. The depth and LiDAR detections are passed to an estimator based on a linear Kalman filter, described in 3.3. The estimator runs at a fixed rate, fuses the individual detections, and filters out measurement noise.

3.1 Detection from RealSense Data

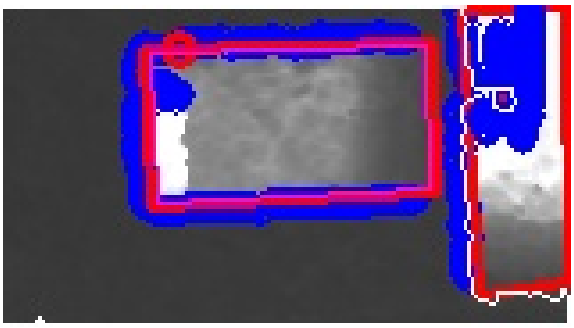
The algorithm for window detection from depth data was developed by Petr Štěpán from the Multi-robot Systems Group at CTU in Prague, and its development was not a subject of this thesis. Therefore, it is described only shortly, and the thesis focuses mainly on its usage and its performance in real-world experiments.

The input of the algorithm is a 1280×720 depth image produced by the RealSense 3D camera. An example of such an image can be seen in Figure 3.1a. The observed window is shown in Figure 3.1c. Each pixel in the depth image describes the measured depth along the camera z-axis in meters.

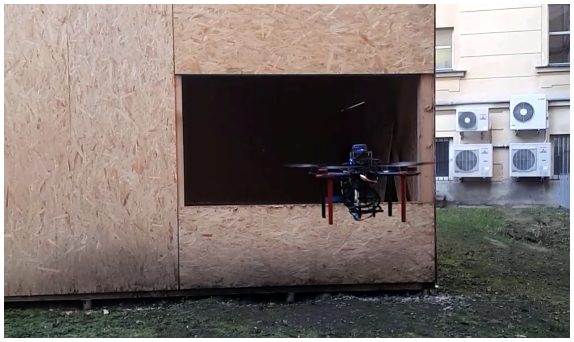
The detection algorithm first downsamples the image by the factor of 8 to reduce the computational demands on the CPU. Then, the algorithm searches for contours in the image and tries to fit rectangles with certain parameters to the data. After identifying such rectangles, a check is performed to ensure that the rectangular shape is a hole and not a protrusion. Furthermore, the algorithm is capable of detecting incomplete windows, where some of their edges are missing from the image (UAV flying close to the window can often see only a part of it). The information about the missing edges is published as a part of the detection data. An example of the detection can be seen in Figure 3.1b. There is a correct



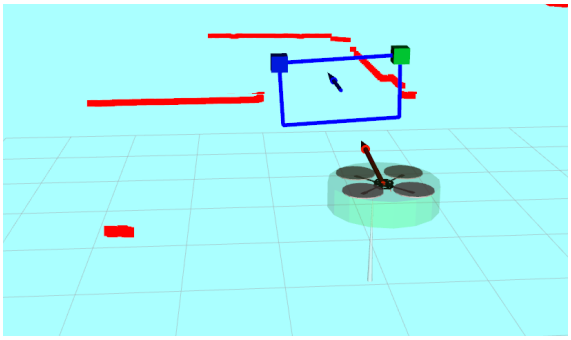
(a) Original depth image



(b) Detection in the depth image



(c) UAV in front of the window



(d) 3D visualization of the detection in Rviz

Figure 3.1: Example of window detection from RealSense depth data

detection in the center of the image and a false-positive detection of an incomplete window on the right side. The false-positive detection needs to be filtered out.

Finally, the position, orientation, and size of the detected window are calculated and published. 3D visualization of the filtered detection in Rviz can be seen in Figure 3.1d. The blue rectangle with an arrow in the middle represents the window detection and red squares show walls of the building, as seen on LiDAR.

The detection output is defined by the following parameters:

- Window center, defined by coordinates $C = [c_x, c_y, c_z]^T$ either in the world coordinate frame or in the local coordinate frame relative to the UAV position
- Window corners, each defined by their Cartesian coordinates $\gamma_{1...4} = [\gamma_{i,x}, \gamma_{i,y}, \gamma_{i,z}]^T$
- Normal vector $\mathbf{v} = [v_x, v_y, v_z]^T$ pointing away from the UAV, describing the orientation of the window plane
- Distance d_{win} of the window center from the UAV
- Width w and height h of the window
- Validity of edges - information whether the top, left, right, and bottom edge of the window was seen in the depth image

In the real-world experiments, the RealSense camera is set to produce depth images at the rate of 15 Hz, which produces a sufficient number of detections for navigation and does

not cause too high CPU load. The algorithm also offers the option to filter out false detections based on apriori known window size.

3.2 Detection from LiDAR Data

The RPLIDAR-A3 laser scanner provides range data in a 2D horizontal plane in 360 degrees around the UAV. When compared to the depth camera, it is not constrained by a limited field of view in the 2D plane and can see the window regardless of the current UAV orientation. Furthermore, it is less influenced by the distance from the window as the RPLIDAR-A3 can see objects up to 25 m away. The range measurements obtained from the LiDAR are very precise. However, as it is a 2D laser scanner, the data only contains the left and right edges of the window.

This section describes the algorithm capable of detecting left and right window edges from 2D laser scanner data. The algorithm can either accept an existing window estimate and search for corresponding edges, or identify window edge candidates based on apriori-known window width without the need of an existing estimate.

Although the algorithm was designed to be used with the RPLIDAR-A3 scanner, it can work with any sensor able to produce 2D laser scanner data. Each scan contains an array of ranges obtained by measuring the time of IR signals reflected from obstacles around the UAV. Each scan is processed and checked for window edge detections individually.

The algorithm works by first identifying points representing potential window edges in the data and then by either linking them to an existing window estimate or linking two edges together based on apriori-known information.

3.2.1 Detection of Potential Window Edges

The range data are published in the polar form, each point in the scan can be described by a pair of values $[d_i, \theta_i]$, where d_i is the distance from the center of the laser scanner to the detected obstacle and θ_i is the angle from the axis pointing towards the front of the UAV. When converting the observed points to Cartesian coordinates, the origin is located in the center of the LiDAR, the x axis points forward from the UAV, the y axis points to the left, and the z axis points upwards. The chosen polar and Cartesian coordinate systems can be seen in Figure 3.2.

The Cartesian coordinates $[x_i, y_i]$ of each point obtained from the scan can be calculated as

$$x_i = d_i \cos(\theta_i) \tag{3.1}$$

$$y_i = d_i \sin(\theta_i). \tag{3.2}$$

The RPLIDAR produces the range data in an array with start and end at the back of the UAV. Therefore, the measurement angles from the source array need to be shifted by π. If the sensor does not detect a sufficiently strong reflection of the transmitted IR signal, an empty range is returned. For further processing, the empty range is replaced with an infinite value in the ROS datatype used in the implementation.

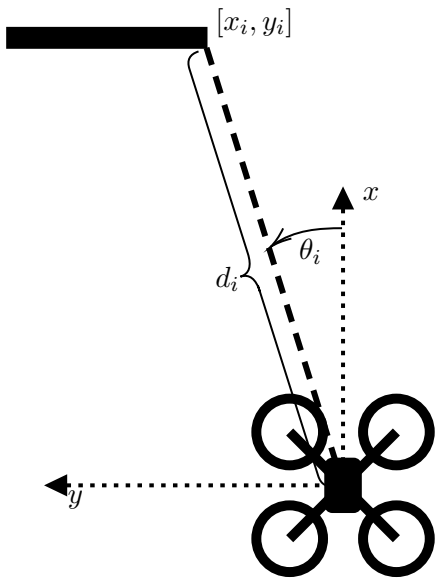


Figure 3.2: LiDAR coordinate system

This section describes the process of parsing the array of ranges into a list of features, which could potentially represent edges of windows.

The detection of potential window edges is performed in several steps based on the Successive Edge Following and Iterative End-Point Fit algorithms. These algorithms are simple and widely used line extraction algorithms that are ideal for parsing the LiDAR data. First, the array of ranges is parsed into a set of segments using a modified version of Successive Edge Following. The endpoints of these segments are added to the list of potential window edges. Then, a variation of the Iterative End-Point Fit algorithm is used to further divide the obtained segments into smaller subsegments, and their endpoints are again added to the list of potential window edges.

Modified Successive Edge Following Algorithm

The Successive Edge Following (SEF) is a line extraction algorithm, which works directly with the polar form of the laser scan and does not require any line fitting. It goes through the measured ranges iteratively and therefore does not need the complete laser scan to work. As such, it is very fast and has low computational requirements. Information about the algorithm can be found in [39] and [40]. Its primary purpose is the extraction of line segments from laser scanner data. Here, a modified version of SEF is used for both dividing the laser scan into segments and extraction of features, which could potentially be window edges.

The algorithm iteratively traverses the array of ranges in the laser scan. For each range d_i, it calculates differences from the previous range d_{i-1} and the following range d_{i+1}. The calculated differences are then compared with the value of a threshold d_{thr}. If the differences are lower than the value of the threshold, the points are assumed to be a part of the same

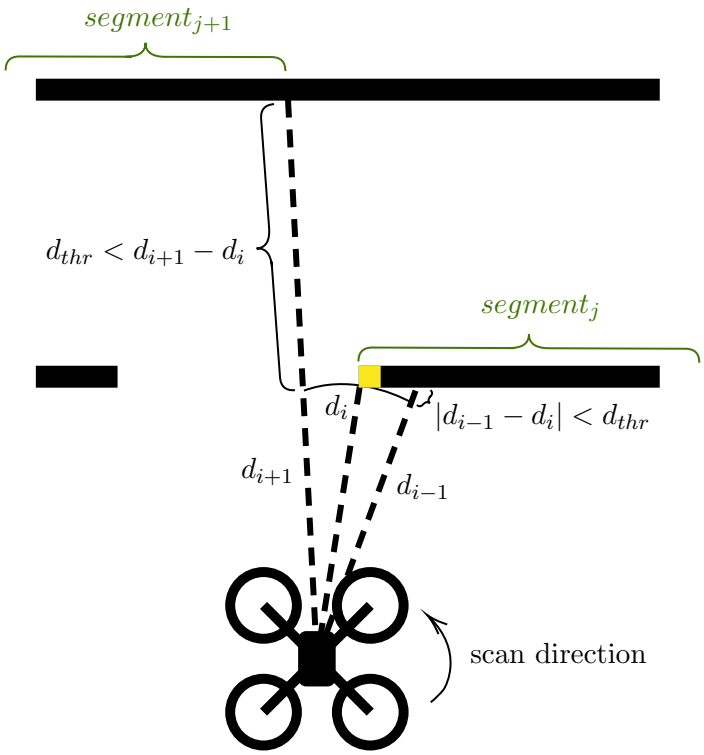


Figure 3.3: Modified Successive Edge Following - yellow color represents the detected window edge.

continuous segment. If the condition

$$(d_{i+1} - d_i) > d_{thr} \text{ and } |d_{i-1} - d_i| < d_{thr} \tag{3.3}$$

is satisfied (the increase in distance of the following range is larger than d_{thr} and the previous range is still in the same segment), a window starting edge is detected. This situation is depicted in Figure 3.3. If the condition

$$(d_{i-1} - d_i) > d_{thr} \text{ and } |d_{i+1} - d_i| < d_{thr} \tag{3.4}$$

is satisfied (the decrease of range from the previous range to the current one is larger than d_{thr} and both the current and following ranges are in the same segment), a window ending edge is detected. Every time an edge is detected, the segment of points between the detected edges is also saved for future processing.

Algorithm 1 describes these steps in detail. The difference of this version from the Successive Edge Following algorithm, described in [39], is that three consecutive points, instead of two, are used for the detection of line endpoints, and the algorithm can be used not only for line extraction but specifically for detection of the line endpoints.

Iterative End-Point Fit Segmentation and Corner Detection

After detecting edge candidates based on sudden changes in distance and separating the laser scan into segments using SEF, it is necessary to perform further processing of the

Algorithm 1 Modified Successive Edge Following

Input: Circular list $\mathcal{P} = \langle P_1, \dots P_n \rangle$, where $P_i = (d_i, \theta_i)$ are points obtained from a single laser scan; threshold d_{thr} representing maximal difference between two consecutive ranges to consider them the same segment.

Output: List $\mathcal{E} = \langle P_1, \dots P_m \rangle$ of window edge candidates; List $\mathcal{S} = \langle s_1, \dots s_o \rangle$ of line segments parsed from the scan.

function MSEF(P, d_{thr})
  $seg\_start,\ seg\_first\_end \leftarrow \varnothing$
  for all $P_i \in P$ **do**
    if $(d_{i+1} - d_i) > d_{thr}$ and $|d_{i-1} - d_i| < d_{thr}$ **then**
      Add P_i to $\mathcal{E}$ as window starting edge
      END_SEGMENT_AT($i,\ seg\_start,\ seg\_first\_end,\ \mathcal{S},\ \mathcal{P}$)
      if $d_{i+1} < \infty$ **then**
        $seg\_start \leftarrow i + 1$
      end if
    else if $(d_{i-1} - d_i) > d_{thr}$ and $|d_{i+1} - d_i| < d_{thr}$ **then**
      Add P_i to $\mathcal{E}$ as window ending edge
      if $d_{i-1} < \infty$ **then**
        END_SEGMENT_AT($i - 1,\ seg\_start,\ seg\_first\_end,\ \mathcal{S},\ \mathcal{P}$)
      end if
      $seg\_start \leftarrow i$
    end if
  end for
  if $seg\_start \neq \varnothing$ and $seg\_first\_end \neq \varnothing$ **then**
    Add line segment $\langle P_{seg\_start}, \dots P_{seg\_first\_end} \rangle$ to $\mathcal{S}$
  end if
  return $\mathcal{E}, \mathcal{S}$
end function
▷ End of one of the segments can precede its start as $\mathcal{P}$ is a circular list
procedure END_SEGMENT_AT($index,\ seg\_start,\ seg\_first\_end,\ \mathcal{S},\ \mathcal{P}$)
  if $seg\_start \neq \varnothing$ **then**
    Add line segment $\langle P_{seg\_start}, \dots P_{index} \rangle$ to $\mathcal{S}$
    $seg\_start \leftarrow \varnothing$
  else if $seg\_first\_end = \varnothing$ **then**
    $seg\_first\_end \leftarrow index$
  end if
end procedure

obtained segments. The window edge does not necessarily need to have the form of a sudden change in distance, but the change can be gradual, e.g., when the window is close to a wall perpendicular to the window. Therefore, it is necessary to detect features in the form of corners sticking out towards the UAV as window edge candidates. For that, Iterative End Point Fit (IEPF) based segmentation further divides the segments of points into smaller subsegments and detects these corner-like features.

The IEPF algorithm is a line extraction algorithm, working with a segment of laser

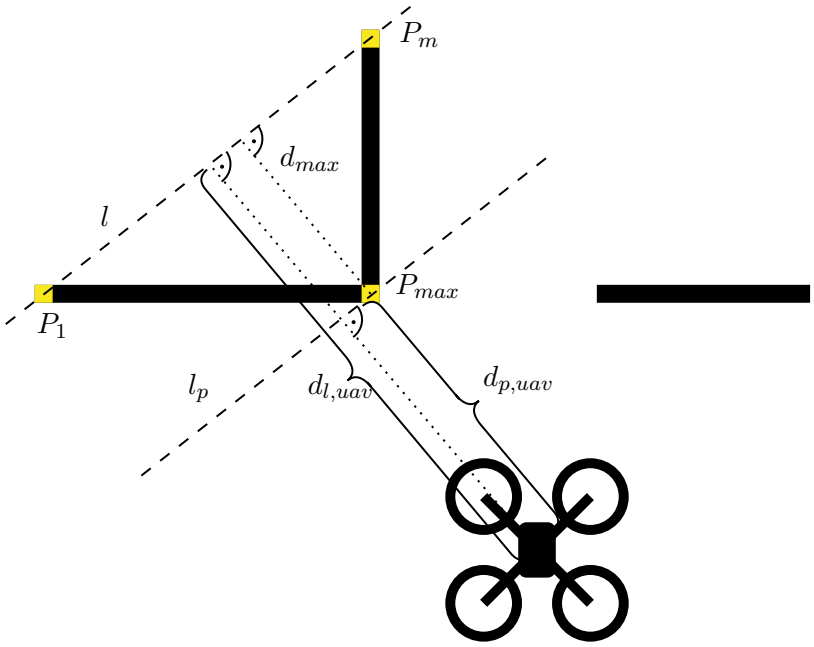


Figure 3.4: Iterative End Point Fit and corner detection - point p_{max} highlighted in yellow is detected as a window edge candidate.

scan with points defined in Cartesian coordinates. The algorithm iteratively goes through the segment and divides it into smaller and smaller segments by fitting a line through the endpoints of each segment. The algorithm finds the most distant point from the line in the segment and splits the segment at that point. The complete IEPF algorithm is described in [39] and [40]. The implementation can optionally include a merging step, which merges collinear line segments together. However, this step is not required for the purpose of this thesis.

The process of finding a point, where a segment can be split, and classifying it as a window edge candidate, is illustrated in Figure 3.4, and described in Algorithm 2. The algorithm accepts the segments of scan points produced by the previously described SEF algorithm. It iteratively traverses these segments and consequently their subsegments. For each segment $s_i = \langle P_1, \dots P_m \rangle$, it fits a line l to its endpoints P_1 and P_m. Then, point P_{max} with the largest distance d_{max} from the fitted line is found. If the distance d_{max} is larger than the value of a threshold d_{thr}, the segment is split into two subsegments $s_a = \langle P_1, \dots P_{max} \rangle$ and $s_b = \langle P_{max}, \dots P_m \rangle$. The original segment s_i is replaced with these two subsegments.

Furthermore, if the point P_{max} has the form of a corner sticking out towards the UAV, it is added to the list of window edge candidates. This is checked by defining a parallel line l_p to line l, which passes through the point P_{max}. Then, the distances between the UAV and the lines l and l_p are compared. If the line belonging to the point P_{max} is closer to the UAV than the original fitted line, the point P_{max} is added to the list of window edge candidates.

A proper value of the threshold d_{thr} has to be determined experimentally. For the configuration used in this thesis with laser scan data obtained from the RPLIDAR-A3, the value of d_{thr} was 0.1 m.

The IEPF algorithm again has low computational requirements, because it requires only

Algorithm 2 Iterative End-Point Fit and Corner Detection

Input: List $\mathcal{S} = \langle s_1, \dots s_n \rangle$, where $s_i = \langle P_1, \dots P_m \rangle$ are segments obtained from the modified Successive Edge Following consisting of individual scanpoints in Cartesian coordinates; threshold d_{thr} representing minimal distance between point and the fitted line to divide the segment / consider the point as a corner

Output: List $\mathcal{E} = \langle p_1, \dots p_o \rangle$ of additional window edge candidates (corners)

function SEGMENTIZE AND FIND CORNERS($\mathcal{S}, d_{thr}$)
  for all $s_i \in \mathcal{S}$ **do**
    Line l is defined by endpoints of s_i
    Find point $p_{max} \in s_i$ with largest distance d_{max} from line l
    if $d_{max} > d_{thr}$ **then**
      Divide segment s_i into 2 subsegments $\langle P_1, \dots P_{max} \rangle$ and $\langle P_{max}, \dots P_m \rangle$
      Replace segment s_i in $\mathcal{S}$ with the obtained subsegments
      $l_p \leftarrow$ line parallel to l going through point P_{max}
      $d_{p,uav} \leftarrow$ distance of line l_p from the UAV
      $d_{l,uav} \leftarrow$ distance of line l from the UAV
      if $d_{p,uav} < d_{l,uav}$ **then** ▷ take only corners sticking out towards UAV
        Add P_{max} to $\mathcal{E}$ as corner point type
      end if
    end if
  end for
  return $\mathcal{E}$
end function

the endpoints of each segment to fit the line. Other alternatives exist, such as the Split & Merge algorithm, which uses all the points in each segment to fit the line.

When the algorithm is finished, a complete list of window edge candidates is produced. The list includes all detected candidates, one type detected based on a sudden change in range value from the SEF, and the other type detected as corners by the IEPF.

3.2.2 Linking Edges to Existing Window Estimates

Given the list of window edge candidates, it is necessary to determine which candidates can correspond to a window. The first option is to use existing window estimates, provided by the Kalman filter-based estimator, which can be initially generated based on previous detections from RealSense depth data. The second option is to detect the windows based on a priori information about the window size and altitude, as described in Section 3.2.3.

The existing window estimates are given as sets of estimated window center position $C = [c_x, c_y, c_z]^T$, width w_w, height w_h, and four corners $\gamma_{1\dots4} = [\gamma_{i,x}, \gamma_{i,y}, \gamma_{i,z}]^T$. The window estimates are assumed to be without tilt. Therefore, x- and y-coordinates of corners both located on the left or right side of the window are equal. Let γ_1 be a left window corner and γ_2 be a right window corner. The current UAV position is given as $U = [u_x, u_y, u_z]^T$.

The algorithm for finding edges corresponding to existing window estimates is described in Algorithm 3. It goes through all the available estimates. All window estimates that are located at altitudes, where they cannot be seen by the UAV, are skipped. This is done by

Algorithm 3 Find Corresponding Edges to Existing Estimate

Input: List $\mathcal{E} = \langle E_1, \dots E_n \rangle$ of window edge candidates; list $\mathcal{W}$ of existing window estimates; UAV altitude u_z; thresholds $h_{tol}; d_{cor,max}; w_{w,min}; w_{w,max}$
Output: List $\mathcal{D}$ of window edge pairs and window estimate id (w, E_1, E_2)

```
function FIND CORRESPONDING EDGES(E, W)
    for all w ∈ W do
        h_{w,uav} ← |u_z − c_z|
        if h_{w,uav} < w_h/2 + h_tol then                ▷ Skip windows at wrong altitude
            ▷ d_{sum,i} - weighted sum of difference from expected distance to center and
              distances from corners
            Find E_1 with minimal d_{sum,1} and d_{cor,1} < d_{cor,max}
            Find E_2 with minimal d_{sum,2} and d_{cor,2} < d_{cor,max}
            if E_1 found and E_2 found and ‖E_1 − E_2‖_2 is between limits then
                if SPACE_IS_EMPTY(E_1, E_2) then
                    Add E_1, E_2 to D for estimate w
                end if
            end if
        end if
    end for
    return D
end function
```

checking whether the absolute altitude difference between window center and the UAV does not exceed a given threshold as

$$h_{w,uav} < \frac{w_h}{2} + h_{tol}, \tag{3.5}$$

where $h_{w,uav}$ is the absolute difference in altitude, w_h is window height and h_{tol} is an altitude error tolerance, depending on maximal expected UAV tilt and maximal detection distance. Then, the algorithm traverses all the window edge candidates and tries the find a pair of edges, which best fits the provided estimate. While going through the edge candidates, it tries to find the edges $E_1 = [e_{1,x}, e_{1,y}]^T$ and $E_2 = [e_{2,x}, e_{2,y}]^T$, which minimize the weighted sum of distances defined as

$$d_{sum,i} = d_{cor,i} + \alpha d_{cent} \text{ for corner } i \in \{1, 2\}, \tag{3.6}$$

where $d_{cor,i}$ is the Euclidean 2D distance from the left or right window corner defined as

$$d_{cor,i} = \sqrt{(\gamma_{i,x} - e_{i,x})^2 + (\gamma_{i,y} - e_{i,y})^2}. \tag{3.7}$$

α is a weighting factor, and d_{center} is the difference of the Euclidean 2D distance of the edge candidate from the window center C from the estimated distance given known window width calculated as

$$d_{cent} = \left| \sqrt{(e_x - c_x)^2 + (e_y - c_y)^2} - \frac{w_w}{2} \right|. \tag{3.8}$$

This way, a pair of edges corresponding to the left and right window corners is obtained. Furthermore, the distance $d_{cor,i}$ needs to be lower than a predefined threshold $d_{cor,max}$. If

both edges are found, their mutual distance d_{mutual} is compared with minimal and maximal values allowed for the window width:

$$w_{w,min} < d_{mutual} < w_{w,max}. \tag{3.9}$$

Finally, the function SPACE_IS_EMPTY(E_1, E_2) is used to check whether the space between the edges is empty as follows. Let edge E_1 correspond to laser scan range d_1 and edge E_2 correspond to laser scan range d_2. Then, the maximal range is found as

$$d_{max} = \max(d_1, d_2). \tag{3.10}$$

For all range measurements between E_1, E_2 decide if $d_i > d_{max}$. Then, n_{large} means the number of measurements larger then d_{max} and n means the number of measurements between E_1, E_2. The fraction

$$f = \frac{n - n_{large}}{n} \tag{3.11}$$

then needs to be smaller than a predefined maximal fraction value f_{max}.

After passing through all the window estimates, the pairs of edges linked to the estimates are passed to the window position estimator.

3.2.3 Standalone Detection based on Window Width

When an existing window estimate is not available, it is possible to perform the detection based on apriori-known window width, height, and altitude. As the apriori information is defined in a certain frame of reference, the UAV altitude needs to be precisely estimated in the same frame of reference to use this approach. This approach allows the detection of windows even when the depth data are unavailable, e.g., when the window is filled with smoke, or a problem occurs with the depth camera detection algorithm.

In real-world deployments, the window parameters can be known from previous measurements or can be estimated from observing other windows, for which the depth data are available.

The detection process is described in Algorithm 4. It goes through all possible combinations (E_i, E_j) of edge candidates and filters them based on multiple properties to obtain as reliable detections as possible.

First, the algorithm checks the length, defined by the number of measurements, of the line segments previously produced by the IEPF algorithm connected to the edges E_i and E_j. At least one of the segments needs to be longer than threshold l_{min}. It is assumed that a window will always be connected to a wall that can be seen in the LiDAR data. This filtering step ensures that all the detections generated by noise will be filtered out.

Then, filtering based on the distance between edges is done. The distance between the edges is compared with all the different window widths, which can be expected at the current UAV altitude. The apriori information about windows is predefined as sets of window width, height, and altitude values. First, the algorithm checks that the UAV is at an altitude where it can see the particular window by fulfilling the condition

$$|u_z - c_z| < \frac{w_h}{2} + h_{tol}, \tag{3.12}$$

Algorithm 4 Standalone Detection of Windows from 2D LiDAR Data

Input: List $\mathcal{E} = \langle E_1, \dots E_n \rangle$ of window edge candidates; list $\mathcal{W}$ of apriori-known window widths; thresholds and tolerances
Output: List $\mathcal{D}$ of window edge pairs (E_1, E_2)

```
function DETECT WINDOWS(E, W,...)
    for all E_i ∈ E do
        for all E_j ∈ E do
            if j > i then
                Check that E_i or E_j is connected to long segment
                Skip (E_i, E_j) if it does not correspond to any window width
                if SPACE_IS_EMPTY(E_i, E_j) then
                    Check minimal FOV of the window
                    if UAV is outside then
                        Skip (E_i, E_j) if fraction of finite ranges between the edges is too low
                        if Direction from the UAV to the building is known then
                            Skip (E_i, E_j) in the direction opposite to the building
                        end if
                    end if
                    Add (E_i, E_j) to D
                end if
            end if
        end for
    end for
    return D
end function
```

where u_z is the UAV altitude, c_z is the window center altitude, w_h is the height of the window, and h_{tol} is a specified altitude estimation error tolerance, as described in Section 3.2.2. Then, the distance between the edges is checked to fulfill the condition

$$|d_{E_i,E_j} - w_w| < w_{tol}, \tag{3.13}$$

where d_{E_i,E_j} is the distance between points E_i and E_j, w_w is the apriori-known window width, and w_{tol} is a tolerated error in the window width.

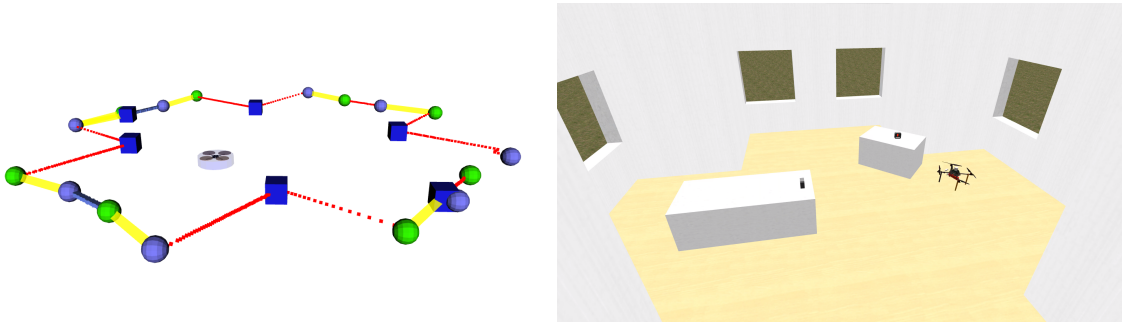
Furthermore, the space between the edges is checked to be sufficiently empty using the function SPACE_IS_EMPTY(E_i, E_j) defined in Section 3.2.2 with the exception that if the UAV is inside, equation (3.10) is modified as

$$d_k < \max(d_i, d_j) + d_{empty}, \tag{3.14}$$

where d_{empty} is the distance outside the window, which should also be empty. It is assumed that there are no obstacles in a small space outside the window, and this assumption is used to filter out possible false detections.

Furthermore, a minimal field of view of the window is required. Given the specified field of view in radians FOV_{min}, the number of measurements n between points E_1, E_2 needs to fulfill the condition

$$n < \frac{FOV_{min}}{\Delta\theta}, \tag{3.15}$$



(a) RViz visualization (b) Image of the visualized room

Figure 3.5: Visualization of window edge candidate detection in Rviz with the UAV inside a building. Red squares represent LiDAR data, sphere markers represent edges detected from range changes by SEF, cube markers represent corners detected by IEPF, yellow lines represent windows detected based on width.

where $\Delta\theta$ is the angle increment in radians between two consecutive LiDAR measurements.

When the UAV is outside the building, the algorithm can skip combinations of edges with an insufficient amount of measured ranges between them (where the LiDAR ray did not reflect). If the fraction of finite ranges obtained between the edges E_i and E_j is lower than threshold f_{min_finite}, the combination is skipped. The reason for this filtering is that the RPLIDAR cannot see black objects in its outdoor setting and could therefore produce false detections when seeing such objects. For example, at the competition there were black shutters closing some of the windows, which the algorithm would otherwise detect as viable entrances to the building. However, this filtering can produce problems with the detection of smoke-filled windows.

Additionally, if the UAV is outside and the direction from the UAV to the building is known, it is possible to filter out all the edge combinations in the half-plane with the center in the opposite direction in order to focus on the observed building and reduce the amount of false-positive detections. The direction to the building is known, e.g., when the UAV is continuously oriented towards the building by another path planning algorithm.

After performing all these filtering steps, the remaining pairs of edges are passed over to the window position estimator as the obtained 2D detections. Figure 3.5 shows a visualization of window edge detection from LiDAR with the UAV inside a building in a simulation.

3.3 Sensor Fusion

In order to have a reliable estimation of the window position continuously available, the individual depth and LiDAR detections are passed to the position estimator, which fuses them together and filters out measurement noise. Combining different types of detections is essential for achieving reliable estimation at every step of the flythrough maneuver. The depth data from the RGB-D camera provide complete information about the 3D position, orientation, and size of the window. However, these detections are quite noisy and are limited by the field of view of the camera. After a certain point of the flythrough, the window can

no longer be visible on the depth camera. Similarly, the camera loses track of the window when the UAV rotates away from it. On the other hand, the LiDAR can detect the window at every point of the trajectory when the LiDAR is located at the same altitude as the window. Furthermore, the range data obtained from LiDAR are very precise, but they only provide information about the window in the 2D horizontal plane. Therefore, combining different sources of detections is very important for safe flight. The sensor fusion is very beneficial under decreased visibility conditions, such as in smoke.

The estimator has the form of a Linear Kalman Filter (LKF) [41][42] for each window, which contains a static model of the window position, orientation and size and updates its state based on the received detections. The LKF is an optimal state estimator based on a linear model of the underlying system, which models the state of the system using Gaussian probability distribution. Although in reality, the system cannot be said to exactly follow the Gaussian distribution, it proves itself to be a reliable approximation.

The estimator enables three different modes of operation:

- *realsense + lidar*: Uses a combination of RealSense depth and LiDAR data for estimation. New estimates are initialized based only on incoming depth detections with all valid edges, and window edges in LiDAR data are detected by linking the data to existing estimates.

- *realsense + lidar + apriori*: Combines RealSense depth and LiDAR data with apriori information about window size and altitude. New estimates can be initialized based on depth detections containing invalid edges. Window edges from LiDAR data are detected based on the apriori information.

- *lidar + apriori*: Uses only LiDAR data combined with apriori information to produce the estimates. New estimates are initialized based on the LiDAR detections combined with apriori information, and the window edges from LiDAR data are detected using the apriori information as well.

3.3.1 Kalman Filter Description

Linear Kalman Filter is a state observer algorithm working with a Linear Time-Invariant (LTI) model of the observed system. The discrete system model is given in the state-space form as

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k + \mathbf{G}\mathbf{u}_k + \mathbf{w}_k, \qquad \mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k) \tag{3.16}$$

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k, \qquad \mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k) \tag{3.17}$$

where $\mathbf{x}_k$ is the state of the system at time step k, $\mathbf{F}$ is the state-transition matrix, $\mathbf{G}$ is the input matrix, $\mathbf{u}_k$ is the input to the system at time step k, $\mathbf{z}_k$ is the measured output of the system, $\mathbf{H}$ is the observation matrix, $\mathbf{w}_k$ is the process noise given by Gaussian distribution with zero mean and covariance matrix $\mathbf{Q}_k$, and $\mathbf{v}_k$ is the measurement noise given by Gaussian distribution with zero mean and covariance matrix $\mathbf{R}_k$.

The Kalman Filter describes the system at every step by Gaussian distribution with state vector $\hat{\mathbf{x}}_k$ and covariance matrix $\mathbf{P}_k$. The state estimation can be divided into two basic

steps. First during the predict step, the state and covariance at time step $k-1$ is propagated to time step k using the system model as

$$\hat{\mathbf{x}}_k^- = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{G}\mathbf{u}_{k-1} \tag{3.18}$$
$$\mathbf{P}_k^- = \mathbf{F}\mathbf{P}_{k-1}\mathbf{F}^T + \mathbf{Q}. \tag{3.19}$$

Then, the measurements are incorporated into the estimation in the update step as

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^- \tag{3.20}$$
$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k \tag{3.21}$$
$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{3.22}$$
$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k\mathbf{y}_k \tag{3.23}$$
$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^-, \tag{3.24}$$

where $\mathbf{y}_k$ is the residual of the measurements, $\mathbf{S}_k$ is the innovation covariance, and $\mathbf{K}_k$ is the Kalman gain.

3.3.2 System Model and Estimation Details

The state $\mathbf{x}$ of the Kalman filter describing a single window is defined as

$$\mathbf{x} = \left[c_x, c_y, c_z, \phi, w, h\right]^T, \tag{3.25}$$

where c_x, c_y, c_z are Cartesian coordinates of the window center, $\phi \in \langle -\pi, \pi \rangle$ is the angle between the projection of the normal vector of the window to the xy-plane and the x-axis (rotation around the z-axis), w is the width, and h is the height of the window. It is assumed that the window is not tilted and perpendicular to the ground plane. The respective components of the state are visualized in Figure 3.6. The position and orientation of the window are specified in a global coordinate frame with a static position of the window.

As the window parameters are modeled as static, the state-transition matrix $\mathbf{F}$ of the system is an identity matrix and the input matrix $\mathbf{G}$ is a zero matrix:

$$\mathbf{F} = \mathbf{I}_{6\times 6} \qquad\qquad \mathbf{G} = \mathbf{0}_{6\times 1}. \tag{3.26}$$

The process noise covariance matrix $\mathbf{Q}$ is a constant diagonal matrix in the form of

$$\mathbf{Q} = \begin{bmatrix} q_{c_x} & & \\ & \ddots & \\ & & q_h \end{bmatrix}, \tag{3.27}$$

where the diagonal elements correspond to the respective elements of state vector $\mathbf{x}$ and are chosen experimentally based on real data to achieve reliable estimator performance.

The observation matrix $\mathbf{H}$ and noise covariance matrix $\mathbf{R}$ depend on what kind of detections are available in each update step. These can be 3D depth detections with a variable number of valid edges, 2D LiDAR detections, or both. The matrices are appropriately chosen before each update step.

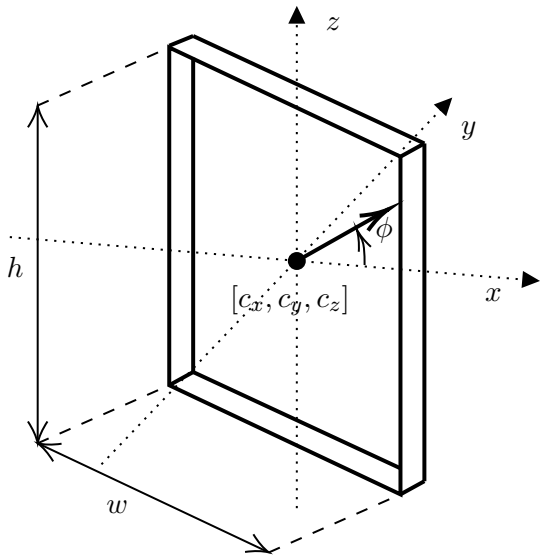


Figure 3.6: Kalman filter state description

Update by Depth Detection

The incoming depth detections can contain a lot of false-positive and noisy data. Therefore, the detections are first filtered by placing several limits on their values. These limits include minimal and maximal values of the detection width w, height h and vertical tilt specified by the z component of the detection normal vector $\mathbf{v}$:

$$w_{min} < w < w_{max}, \tag{3.28}$$

$$h_{min} < h < h_{max}, \tag{3.29}$$

$$|v_z| < v_{max}. \tag{3.30}$$

The complete depth detection provides information about all states of the filter. However, the detector can also output incomplete detections when it cannot see the full window. The state of the filter needs to be appropriately updated based on which window edges were observed by the detector.

The observation matrix for updating all the filter states from a complete window detection is an identity matrix with dimensions equal to the number of LKF states:

$$\mathbf{H}_{depth} = \mathbf{I}_{6\times 6}. \tag{3.31}$$

If the detection does not contain information about some of the states, the corresponding element on the diagonal representing the state in the observation matrix is replaced with a zero. The values of the filter states are either taken directly from the received detection or calculated from its values. Section 3.1 contains description of the depth detections. The measurement vector has the same shape as the LKF state vector:

$$\mathbf{z}_{depth} = \left[c_x, c_y, c_z, \phi, w, h\right]^T. \tag{3.32}$$

If all the edges in the window detection are valid, the center coordinates c_x, c_y, c_z are taken directly from the detection. The ϕ angle is calculated from the window plane normal vector

$\mathbf{v} = [v_x, v_y, v_z]^T$ as

$$\phi = \text{atan2}(v_y, v_x), \tag{3.33}$$

with the function atan2 defined as

$$\text{atan2}(y,x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0, \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases} \tag{3.34}$$

The width w and height h are included directly in the detection parameters.

The situation is different if some of the window edges are invalid. Figure 3.7 depicts all the possible combinations of the valid and invalid edges, which the detection can theoretically contain. A solid line represents a valid edge, while a dashed line represents an invalid edge. An invalid edge is produced if the edge cannot be seen in the camera picture because of the limited field of view of the camera. For every edge combination, a list of the states which will be updated, is shown. Not updated states are replaced with an underscore.

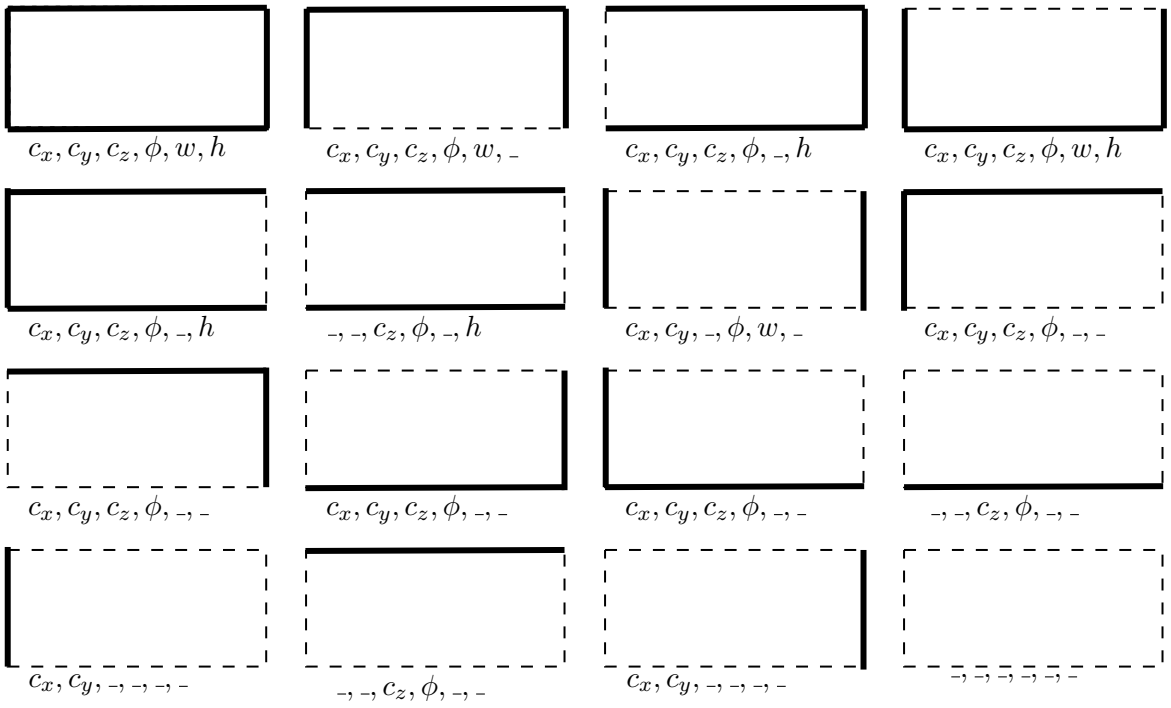


Figure 3.7: Possible combinations of valid window edges and updated filter states tied to the particular combination - dashed line represents invalid edge, underscore represents not updated state.

The measurements vector $\mathbf{z}$ from an incomplete detection is generated in a series of steps based on which edges are valid by combining the received detection and existing estimate.

Similar process is used for initialization of a new estimate from incomplete depth detection and apriori data, as described in Section 3.3.2

1. If top or bottom edge is valid, the angle ϕ is calculated from the detection normal vector as it is assumed that the detection includes correct information about the orientation of the window.
2. If both top and bottom edges are valid, the center altitude c_z and height h are updated.
3. Similarly, if both left and right edges are valid, the center coordinates c_x, c_y, angle ϕ and width w are updated.
4. If c_z was not updated yet and top or bottom is valid, c_z is calculated from the mean z-coordinate of the upper or lower window corners and the estimate's height h.
5. If c_x, c_y were not updated yet and left or right is valid, c_x, c_y are calculated from mean coordinates of left or right corners, angle ϕ and width w. (assuming that the orientation of the window is static in the used coordinate frame)

The measurement covariance matrix for depth detections is a 6×6 constant diagonal matrix in the shape of

$$\mathbf{R}_{depth} = \begin{bmatrix} r_{depth} & & \\ & \ddots & \\ & & r_{depth} \end{bmatrix}, \tag{3.35}$$

where r_{depth} is a constant value chosen experimentally to achieve reliable estimator performance.

Update by LiDAR Detection

LiDAR detection always provides only partial information about the window. The output of the detector consists of two points $E_1 = [e_{1x}, e_{1y}]$ and $E_2 = [e_{2x}, e_{2y}]$, one on the left and the other on the right edge of the window. From this information, the measurement vector

$$\mathbf{z}_{lidar} = [c_x, c_y, \phi, w]^T \tag{3.36}$$

can be calculated. The 2D position of the center of the window can be calculated as

$$c_x = \frac{e_{1x} + e_{2x}}{2} \tag{3.37}$$

$$c_y = \frac{e_{1y} + e_{2y}}{2}. \tag{3.38}$$

The orientation angle can be calculated as

$$\phi = \text{atan2}(e_{1y} - e_{2y}, e_{1x} - e_{2x}) + \frac{\pi}{2} \tag{3.39}$$

and finally the window width can be obtained as

$$w = \sqrt{(e_{1x} - e_{2x})^2 + (e_{1y} - e_{2y})^2}. \tag{3.40}$$

The observation matrix used for updating the appropriate LKF states is then

$$\mathbf{H}_{lidar} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{3.41}$$

The measurement covariance matrix corresponding to the LiDAR detections is then chosen as a 4×4 constant diagonal matrix

$$\mathbf{R}_{lidar} = \begin{bmatrix} r_{lidar} & & \\ & \ddots & \\ & & r_{lidar} \end{bmatrix}, \tag{3.42}$$

where r_{lidar} is a again a constant value chosen experimentally based on LKF performance on real-world data.

If multiple LiDAR detections are available at a single estimator step, the update step of the Kalman Filter is performed several times sequentially to incorporate all the available detections.

Update by Both Detection Types

If both detection types are available at the same time, both are used to update the estimate at the same step. In such case, the measurement vector is constructed as a concatenation of $\mathbf{z}_{depth}$ from equation (3.32) and $\mathbf{z}_{lidar}$ from equation (3.36) as

$$\mathbf{z}_{both} = [\mathbf{z}_{depth}, \mathbf{z}_{lidar}]^T. \tag{3.43}$$

Similarly, the observation matrix is a 10×6 matrix constructed as

$$\mathbf{H}_{both} = \begin{bmatrix} \mathbf{H}_{depth} \\ \mathbf{H}_{lidar} \end{bmatrix} \tag{3.44}$$

and the measurement covariance matrix is a 10×10 diagonal matrix

$$\mathbf{R}_{both} = \begin{bmatrix} \mathbf{R}_{depth} & 0 \\ 0 & \mathbf{R}_{lidar} \end{bmatrix}. \tag{3.45}$$

Linking Detections to Existing Estimates

The received RealSense depth detections are always linked to the existing estimates based on the 3D Euclidean distance between the center of the estimate and the center of the received detection. If this distance is lower than a threshold d_{depth_thr}, the detection is linked to the estimate. Otherwise, a new estimate is initialized based on the detection if the detection fulfills criteria for new estimate initialization.

Linking of received LiDAR detections depends on the chosen estimation mode. If the *realsense+lidar* mode is chosen, LiDAR detections are made directly for an existing estimate (Section 3.2.2) and as such, can be straightforwardly linked back to it.

If the $realsense + lidar + apriori$ or $lidar + apriori$ mode is chosen, the estimate is linked based on the 2D Euclidean distance between the center of the existing estimate and the center of the received detection and based on the condition that the absolute difference in altitude of the estimate and the detection should not exceed a specified threshold.

If the estimate was initialized before entering the building, the detections from inside the building can have opposite orientation to the estimate. In such case, the yaw orientation of the detection is flipped by 180 degrees.

New Estimate Initialization

If a received detection cannot be linked to an existing estimate, a new estimate linked to a new LKF instance is initialized. The initialization of new estimates depends on the currently chosen mode of estimation.

If the $realsense + lidar$ mode is chosen, a new estimate can only be initialized based on received RealSense depth detection with all valid edges. The initial state estimate vector $\mathbf{x}_0$ is then filled with values obtained exclusively from the received detection, and the initial state covariance matrix $\mathbf{P}_0$ is initialized as an identity matrix.

If the $realsense + lidar + apriori$ mode is chosen, a new estimate can be initialized based on a RealSense depth detection containing both valid and invalid edges. The initial state estimate vector $\mathbf{x}_0$ is generated from the received detection and apriori information about window altitude and size using the same sequence of steps that is used in Section 3.3.2 for generation of measurement vector from incomplete window detection. The only difference from the measurement vector generation is that step 5 (calculating center coordinates c_x, c_y from left or right valid edge) additionally requires the angle ϕ to be calculated in one of the previous steps because there is no apriori information about the window orientation. If all the state vector values are successfully generated, a new estimate is initialized.

If the $lidar + apriori$ mode is chosen, a new estimate can be initialized based only on a received LiDAR detection, which cannot be linked to an existing estimate. The initial state estimate vector $\mathbf{x}_0$ is then generated with states c_x, c_y, ϕ and w calculated from the detection and with states c_z and h taken from the apriori information. The apriori-known window width is used for window detection, but the detected width, which can be slightly different, is used for new estimate initialization.

Filtering of Estimates

Some of the window estimates can be initialized based on false detections, especially when the $lidar + apriori$ mode is used. Therefore, it is necessary to filter the estimates to minimize the threat of trying to fly through a false detection. After an estimate is initialized, it is marked as $unsafe$.

When the number of LKF iterations exceeds a specified threshold k_{safe}, the estimate is marked as $safe$ if the variances of states c_x, c_y are lower than the value of a threshold $\sigma^2_{max,safe}$ and at least one LiDAR detection was received (if $realsense + lidar$ or $realsense + lidar + apriori$ mode is used; LiDAR detections are necessary for keeping the estimate during the whole flythrough maneuver). If the estimate does not fulfill these conditions, it is discarded.

Additionally, an estimate is discarded if the number of consecutive LKF steps without detections available (when only the prediction step is performed) exceeds the value of a threshold.

All current estimates are then published for the purpose of navigation during entering or leaving the building. Furthermore, a single estimate is selected as the best one available as a suggestion for the high-level planner deciding which window to use. This selection is performed based on several conditions. The estimate must be marked as $safe$, the variances of states c_x, c_y must be lower than a threshold $\sigma^2_{max,best}$, and the overall number of received detections for the estimate must be larger than a threshold n_{min}. The estimate with the lowest 3D Euclidean distance from the UAV is selected from the estimates fulfilling these conditions, and this estimate is published as the best estimate available.

The threshold values are tuned experimentally to achieve reliable estimator performance on real-world data.

Chapter 4

Path Planning

This chapter contains a description of algorithms for planning and executing a safe flight through a window into or out of the target building and along the walls inside the building. Section 4.1 contains a path planning algorithm generating reference trajectories for flying through a detected window. Section 4.2 describes a wall following algorithm designed for flying in the target building to search for the indoor fire. Section 4.3 describes an algorithm employed for filtering of sensor altitude data to enable a safe transfer through the window into the building and safe flight above obstacles. The algorithms are processing window estimates generated by algorithms described in the previous chapter, onboard sensor data, and UAV position estimate obtained from the currently available localization method.

The planners implement basic functionalities for a higher-level planner or for the human user. The interface enables to request flying through a specific window, start wall following action with certain parameters or cancel the ongoing maneuver.

4.1 Flying through a Window

The planner responsible for flying through the window contains a state machine generating appropriate trajectories based on the UAV position relative to the window position. The planner accepts the list of window estimates from Section 3.3, current UAV position, and requests to fly through a specific window or to cancel the current flythrough maneuver. At a rate of 50 Hz, the planner publishes a list of set-points representing the desired trajectory. The list of set-points is used by the MPC tracker, described in Section 2.2.1.

4.1.1 State Machine

Figure 4.1 depicts a diagram of the state machine used for flying through the selected window. Solid lines connecting the states represent normal behavior loop when no preemption is toggled, and no error occurs. Dashed lines represent various kinds of emergency state transitions.

The state machine contains six different states:

- *idle*: Default state represents that the planner is inactive. The planner waits for a request from high-level planning to fly through a certain window.

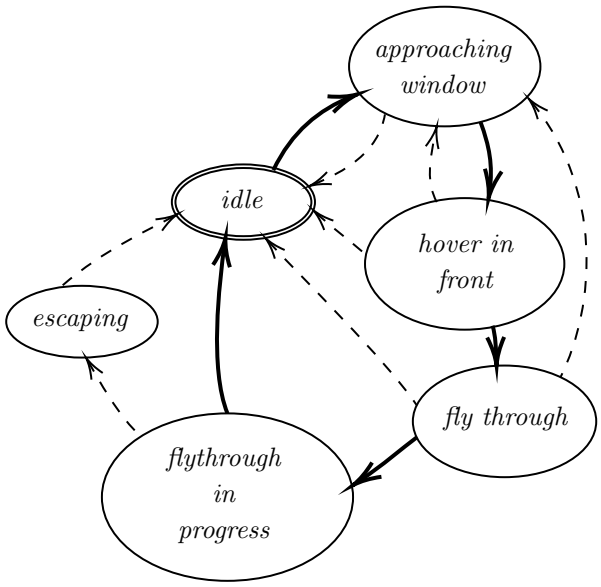


Figure 4.1: State machine for flying through a window

- *approaching window*: The UAV is flying to a position in front of the window while continuously facing the center of the window. Once the 3D Euclidean distance from the goal position and the difference from target orientation drop under predefined thresholds, the state machine switches to the *hover in front* state. If the window estimate is lost, the state machine returns to the *idle* state.

- *hover in front*: The UAV is hovering in front of the center of the window. Once the 3D Euclidean distance from the target hovering position and the orientation difference drop under predefined thresholds and minimal hovering time is reached, the state machine transfers to the *fly through* state. If the window estimate is lost, the state machine returns to the *idle* state.

- *fly through*: The state machine waits for an up-to-date window estimate corrected by new detections and starts the flythrough maneuver by transferring to the *flythrough in progress* state. If the window estimate is lost, the state machine returns to the *idle* state.

- *flythrough in progress*: The UAV flies through the center of the window to a goal position at a predefined distance behind the window while maintaining a constant altitude. When the 2D Euclidean distance from the goal position and the difference from the target orientation drop below thresholds, the flythrough is finished, and the state machine switches to the *idle* state.

  Additionally, a timeout limit is placed on the duration of the flythrough in case the goal position cannot be reached because low-level obstacle avoidance control algorithm prohibits the UAV from reaching the goal. In case of timeout, the state machine switches to *idle* if the UAV passed through the window. The state machine switches to the *escaping* state and returns to the hovering position if the UAV is still in front of the window.

- *escaping*: The flythrough maneuver was canceled, the UAV is returning to its original hovering position in front of the window at maximal allowed velocity.

The flythrough maneuver can be canceled in any of the states because the window estimate has been lost or because of a request from a higher level planner. If the flythrough is canceled during the *approaching window*, *hover in front*, or *fly through* states, the UAV stops following the current trajectory, and the state machine switches to *idle*. If it is canceled during the *flythrough in progress* state, and the UAV has already entered the window, the UAV finishes the maneuver. Otherwise, the state machine switches to the *escaping* state, and the UAV returns to the original hovering position in front of the window.

If a new goal is received while the planner is in the *idle*, *hover in front*, or *fly through* states, it switches to the *approaching window* state and initiates the flythrough maneuver for the newly selected window.

4.1.2 Trajectory Sampling

All the produced trajectories are generated as line trajectories with constant speed and passed to the MPC tracker, described in Section 2.2.1. The MPC tracker then generates feasible trajectories obeying the dynamics of the UAV according to the provided reference.

Let $U = [u_x, u_y, u_z]^T$ be the current UAV position provided by the linear UAV model in the MPC tracker and $P_g = [p_{gx}, p_{gy}, p_{gz}, p_{g\phi}]^T$ be the goal position of the trajectory defined by its Cartesian coordinates in the world coordinate frame and the goal yaw orientation. As the trajectory needs to be sampled with a constant time step Δt, a sampling factor is calculated as

$$\alpha = \frac{v_{des}\Delta t}{s}, \tag{4.1}$$

where v_{des} is the desired flight velocity, Δt is the time step and s is the Euclidean distance between the UAV and the goal position P_g. Trajectory sampling steps are then computed as

$$\Delta x = (p_{gx} - u_x)\alpha, \tag{4.2}$$
$$\Delta y = (p_{gy} - u_y)\alpha, \tag{4.3}$$
$$\Delta z = (p_{gz} - u_z)\alpha. \tag{4.4}$$

The whole trajectory is generated iteratively. Coordinates of the i-th point P_i and the desired UAV orientation can be calculated as

$$p_{i,x} = u_x + i \cdot \Delta x, \tag{4.5}$$
$$p_{i,y} = u_y + i \cdot \Delta y, \tag{4.6}$$
$$p_{i,z} = u_z + i \cdot \Delta z, \tag{4.7}$$
$$p_{i,\phi} = p_{g\phi}. \tag{4.8}$$

In this case, the final UAV orientation is set to all trajectory points.

4.1.3 Goal Point Calculation

Approaching Window

The goal position of the trajectory is calculated as

$$P_g = C - s \cdot d_{front}\mathbf{v}, \tag{4.9}$$

where $P_g = [p_{gx}, p_{gy}, p_{gz}]^T$ is the goal position, $C = [c_x, c_y, c_z]^T$ is the center of the window, d_{front} is the desired distance of the goal position from the center of the window, $\mathbf{v}$ is the unit normal vector of the window plane and s is a constant determined based on the orientation of the vector $\mathbf{v}$ relative to the UAV. Let $\mathbf{w} = [u_x - c_x, u_y - c_y, u_z - c_z]^T$ be the vector pointing from the window center to the drone. Then, s is determined based on the value of the scalar product of $\mathbf{v}$ and $\mathbf{w}$ as

$$s = 1 \iff \mathbf{v} \cdot \mathbf{w} \leq 0 \tag{4.10}$$

$$s = -1 \iff \mathbf{v} \cdot \mathbf{w} > 0 \tag{4.11}$$

Desired velocity v_{appr} is used for generating the trajectory. When approaching the window, the UAV should face the center of the window at every point of the trajectory to maximize the visibility of the window by the depth camera. Therefore, the UAV yaw orientation at each trajectory point is calculated as

$$p_{i,\phi} = \text{atan2}(c_y - p_{i,y}, c_x - p_{i,x}). \tag{4.12}$$

Hovering in Front of the Window

The trajectory is generated in the same way as for approaching the window. A hovering point is calculated in front of the center of the window, and a trajectory is generated towards it. A lower velocity v_{hover} is used while generating the trajectory.

Flying through the Window

A goal position behind the window is generated in a similar fashion as in the previous sections - goal coordinates and orientation in the horizontal plane are calculated as

$$p_{gx} = c_x + s \cdot d_{beh}v_x, \tag{4.13}$$

$$p_{gy} = c_y + s \cdot d_{beh}v_y, \tag{4.14}$$

$$p_{g\phi} = \text{atan2}(s \cdot v_y, s \cdot v_x). \tag{4.15}$$

where d_{beh} is the target distance behind the window. The relative orientation of the window normal vector to the UAV, described by s, flips at the moment of the flythrough because the orientation of the window estimate stays the same in the global coordinate frame.

Afterwards, the desired trajectory to the goal position is generated in the same way as previously with velocity $v_{through}$. The altitude of all the trajectory points is set to be the same as the original UAV altitude at the trajectory start to prevent any sudden changes in UAV altitude during the flythrough maneuver

$$p_{i,z} = u_{0,z}. \tag{4.16}$$

4.2 Path Planning inside the Building

The indoor path planner uses data from a single 2D laser scan to plan local wall-following trajectories to guide the UAV around the room while searching for the fire. It is assumed that the room contains obstacles next to the walls, but no obstacles are present at the center of the room. The approach enables the UAV to quickly scan the entered room and is very computationally efficient as only the raw sensor data are used without the construction of any kind of map.

The path planning consists of two separate steps. First, the UAV moves to a goal position at the desired distance and orientation relative to the wall. Then, the UAV moves along the wall at the desired distance and with the desired orientation relative to the movement direction until scanning of the room is completed or canceled. The planner enables to specify the flight velocity, movement direction, desired distance from the wall, and desired orientation relative to the UAV movement direction.

The trajectory is sampled in the same way as in Section 4.1.2, therefore only the goal point calculation is described.

4.2.1 Approaching the Wall

The planner first finds the closest wall to the UAV and moves the UAV to a position at the desired distance d_{des} and yaw orientation ϕ_{des} relative to it.

Each point in the laser scan obtained from the LiDAR is described by its polar coordinates $[d_i, \theta_i]$, where d_i is the distance from the center of the LiDAR and θ_i is the angle from the x-axis pointing towards the front of the UAV. First, the closest point $P_{closest}$ with the lowest range d_{min} is found. Then, the required length of the trajectory is calculated as

$$d_{traj} = d_{min} - d_{des} \tag{4.17}$$

and the desired goal position P_g in the LiDAR coordinate frame is calculated as

$$p_{gx} = d_{traj} \cos(\theta_{closest}) \tag{4.18}$$

$$p_{gy} = d_{traj} \sin(\theta_{closest}). \tag{4.19}$$

The coordinates of the goal position are then transformed to the world coordinate frame. The goal yaw orientation is calculated as

$$p_{g\phi} = u_\phi + \theta_{closest} + \frac{\pi}{2} + \phi_{des}, \tag{4.20}$$

where u_ϕ is the current UAV yaw orientation in the world coordinate frame. The orientation is normalized to $\langle -\pi, \pi \rangle$. Afterwards, the trajectory to the goal position P_g is generated equally as in Section 4.1.3.

In order to prevent switching between two similarly distant wall segments, the last calculated goal position is used if the current one is too far away or has a significantly different orientation from the previous one.

When the 2D Euclidean distance between the UAV and the goal position and the difference between the UAV orientation and the desired yaw angle drop below predefined thresholds, the wall following is started.

4.2.2 Wall Following

The UAV needs to move along the wall while maintaining the desired distance from it. At the same time, it needs to avoid obstacles placed next to the wall and react quickly to an oncoming wall perpendicular to the one currently being followed. Furthermore, the wall contains windows seen as holes in the data, which the UAV needs to skip without trying to fly through them.

The planner searches for two points on the wall next to the UAV and calculates a wall vector from these two points. Then, it moves along this vector while simultaneously maintaining the desired distance from the wall and checking for obstacles in front of the UAV.

Figure 4.2a shows selection of the first wall point. The planner goes through measured ranges in the circular sector denoted by angle α_1. The sector spans from the starting angle θ_s to the ending angle θ_e. The sector is specified so that it covers the side of the UAV facing the wall and the desired direction of movement, as can be seen in the figure. The bounding angles are calculated to be independent of the choice of the desired UAV orientation relative to the wall ϕ_{des}. The point $P_1 = [d_1, \theta_1]^T = [p_{1x}, p_{1y}]^T$ is obtained as the point corresponding to the shortest range in the sector.

Figure 4.2b depicts selection of the second point. The sector used for searching begins at starting angle

$$\theta_s = \theta_1 + \alpha_{off}, \tag{4.21}$$

where α_{off} is the angle offset from the range corresponding to the first point and ends at angle

$$\theta_e = \theta_s + \alpha_2, \tag{4.22}$$

where α_2 is angle describing the size of the sector. The sector is searched iteratively from the starting angle θ_s to the ending angle θ_e. The first range encountered, which is finite and smaller than a predefined threshold, is used as the second point $P_2 = [d_2, \theta_2]^T = [p_{2x}, p_{2y}]^T$. Afterwards, wall vector $\mathbf{v}$ is calculated from the points P_1 and P_2 as

$$\mathbf{v} = [p_{2x} - p_{1x}, p_{2y} - p_{1y}]^T. \tag{4.23}$$

Figure 4.2c shows calculation of the trajectory goal point P_g. Distance from the wall d_{wall} is calculated as the distance of the UAV from the line l passing through points P_1 and P_2. Error in wall distance is calculated as

$$e_d = d_{wall} - d_{des}, \tag{4.24}$$

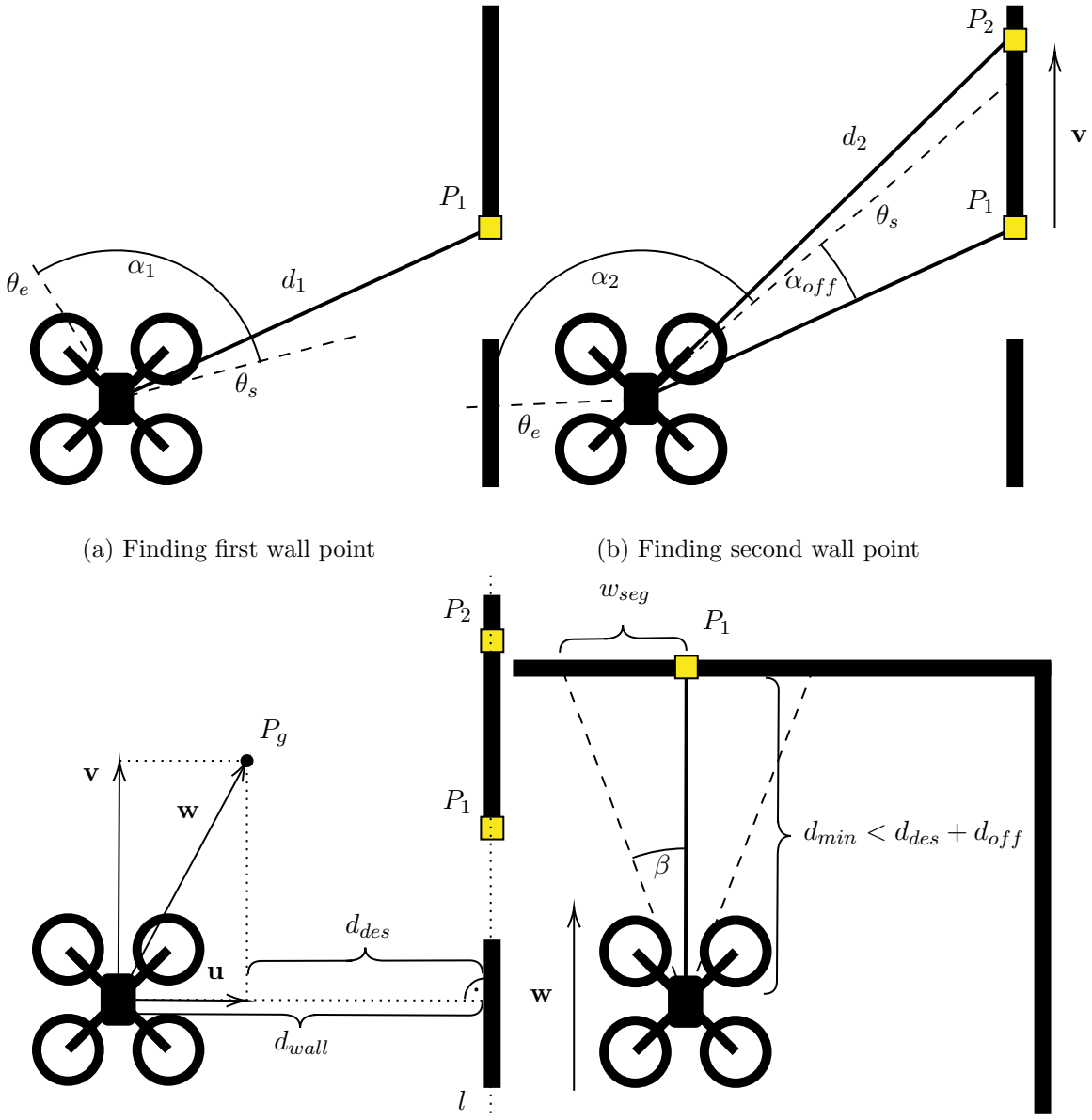
where d_{des} is the desired wall distance. Vector $\mathbf{u}$ is constructed as a vector perpendicular to $\mathbf{v}$ aiming towards the wall. The desired movement vector $\mathbf{w}$ is then constructed by summing the vectors as

$$\mathbf{w} = \mathbf{v} + e_d\mathbf{u} \tag{4.25}$$

and the goal point of the local wall following trajectory is calculated as

$$P_g = U + \mathbf{w}, \tag{4.26}$$

where U is the current UAV position.



(a) Finding first wall point

(b) Finding second wall point

(c) Goal point calculation

(d) Detection of wall in front of the UAV

Figure 4.2: Wall following process

Figure 4.2d displays the detection of a wall in front of the UAV. The planner needs to react quickly to an oncoming wall or obstacle. The planner searches for potential obstacles in a sector with its center defined by the desired UAV movement vector $\mathbf{w}$. The size of the sector scales with the expected distance from the obstacle. The half of the central angle of the sector is calculated as

$$\beta = \operatorname{atan} \frac{w_{seg}}{d_{min,prev}}, \tag{4.27}$$

where w_{seg} is a parameter specifying the half-length of the wall segment, which should be inspected, and $d_{min,prev}$ is the minimal range found in the previous wall following iteration. If

no range from the previous iteration is available, a default β angle value is used. The scaling ensures that the planner will search for the wall in a large enough area without searching on the wall along which the UAV is moving. The sector is searched for the shortest range d_{min}. If the shortest range satisfies the condition

$$d_{min} < d_{des} + d_{off}, \tag{4.28}$$

where d_{des} is the desired wall distance and d_{off} is offset for the oncoming wall detection, a wall is detected. In such case, the point corresponding to the range d_{min} is used as the first point P_1 of the wall vector, the wall vector is consequently constructed using the oncoming wall, and the UAV turns to move parallel to it.

The wall following can be stopped when the UAV returns to its original position after going around the whole room. This is checked based on the distance of the UAV from its starting position and on the condition that the UAV performed a full rotation around its z-axis.

4.3 Altitude Filter

The altitude filter accepts data from rangefinders mounted on the UAV and filters them to provide reliable information about current UAV altitude. The used UAV platform employs a downward-looking rangefinder for height measurement. The data from the rangefinder are passed to the state estimation system described in Section 2.2.2, which passes the data to a Kalman filter and estimates the current UAV altitude based on the rangefinder measurement and barometer data. Because very precise altitude control is required for indoor-outdoor transition and indoor flight, the estimation is done mainly based on the precise rangefinder data. However, this data contains sudden large changes when the UAV flies over obstacles or flies through a window. Therefore, it is necessary to filter this data to ensure that the UAV will always have a reliable altitude estimate without abrupt changes.

The used UAV platform carries two rangefinders, the upper rangefinder is directed upwards and measures the distance from the ceiling while the lower rangefinder is directed downwards and measures the current height above the ground. The rangefinder measurements are also influenced by the tilting of the UAV during its movement. The altitude filter consists of several one-dimensional median filters that filter out the sudden changes in rangefinder data and calculate offset, which is added to the measured data and passed to the state estimation system. The estimation of UAV altitude based on rangefinder data is depicted in Figure 4.3 with r_{low} representing the lower and r_{up} representing the upper rangefinder measurement. For example, during the flight from position a) to position b) in the figure, an offset value (marked as alt_off in the image) is calculated and added to the r_{low} value to be passed to the state estimation system.

4.3.1 Median Filter Description

The median filter is a nonlinear filtering technique capable of detecting abrupt changes (or sharp edges) in data. The input data are iteratively passed through the filter, and the

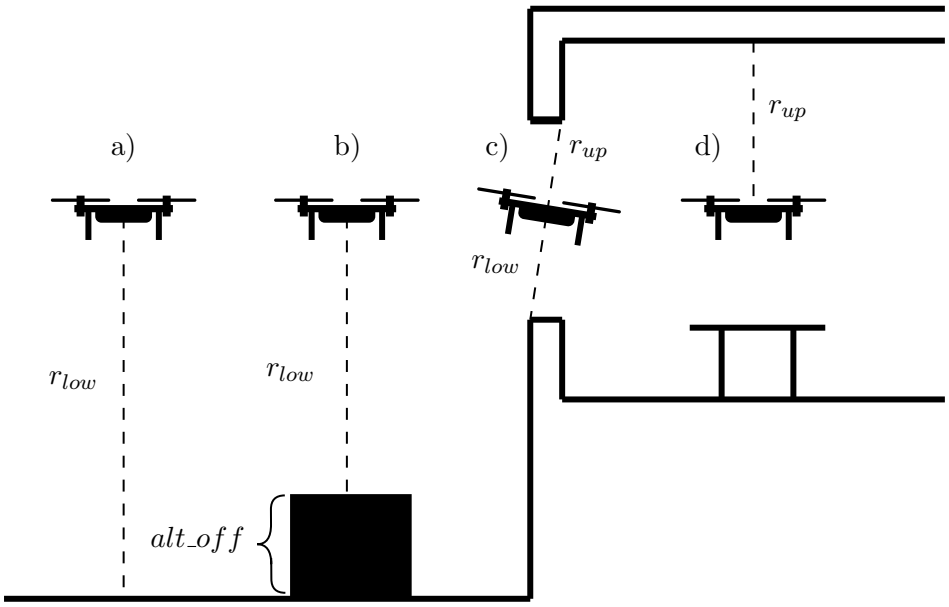


Figure 4.3: Altitude filter - a) no obstacles below UAV, b) UAV above obstacle, c) UAV flies through a window, d) UAV inside a building

median value m of a floating window of k input values is calculated. For each input value v, the absolute difference from the window median is calculated as

$$m_{diff} = |v - m|, \tag{4.29}$$

where v is the new value and m is the current median. If the difference m_{diff} is larger than maximal difference δ_{max}, the value is marked as invalid, and an abrupt change in the data is detected.

4.3.2 Outside Filtering

If the UAV is outside, only the lower rangefinder data are used. The data are passed to a set of two median filters. The first one, with buffer size k_{edge} and maximal difference δ_{edge}, is used for the detection of sharp edges in data (when the UAV flies over the edge of an obstacle). The second one, with buffer size k_{stable}, and maximal difference δ_{stable}, is used for the detection of a situation when the rangefinder data are approximately stable in value. The median filter instances implement the methods for detection if a new value is valid (sufficiently close to the window median) or not.

The process of modifying the lower rangefinder value by calculating offsets is described in Algorithm 5. The algorithm calculates two separate offset values: $altitude\_offset$ and $stable\_offset$. Both are initialized to zero in the beginning. The altitude offset is recalculated every time the edge median filter detects a sudden change in the range value as

$$altitude\_offset \leftarrow altitude\_offset + (r_{prev} - r),$$

where r is the current rangefinder value and r_{prev} is the previous rangefinder value. The rangefinder value is modified as

$$r_{out} \leftarrow r + altitude\_offset.$$

Algorithm 5 Filtering of Lower Rangefinder Data

Input: Current range value r, previous range value r_{prev}, last stable range value r_{stable}, current altitude offset $altitude\_offset$, current stable offset $stable\_offset$, threshold $d_{off,max}$
Output: Modified lower rangefinder value r_{out}

function CALCULATE ALTITUDE OFFSET(r)
  if not EDGE_FILTER_IS_VALID(r) **then** ▷ Sharp edge detected
    $altitude\_offset \leftarrow altitude\_offset + (r_{prev} - r)$
  end if
  if STABLE_FILTER_IS_VALID(r) **then** ▷ Measured values are stable
    if $|stable\_offset - altitude\_offset| > d_{off,max}$ **then**
      $altitude\_offset \leftarrow stable\_offset + (r_{stable} - r)$
    end if
    $stable\_offset \leftarrow altitude\_offset$
    $r_{stable} \leftarrow r$
  end if
  $r_{prev} \leftarrow r$
  $r_{out} \leftarrow r + altitude\_offset$
  return r_{out}
end function

However, calculating the offset only based on one edge detection filter would be imprecise because the obstacles do not always produce sufficiently sharp edges. For example, when the UAV flies out of the building, it tilts forward, and the rangefinder picks up the wall below the window in its measurements. Therefore, it is necessary to correct the value of the $altitude\_offset$ from trustworthy measurements when the data are sufficiently stable (i.e., when flying at a constant height above a flat surface). The correction is performed using the second median filter and $stable\_offset$, which is calculated only from the stable values. When the filter detects that the incoming data are sufficiently stable and the value of $stable\_offset$ significantly differs from $altitude\_offset$, the $altitude\_offset$ is corrected as

$$altitude\_offset \leftarrow stable\_offset + (r_{stable} - r),$$

where r_{stable} is the last stable rangefinder value.

The parameters of the median filters need to be carefully tuned so that the filters do not pick up the change of UAV altitude as a sudden change in data caused by flying over obstacles.

4.3.3 Inside Filtering

When the UAV is inside, the altitude should be estimated mainly based on the distance from the ceiling because it is assumed that there are many obstacles on the ground while the ceiling is approximately flat. The stability of the upper rangefinder values is checked by another median filter with buffer size k_{upper} and maximal difference δ_{upper}. When the values are stable and the UAV is inside, the difference between two consecutive upper measurements is calculated as

$$d_{up} = r_{up} - r_{up,prev}. \tag{4.30}$$

The previously outputted lower rangefinder value is then modified by this difference without using the lower rangefinder data as

$$r_{out} = r_{out,prev} - d_{up}. \tag{4.31}$$

If the upper rangefinder values are not stable, the data from the lower rangefinder are used, as described in the previous section.

During the transition from the outside to inside, the altitude filter starts using the upper rangefinder data when it receives information that the UAV has flown through the window, the upper rangefinder data are stable, and that the sum of lower and upper rangefinder value approximately corresponds to the expected height of the room.

After the transition from the inside to outside, the first values of $altitude\_offset$ and $stable\_offset$ are calculated from the last output value r_{out}, which was calculated inside.

Chapter 5

Simulations

Before deployment to real UAV hardware, the correct performance of the algorithms was extensively verified in the Gazebo robotic simulator. Gazebo offers a 3D simulation environment of robots operating in various complex environments. A 3D model of the competition environment was constructed and used in the simulator to test the solutions of all fire challenge subtasks in an authentic environment.

This chapter describes a simulation of a single UAV entering the target building and exploring its interior. In the simulated scenario, the UAV first approached the first floor of the building and flew along its walls while simultaneously detecting multiple windows. Then, one of the detected windows was selected to be used as an entrance to the building. The UAV flew through the window and began moving along the walls of the room while facing away from the followed wall and detecting multiple windows usable for exiting the building. After exploring the entire room, the UAV flew back outside using the same window.

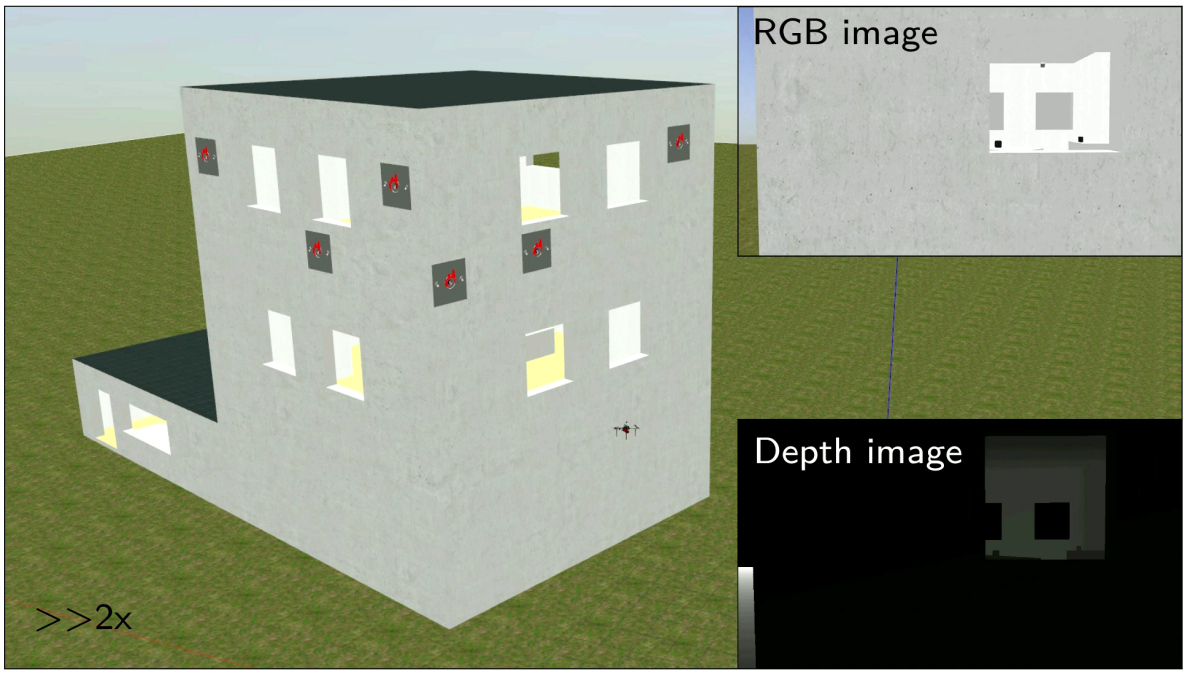
The whole simulation can be seen in the video at [1]. Figure 5.1a shows the simulation environment with the UAV outside the building preparing to enter the selected window. The figure also contains RGB and depth images from the camera onboard the UAV. Figure 5.1b depicts the simulation of flying inside the building. A visualization of the LiDAR data with all existing window estimates and the current UAV position is shown on the left of the figure. On the right, an RGB image from the onboard camera and a screenshot of the simulation environment are shown.

5.1 Window Detection and Flythrough

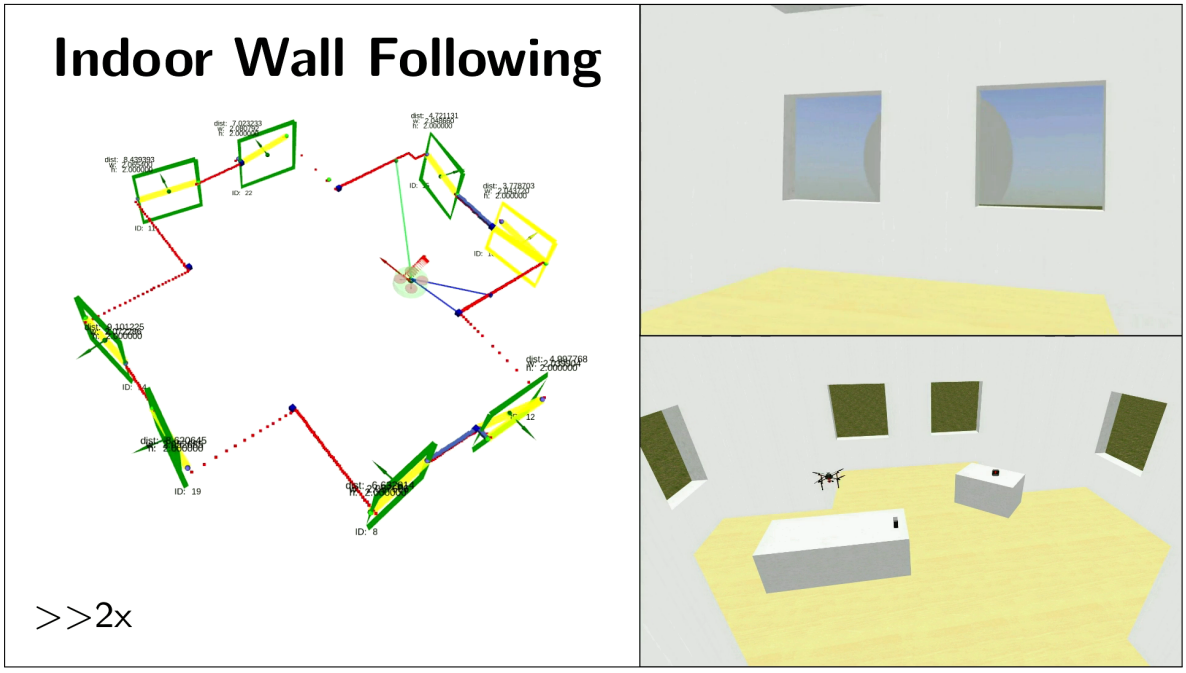
The simulated building contains 2 by 2 meters large windows on its first and second floor. During its flight, the UAV detected windows from both the inside and outside of the building. The simulation was performed in the *lidar + apriori* mode of window estimation, and afterwards, the estimation algorithm was rerun using the remaining estimation modes to verify their correct performance.

The simulation video contains a visual comparison of the three estimation modes. All three modes have been verified as viable approaches to window parameters estimation. The

[1] `http://mrs.felk.cvut.cz/pritzvac-master-thesis`



(a) Simulation environment with the UAV outside the building



(b) Indoor wall following simulation with the UAV detecting multiple windows

Figure 5.1: Screenshots of the simulation

selection of the mode to be used in real-world experiments depends on the available sensors, flight conditions, and available a priori information about the target building.

5.2 Indoor Wall Following

After entering the building, the UAV started following the walls counter-clockwise at the speed of 0.5 m/s while facing away from the wall (90° relative to the direction of movement). The desired distance from the walls was set to $d_{des} = 2$ m and the additional offset for detection of wall in front of the UAV was set to $d_{off} = 1$ m.

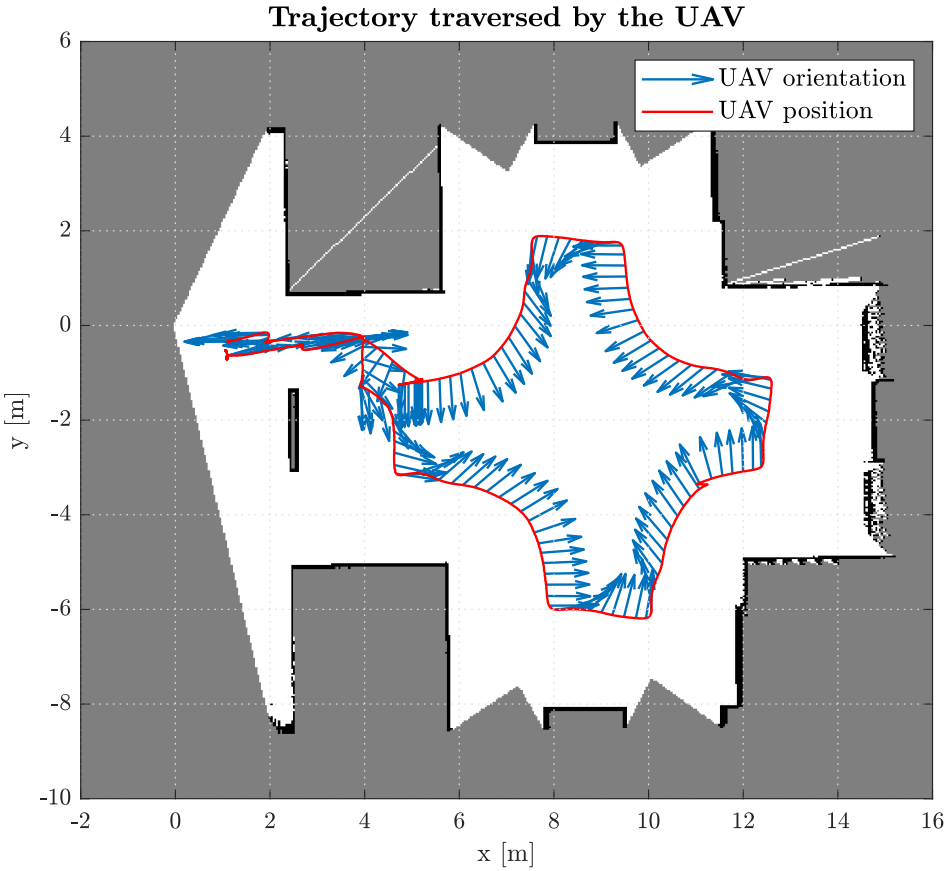


Figure 5.2: Wall following trajectory inside the building

The UAV flew around the entire room and exited the building using the same window as was used for entering. Figure 5.2 contains a visualization of the UAV trajectory inside the building. A map obtained from the Hector SLAM algorithm is drawn in the background of the figure.

5.3 Altitude Filter

Figure 5.3 shows the input and output of altitude filter during the simulation. At time $t = 52$ s, the UAV entered the window to fly into the building and at time $t = 125$ s the UAV left the building. At first, the filtered value (altitude filter output) was equal to the lower rangefinder value. When the UAV entered the window, the lower rangefinder value dropped while the filtered value stayed constant as the altitude filter algorithm added an offset to the

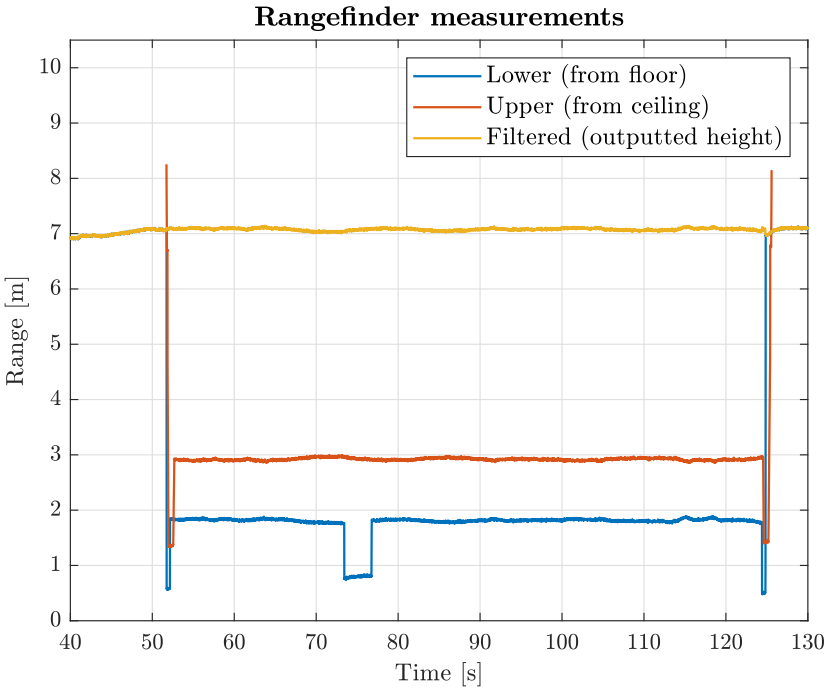


Figure 5.3: Rangefinder measurements - input and output of altitude filter

received measurements. The UAV flew over an obstacle at time $t = 74$ s. The lower rangefinder value decreased, but the filtered value stayed the same as it is calculated from changes seen by the upper rangefinder. When the UAV left the building, the lower rangefinder value was used again, and the altitude offset was reset to zero.

Chapter 6

Real-World Experiments

This chapter describes multiple deployments of the designed algorithms in real-world experiments. Experiments 6.1-6.4 were all performed at a small scale experimental wooden building constructed at the university campus. The building consists of two floors, which are 4×4 m wide and approximately 2.3 m high. The building contains 1.85 m wide and 1 m high closable windows.

Experiments 6.2-6.4 were performed with a smoke-filled environment. The ADJ VF1300 Fog Machine was used for smoke generation.

Experiment 6.5 was performed in the desert in the United Arab Emirates during final preparations for the MBZIRC 2020 competition. Section 6.6 describes the deployment of the designed algorithms at the competition itself. Finally, Section 6.7 provides a summary of the results obtained from the performed experiments.

All described experiments can be seen in the video at [1].

6.1 First Floor Flight with RealSense and LiDAR

During this experiment, the UAV autonomously entered and left the testing building through 2 different windows open on its first floor. The *realsense + lidar* estimation mode was used for window position estimation. A small UAV based on the DJI F450 frame was used for this experiment. During the whole flight, the LiDAR-based Hector SLAM was employed for localization.

The UAV started on the ground outside the building and flew to a position in front of the window on the first floor by flying to manually entered set-points. On its way, the UAV detected both the ground-floor and the first-floor window. Then, the UAV autonomously entered the building through the first-floor window, was turned by flying to a manually entered set-point, and autonomously left the building through the other open window after its detection.

The whole experiment can be seen in the provided video. Figure 6.1 displays a screenshot of the video showing the UAV in front of the first window. The figure contains an image from

[1] `http://mrs.felk.cvut.cz/pritzvac-master-thesis`

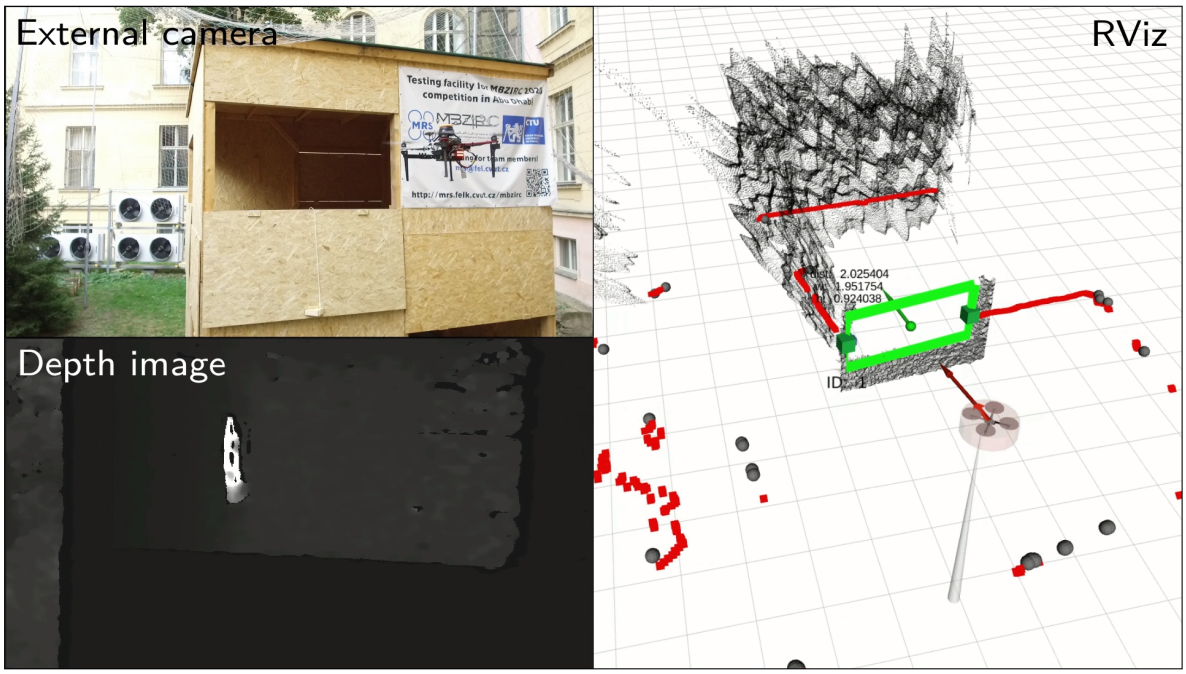


Figure 6.1: Flying through a window on the first floor

an external camera, a depth image from the onboard UAV camera, and an RViz visualization. The RViz visualization displays the LiDAR data as red squares, a depth cloud obtained from the depth camera, and the detected window.

6.1.1 Window Position Estimation

Figures 6.2 and 6.3 display estimation of all states of the window, which was used to enter the building during the experiment. Figure 6.2 shows estimation of the window center coordinates c_x, c_y, c_z and the orientation ϕ. Figure 6.3 shows estimation of the window width w and height h, and contains a visualization of the valid edges seen in the depth detections over time.

The figures displays the time period from the time when the window first appeared on the depth camera up to the time when the UAV flew through the window. The depth detections have already been preprocessed. The detections with widths and heights outside the required limits or detections too far from the initialized estimate have been filtered out. The time scale was adjusted, so that time $t = 0$ s corresponds to the window estimate initialization.

It can be seen that the window was visible on the camera for some time before the estimate initialization, but the detection was not complete (with all four window edges visible), and therefore the estimator waited for a complete detection. The x- and y-coordinates, orientation ϕ, and window width were estimated from both depth and LiDAR data. The plots show that the estimate was initially created with values corresponding to the depth detection but then quickly converged to the values of the more precise LiDAR data. The z-coordinate and window height were estimated only by filtering the depth detections. It can be seen that

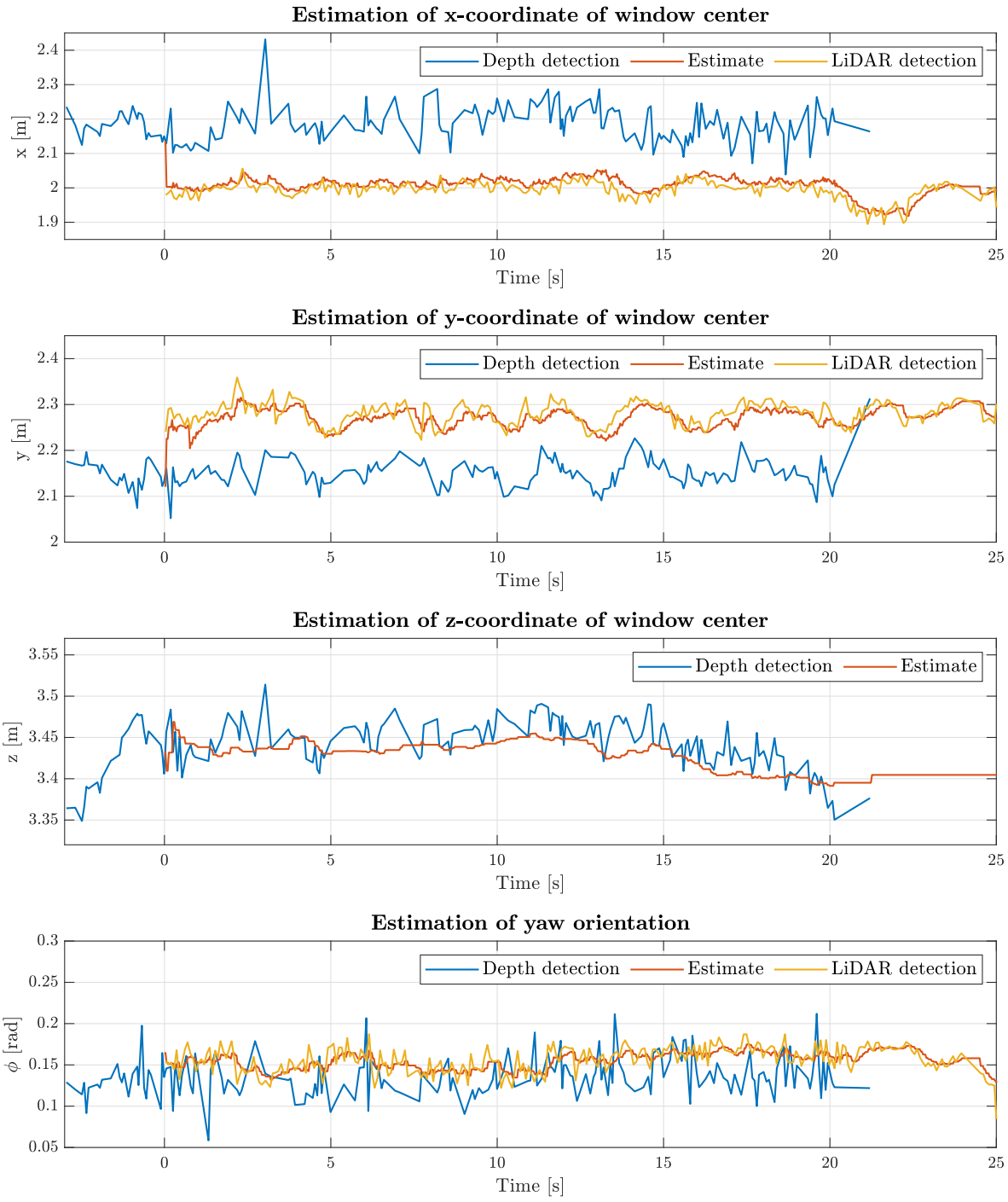


Figure 6.2: Estimation of window coordinates and orientation from depth and LiDAR data

after $t = 21$ s, the window was no longer visible on the depth camera because the UAV was too close to it. After that time, the x- and y-coordinates, orientation, and width were estimated from the LiDAR data only, and the estimated z-coordinate and height stayed constant.

Furthermore, the plots show that the window height was updated only until time

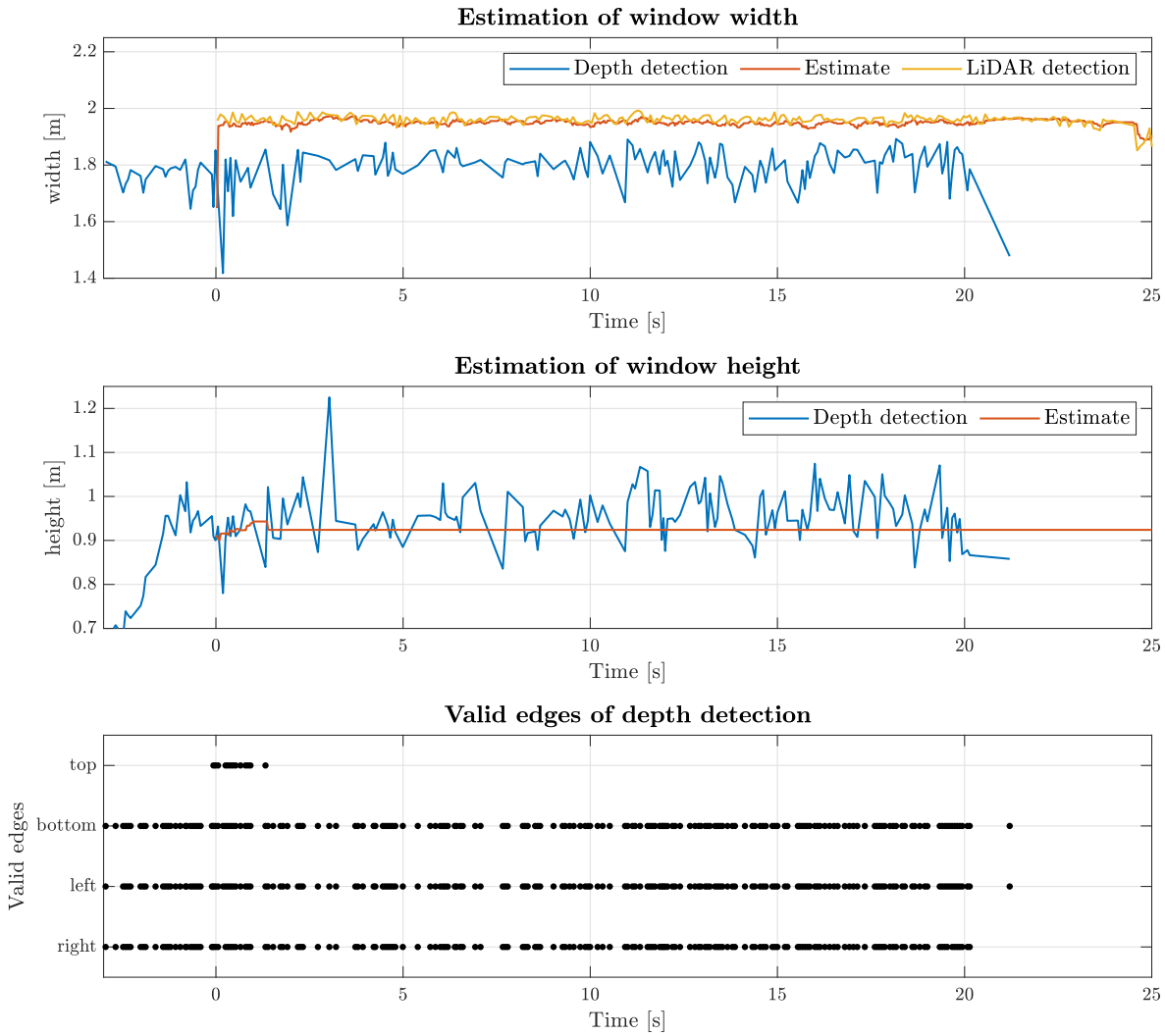


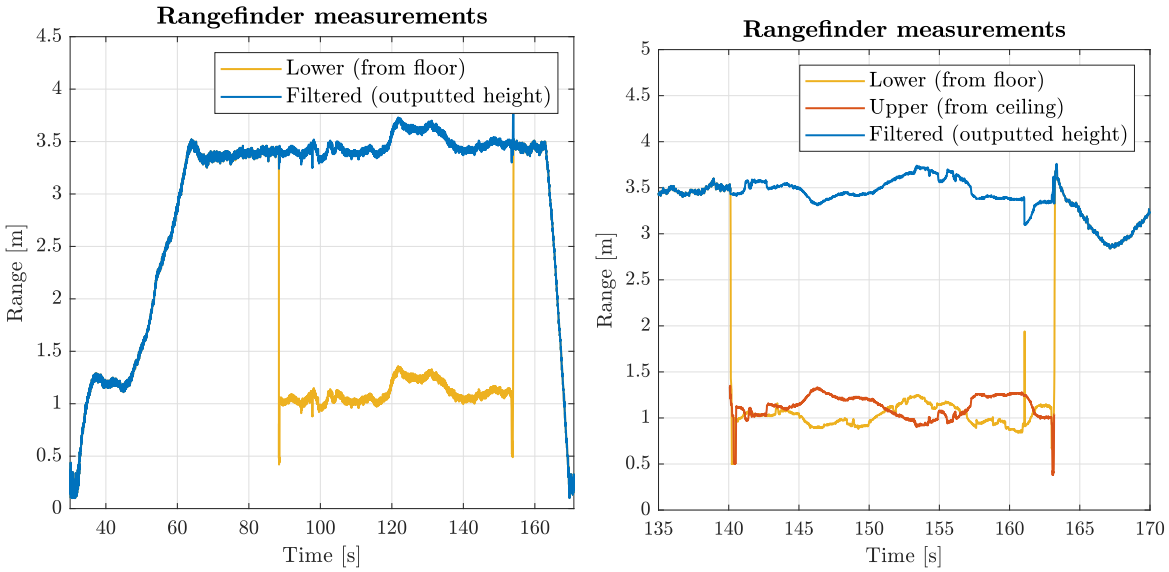
Figure 6.3: Valid edges from depth data and estimation of window width and height

$t = 1.5$ s and stayed constant afterwards. After that time, the top window edge was not visible in the depth data, and therefore the estimator did not have sufficient information for updating the window height.

6.1.2 Altitude Filter

Figure 6.4a contains a plot of rangefinder measurements and altitude filter output from this flight. Only the lower rangefinder was available during this flight. It can be seen that the lower rangefinder value and the filtered value were equal until time $t = 90$ s when the UAV entered the window. There can be seen sudden drops in the lower value caused by flying over the window edge. Inside, the lower value shows the height of approximately 1 m above the ground. The filtered value stays unaffected by entering and leaving the building.

Figure 6.4b shows a plot from another flight through the first floor when both the upper and lower rangefinders were available. During this flight, the UAV entered the first



(a) First flight with only lower range available

(b) Second flight with both ranges available

Figure 6.4: Inputs and output of altitude filter from flights in the first floor

floor through a window, turned around, and left the building through the same window. The figure shows the time period of the flight starting before entering the building and ending after leaving the building. As in the previous case, the filtered value was unaffected by the sudden changes in input values during flight through the window. While inside the building, the filtered value was calculated based on changes in the upper rangefinder value. After the UAV left the building, the altitude offset was reset to zero, and the lower rangefinder value became equal to the filtered value again.

6.2 Ground Floor Flight with LiDAR in Smoke

In this experiment, flying through windows and indoor wall following was combined in a single flight. During the experiment, the UAV started in front of a ground floor window. After the window was detected, the UAV entered the building and started exploring it by flying along its walls while facing away from the currently followed wall. After the UAV completed a full flight around the room, the wall following was finished, and the UAV left the room using the same window as was used for entering. The experiment consisted of two separate flights. The first flight was performed under good visibility conditions without smoke, and the second flight was performed in a smoke-filled room. LiDAR-based Hector SLAM was used for localization during this experiment.

During this experiment, the *lidar + apriori* estimation mode was used for window detection because smoke causes significant problems to the depth camera image (as described in detail in the following section). The estimation algorithm had a priori information about the approximate window width, height, and altitude.

The wall following parameters were set as follows. The desired distance of the UAV from the wall was set to $d_{des} = 1.5$ m. The additional offset for detection of wall in front of

the UAV was set to $d_{off} = 0.5$ m. The desired yaw orientation relative to the direction of movement was set to $\phi_{des} = 90°$ in order to observe the center of the room.

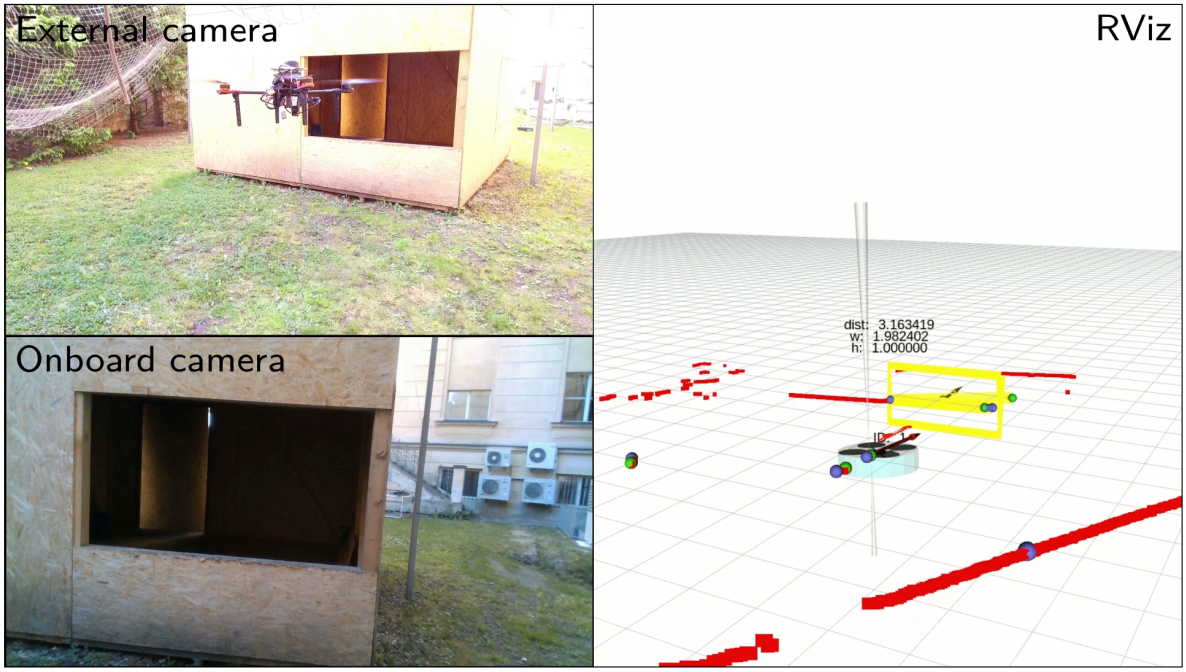


Figure 6.5: Flying through a ground floor window under normal conditions

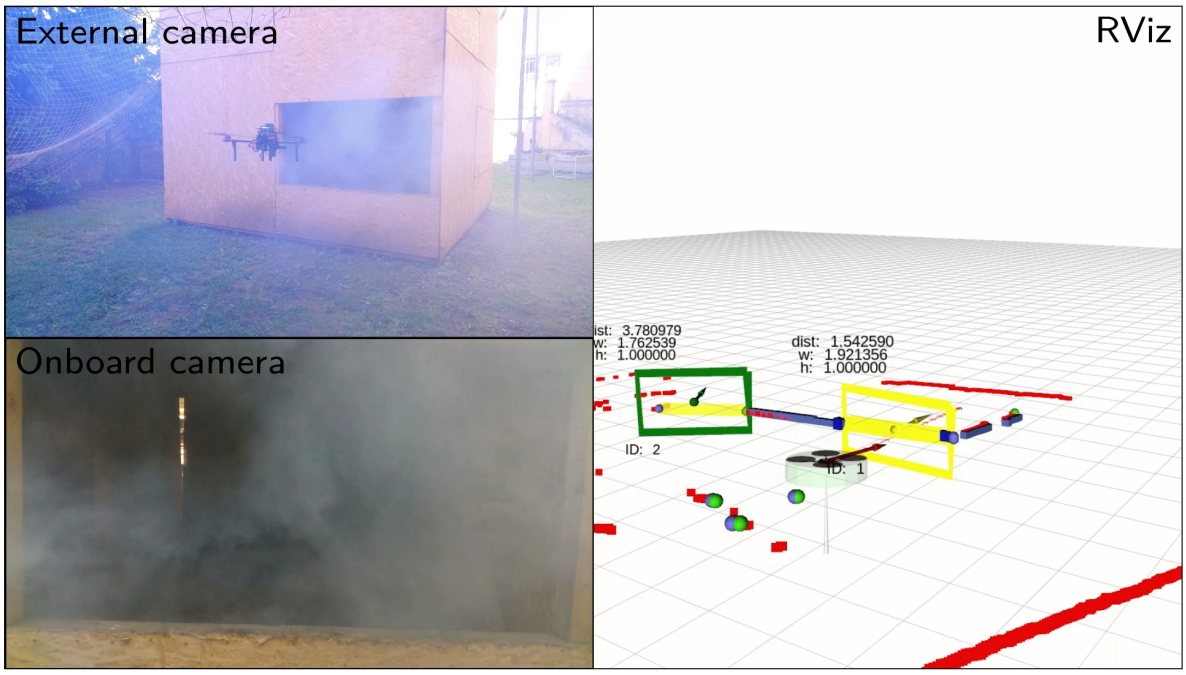


Figure 6.6: Flying through a ground floor window in a smoke-filled environment

The whole experiment can be seen in the provided video. Figures 6.5 and 6.6 contain screenshots of the video with the UAV outside the building under normal conditions and

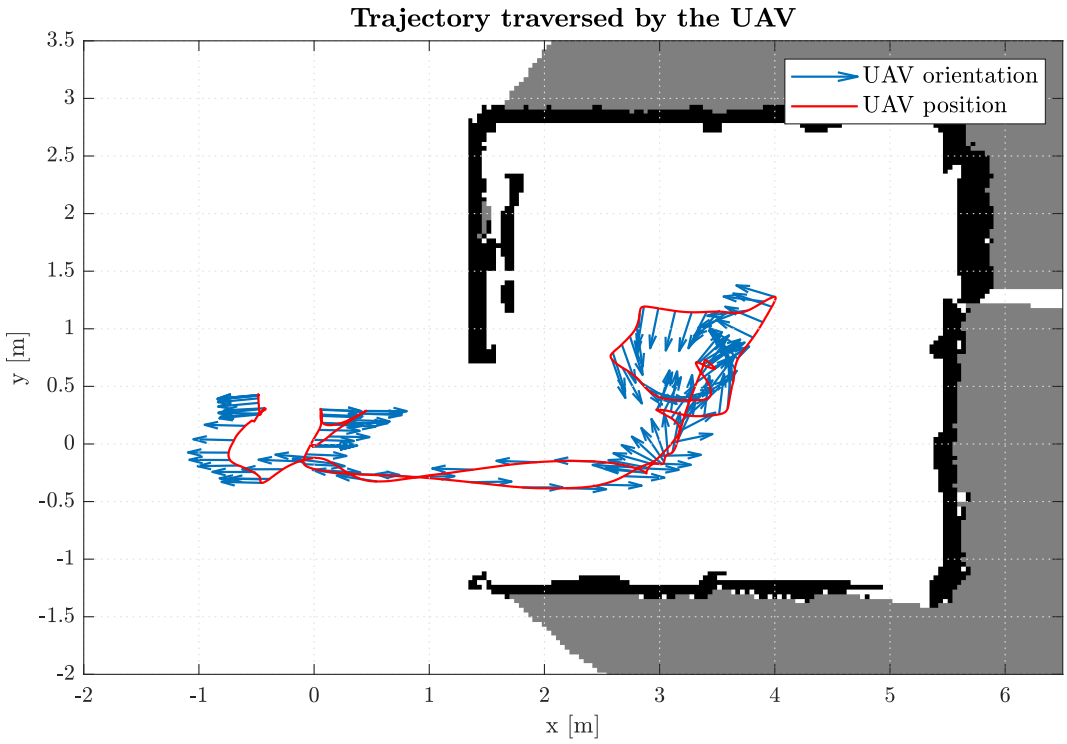


Figure 6.7: Trajectory traversed by the UAV during the ground floor experiment

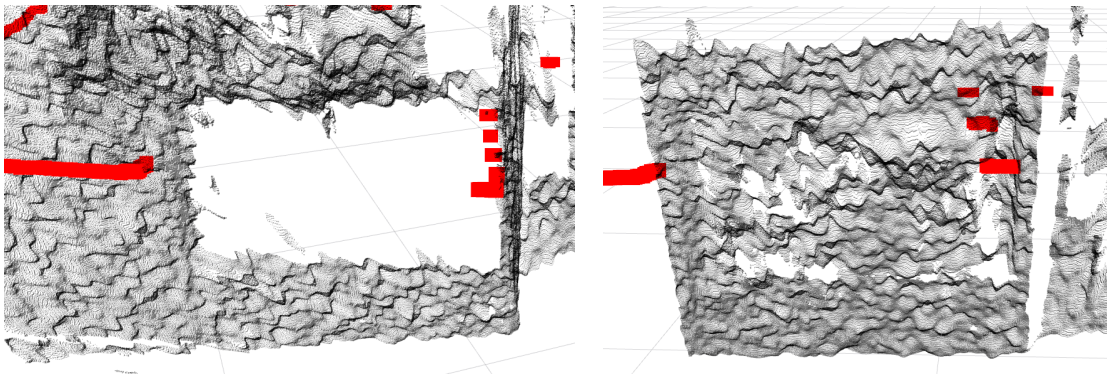
in a smoke-filled environment. The figures show pictures from the onboard RGB camera, external views of the UAV, and RViz visualizations. The RViz visualizations contain LiDAR data depicted by red squares, the UAV, the currently best window estimate drawn as a yellow rectangle, and the Figure 6.6 also contains a false window detection drawn as a green rectangle on the left.

Figure 6.7 shows the trajectory traversed by the UAV during the second flight in a smoke-filled room. The map produced by the Hector SLAM algorithm is drawn in the background of the figure.

The UAV successfully performed both flights without any collisions and proved the ability of the proposed algorithms to function under smoke conditions. However, as can be seen both in the video and in the Figure 6.6, the *lidar + apriori* estimation mode can suffer from a number of false detections because it utilizes only 2D LiDAR data and detects windows only based on their a priori known width. The experiment was performed in a small space with many obstacles close to the target building. The number of false detections is expected to be much lower close to high-rise buildings with significantly more free space around them.

6.3 Smoke Influence on RealSense Depth Camera

An experiment has been performed to evaluate the influence of smoke conditions on the RealSense D435 depth camera. During the experiment, the UAV was manually controlled and hovered in front of the window outside the building. The window was gradually filled with smoke. The experiment can again be seen in the provided video.



(a) Low amount of smoke (b) Large amount of smoke

Figure 6.8: Evaluation of smoke influence on the RealSense D435 depth camera

Figure 6.8 shows the effect of smoke on the depth cloud produced by the RealSense depth camera. The figure shows depth clouds produced by the camera observing the window from the outside and 2D LiDAR data represented by red squares. It can be seen that with a low amount of smoke, only a small amount of noise is present in the data. However, if thick smoke appears in the environment, the window can become completely filled and can be completely unrecognizable in the data.

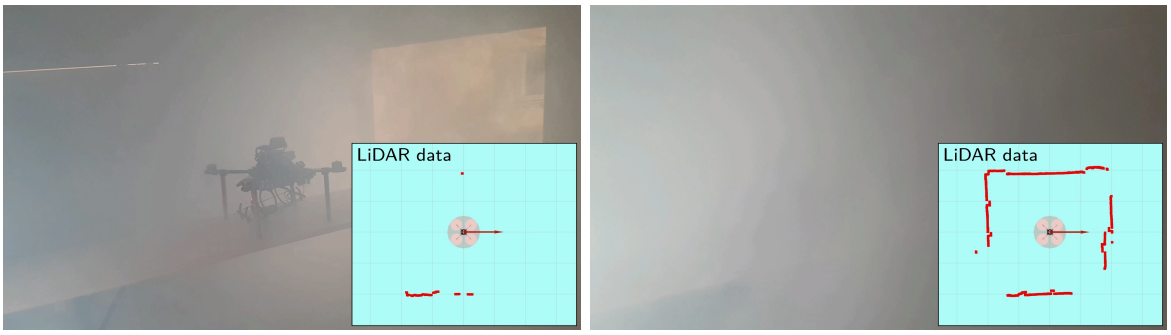
It is apparent that smoke causes significant problems to the depth camera and can render it completely useless depending on the amount of smoke present. Furthermore, the results of running the window detection algorithm described in Section 3.1 on the smoke-filled data show that the current version of the detection algorithm cannot detect the window filled with smoke.

6.4 Smoke Influence on RPLIDAR-A3

An experiment has been performed to assess the influence of smoke-filled environment on LiDAR performance. The UAV has been placed inside a square 4 by 4 m room, which was gradually filled with smoke. The effect of smoke on the data measured by the RPLIDAR-A3 laser scanner was observed. The performance of the LiDAR was evaluated in two modes of operation - the *stability* mode meant for outdoor use and the *sensitivity* mode meant for indoor use. The respective LiDAR performance in these two modes can be seen in the provided video. Figure 6.9 shows a comparison of the two modes. The figure contains images from an external camera observing the UAV and RViz visualizations of the LiDAR data representing the walls of the room.

It can be seen that in the *stability* mode, the RPLIDAR performance is significantly reduced by the smoke. In thick smoke, the RPLIDAR can completely lose sight of the surrounding walls. On the other hand, the *sensitivity* mode proves to be sufficiently resistant to the smoke conditions. Even in very thick smoke, the walls are still mostly visible, and the RPLIDAR suffers from only occasional loss of data. Therefore, the RPLIDAR-A3 proves itself to be usable for deployment in a smoke-filled environment.

However, the *sensitivity* mode is meant for indoor use because of its low environment



(a) Stability (outdoor) mode (b) Sensitivity (indoor) mode

Figure 6.9: Evaluation of smoke influence on RPLIDAR-A3 laser scanner

light elimination. Sunlight present in outdoor environments can cause complete loss of LiDAR measurements in this mode. Therefore, reliable deployment of the system in real-world conditions requires using the *stability* mode while outside and switching to the *sensitivity* mode while entering the building.

6.5 Testing in the Desert

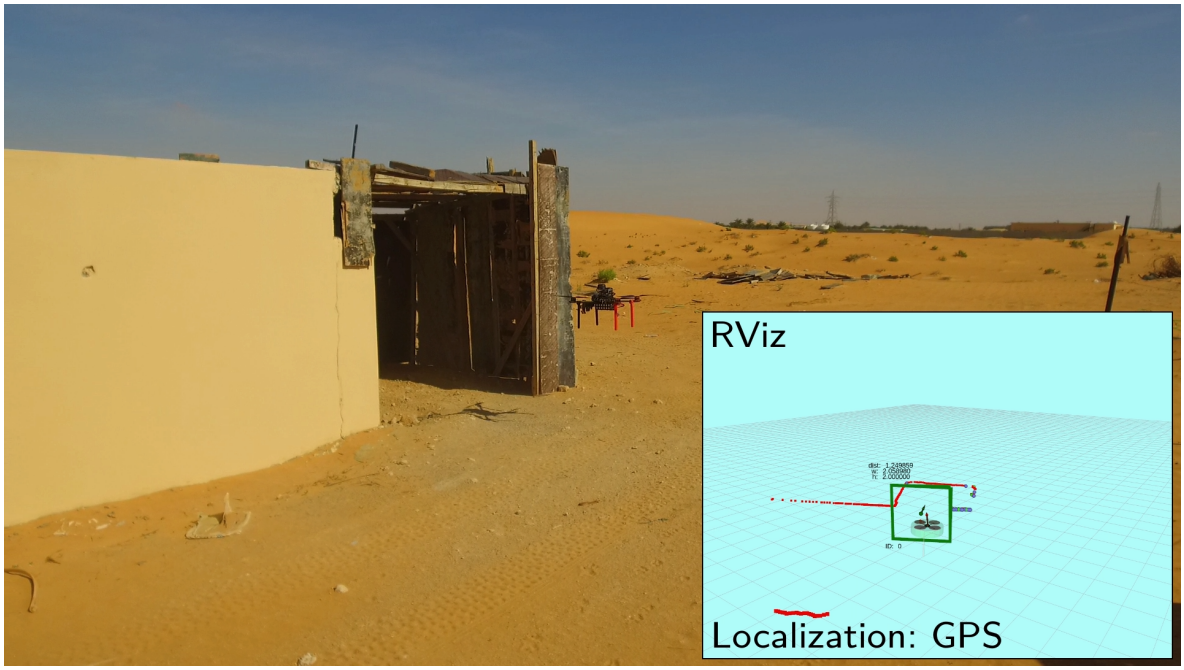


Figure 6.10: UAV flying through an improvised building entrance in the desert

Before the competition itself, additional experimental verification was performed in the desert. This experiment focused mainly on testing correct switching of the UAV localization method between GPS and LiDAR-based Hector SLAM and on partial verification of the state machine used to integrate all the subtasks together.

For the experiment, an improvised wooden building was constructed. The building was placed next to a long wall, was approximately $2.5 \times 3 \times 2$ m large, and contained a single 2×2 m large entrance. During the experiment, the UAV started autonomously flying along the wall until it detected the building entrance while using GPS for localization. Then, the UAV flew in front of the window, and the localization method switched to Hector SLAM. The UAV flew inside, turned around, and flew back outside. Then, the localization method switched back to GPS, and the UAV returned to its starting position.

In this experiment, the *lidar + apriori* window estimation mode was used. The whole experiment can be seen in the provided video. Figure 6.10 contains a screenshot of the video showing the UAV in front of the building entrance. The figure contains an RViz visualization of the LiDAR data and the detected window.

6.6 Deployment at the MBZIRC Competition

The competition environment itself consisted of a $50 \times 60 \times 20$ m large arena containing a 16 m high building. Figure 6.11 displays a 3D model of the building captured by the Leica laser scanner. The building consisted of 3 floors. The ground floor contained one fire to be extinguished by the ground robot, and the first and second floors each contained a simulated indoor fire. There was a single 2×2 m large window open on each floor while the rest of the windows were closed with black covers. The second floor was filled with artificially generated smoke.

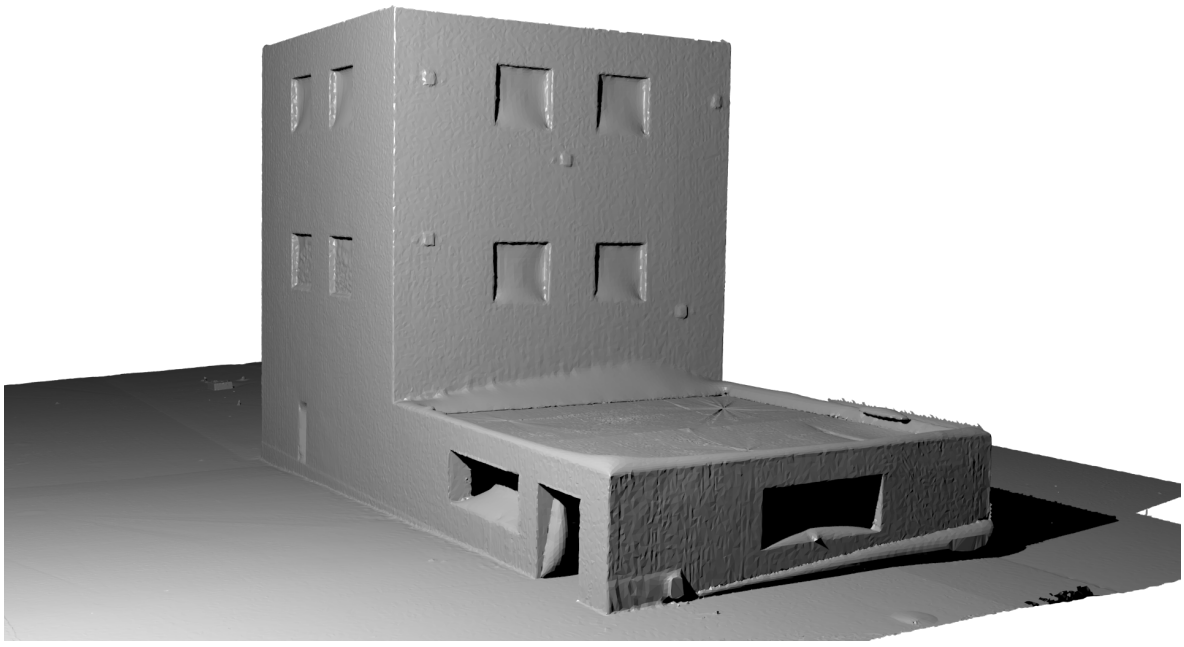


Figure 6.11: 3D model of the competition building from Leica laser scanner

The *lidar + apriori* estimation mode was used for window detection during the competition. From the recorded data, it was discovered that the RPLIDAR laser scanner in its outdoor mode cannot detect the black window covers placed in the closed windows. Therefore,

the detection algorithm was modified to filter out window detections that do not contain any LiDAR measurements of the inside of the room. However, this filtering would not work with a smoke-filled environment because the LiDAR would not be able to obtain a sufficient number of measurements through the smoke either.

Being able to reliably distinguish between closed windows and windows filled with smoke would require either fusing another kind of data with the detections (e.g., detections from RGB images) or switching the RPLIDAR to indoor mode outside the building. More testing would be required to determine whether the UAV would be able to fly inside the building in direct sunlight with RPLIDAR in indoor mode.

The performance of the window detection algorithm can be seen in the provided video. The data were recorded during one of the trial runs at the competition. The UAV was manually controlled and flew along a single wall of the building. The recorded data show detection of multiple open windows while skipping the windows closed by black covers. Figure 6.12 contains a screenshot of the video showing the LiDAR data and detected windows in RViz and a picture from the onboard RGB camera.

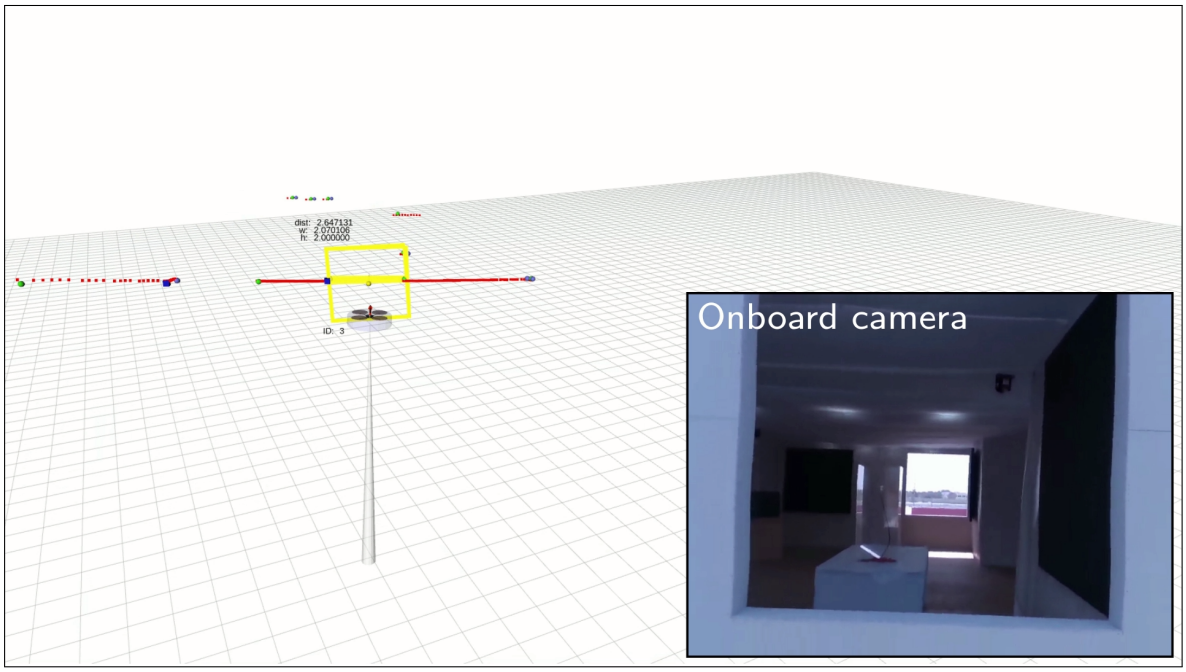


Figure 6.12: Window detection at the MBZIRC competition building

Unfortunately, the integration of all subtasks needed for extinguishing the indoor fires together proved to be too complex for the limited time period available. For this reason, the designed path planning algorithms were never employed during the competition, the UAV never flew inside the building, and only the window detection and estimation algorithms along with the altitude filter (for filtering of height data when flying over the ground floor of the building) were tested there.

6.7 Summary of Experimental Results

All the proposed algorithms have been verified in real-world experiments. The window detection from depth data is able to obtain a complete set of information about the window position, orientation, and size. The window detection from 2D LiDAR data is able to obtain positions of the left and right edges of the window. Fusing the two types of detection benefits from the ability of the depth detection to obtain complete information about each window and from the precision of the LiDAR data, which is also not limited by the field of view of a camera. The availability of a priori information about window size and altitude enables the option to generate window estimates from incomplete depth detections or to detect windows only from 2D LiDAR data. However, the window detection based on only LiDAR data and a priori information can contain a number of false detections depending on the environment in which the UAV operates.

The wall following algorithm for indoor flight has been tested in both normal and decreased visibility conditions and was able to navigate the UAV around the target room without any collision with obstacles. The altitude filter algorithm for supplying correct altitude information to the UAV localization subsystem has been verified to correctly calculate altitude offsets when entering or leaving a building or when flying above obstacles.

The performance of the sensors required for window detection and indoor flight in a smoke-filled environment has been evaluated. The RealSense depth camera suffers from decreased visibility caused by the smoke, which significantly limits its ability to be used for window detection during real firefighting scenarios. The RPLIDAR-A3 laser scanner in its indoor mode performs very well under the simulated smoke conditions. However, the indoor mode suffers from weak environment light elimination, and a safe indoor-outdoor transition in a firefighting scenario therefore requires switching between these two modes during the transition process.

The functionality of the UAV state estimation subsystem has been verified to be able to seamlessly switch between GPS-based localization and LiDAR-based Hector SLAM.

Unfortunately, the integration of all subsystems together has proved to be too complex for the limited time period available at the competition, and the UAV did not fly inside the building at the competition itself.

Chapter 7

Conclusion

In this thesis, multiple algorithms providing reliable guidance of a UAV performing autonomous flight into a building through a window and autonomous exploration of the indoor space were designed. The motivation behind this thesis was the firefighting challenge of the MBZIRC 2020 contest. The navigation algorithms designed in this thesis were crucial for the subtask of flying inside the building and extinguishing a simulated indoor fire.

An algorithm for window detection from 2D laser scanner data based on either existing window estimate or a priori information was designed. A Kalman filter-based estimator was used to fuse its output with the output of an existing algorithm for window detection from data provided by a depth camera and for filtering out measurement noise and false detections. A path planning algorithm was designed for realizing safe transfers through the detected windows into or out of the target building. An algorithm for constructing local wall following trajectories based on a single laser scan was developed to be used for the indoor exploration. Furthermore, a range data filtering algorithm was designed for calculating altitude offsets during the indoor-outdoor transitions and flight over obstacles so that data from upward and downward-looking rangefinder could be used to obtain reliable UAV altitude estimates.

All proposed algorithms have been thoroughly tested in authentic computer simulations before deployment to real UAV hardware. Finally, all proposed algorithms have been tested in multiple real-world experiments both under normal conditions and in a smoke-filled environment, and their performance has been evaluated. Videos of the performed simulations and experiments are available on `http://mrs.felk.cvut.cz/pritzvac-master-thesis`.

The entire thesis assignment has been successfully fulfilled. According to the assignment, the following tasks have been completed.

- A description of the UAV platform and the respective sensors used for the fire challenge was provided in Section 2.1. A description of the control and estimation pipeline used for controlling the UAV was provided in Section 2.2.

- Algorithms for window detection from LiDAR data and fusion with detections from the RealSense depth camera were designed and implemented. Their description can be found in Chapter 3.

- Algorithms for planning flight trajectories through the detected window and for safe flight inside the building were designed and implemented. Their description can be found in Chapter 4.
- All the proposed algorithms have been tested in a simulated environment. The description of the performed simulation can be found in Chapter 5.
- All the proposed algorithms have been verified in real experiments, and their performance was evaluated. The algorithms have been tested both under normal conditions and in a smoke-filled environment. The description of the experiments can be found in Chapter 6.

7.1 Future Work

Future work on window detection and the indoor-outdoor transition could focus on the use of different sensors. The 2D LiDAR employed in this thesis was very precise and was able to function even in a smoke-filled environment. In future work, a 3D LiDAR sensor could be employed for the window detection task. A 3D LiDAR could obtain complete information about the position, orientation, and size of the window and would be less limited by its field of view than the sensors employed in this thesis.

The window detection algorithm for depth camera data could be modified to work more robustly in smoke-filled environments by filtering some of the noise caused by the smoke. However, thick smoke significantly decreases the quality of data obtained from the depth camera, and therefore the possible improvements are only limited.

The window detections could be further fused with, e.g., detections from an RGB camera image, which could prove beneficial for recognizing closed windows.

The proposed indoor path planning algorithm is able to generate only local trajectories based on currently available sensor data. Further work could consider constructing a map of the environment and using, e.g., a frontier-based approach for building exploration. Furthermore, the trajectories could be planned based on 3D sensor data obtained from either a depth camera or a 3D LiDAR.

Finally, the proposed algorithms could be evaluated in a larger-scale scenario closer to a real firefighting situation.

Bibliography

[1] L. Merino, F. Caballero, J. R. Martinez-de Dios, I. Maza, and A. Ollero, "An Unmanned Aircraft System for Automatic Forest Fire Monitoring and Measurement," *Journal of Intelligent and Robotic Systems*, vol. 65, pp. 533–548, Jan. 2012.

[2] A. Restas, "Forest Fire Management Supporting by UAV Based Air Reconnaissance Results of Szendro Fire Department, Hungary," in *2006 First International Symposium on Environment Identities and Mediterranean Area*, Jul. 2006, pp. 73–77.

[3] A. Viguria, I. Maza, and A. Ollero, "Distributed Service-Based Cooperation in Aerial/Ground Robot Teams Applied to Fire Detection and Extinguishing Missions," *Advanced Robotics*, vol. 24, no. 1-2, pp. 1–23, Jan. 2010. [Online]. Available: https://www.tandfonline.com/doi/full/10.1163/016918609X12585524300339

[4] R. Chen, H. Cao, H. Cheng, and J. Xie, "Study on Urban Emergency Firefighting Flying Robots Based on UAV," in *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 1, Dec. 2019, pp. 1890–1893, iSSN: 2381-0947.

[5] H. Qin, J. Q. Cui, J. Li, Y. Bi, M. Lan, M. Shan, W. Liu, K. Wang, F. Lin, Y. F. Zhang, and B. M. Chen, "Design and implementation of an unmanned aerial vehicle for autonomous firefighting missions," in *2016 12th IEEE International Conference on Control and Automation (ICCA)*, Jun. 2016, pp. 62–67.

[6] P. Pecho, P. Magdolenová, and M. Bugaj, "Unmanned aerial vehicle technology in the process of early fire localization of buildings," *Transportation Research Procedia*, vol. 40, pp. 461–468, Jan. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352146519302303

[7] A. Imdoukh, A. Shaker, A. Al-Toukhy, D. Kablaoui, and M. El-Abd, "Semi-autonomous indoor firefighting UAV," in *2017 18th International Conference on Advanced Robotics (ICAR)*, Jul. 2017, pp. 310–315.

[8] M. Popp, G. Scholz, S. Prophet, and G. F. Trommer, "A laser and image based navigation and guidance system for autonomous outdoor-indoor transition flights of MAVs," in *2015 DGON Inertial Sensors and Systems Symposium (ISS)*, Sep. 2015, pp. 1–18, iSSN: 2377-3464.

[9] M. Popp, S. Prophet, G. Scholz, and G. F. Trommer, "A novel guidance and navigation system for MAVs capable of autonomous collision-free entering of buildings,"

Gyroscopy and Navigation, vol. 6, no. 3, pp. 157–165, Jul. 2015. [Online]. Available: https://doi.org/10.1134/S2075108715030128

[10] S. Zhou, G. Flores, E. Bazan, R. Lozano, and A. Rodriguez, "Real-Time Object Detection and Pose Estimation using Stereo Vision. An application for a Quadrotor MAV," Nov. 2015.

[11] G. Flores, S. Zhou, R. Lozano, and P. Castillo Garcia, "A Vision and GPS-Based Real-Time Trajectory Planning for MAV in Unknown Urban Environments," May 2013.

[12] R. C. Julian, C. J. Rose, and H. Hu, "Cooperative Control and Modeling for Narrow Passage Traversal with an Ornithopter MAV and Lightweight Ground Station," *International Conference on Autonomous Agents and Multiagent Systems*, vol. 1, p. 8, Jan. 2013.

[13] L. Mejías, S. Saripalli, P. Campoy, and G. S. Sukhatme, "Visual servoing of an autonomous helicopter in urban areas using feature tracking," *Journal of Field Robotics*, vol. 23, no. 3-4, pp. 185–199, Mar. 2006. [Online]. Available: http://doi.wiley.com/10.1002/rob.20115

[14] S. Xia and R. Wang, "Façade Separation in Ground-Based LiDAR Point Clouds Based on Edges and Windows," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 3, pp. 1041–1052, Mar. 2019, conference Name: IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.

[15] R. Wang, J. Bach, and F. P. Ferrie, "Window detection from mobile LiDAR data," in *2011 IEEE Workshop on Applications of Computer Vision (WACV)*, Jan. 2011, pp. 58–65, iSSN: 1550-5790.

[16] M. Recky and F. Leberl, "Window detection in complex facades," in *2010 2nd European Workshop on Visual Information Processing (EUVIP)*, Jul. 2010, pp. 220–225.

[17] J.-E. Haugeard, S. Philipp-Foliguet, and F. Precioso, "Windows and facades retrieval using similarity on graph of contours," in *2009 16th IEEE International Conference on Image Processing (ICIP)*, Nov. 2009, pp. 269–272, iSSN: 2381-8549.

[18] T. H. Yuan, F. H. Hashim, W. M. D. W. Zaki, and A. B. Huddin, "An automated 3D scanning algorithm using depth cameras for door detection," in *2015 International Electronics Symposium (IES)*, Sep. 2015, pp. 58–61.

[19] S. Meyer Zu Borgsen, M. Schöpfer, L. Ziegler, and S. Wachsmuth, "Automated Door Detection with a 3D-Sensor," in *2014 Canadian Conference on Computer and Robot Vision*, May 2014, pp. 276–282.

[20] D. Anguelov, D. Koller, E. Parker, and S. Thrun, "Detecting and modeling doors with mobile robots," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 4, Apr. 2004, pp. 3777–3784 Vol.4, iSSN: 1050-4729.

[21] W. Shi and J. Samarabandu, "Investigating the Performance of Corridor and Door Detection Algorithms in Different Environments," in *2006 International Conference on Information and Automation*, Dec. 2006, pp. 206–211, iSSN: 2151-1810.

[22] J. Hensler, M. Blaich, and O. Bittel, "Real-Time Door Detection Based on AdaBoost Learning Algorithm," vol. 82, Nov. 2010, pp. 61–73.

[23] X. Yang and Y. Tian, "Robust door detection in unfamiliar environments by combining edge and corner features," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, Jun. 2010, pp. 57–64, iSSN: 2160-7516.

[24] M. ElKaissi, M. Elgamel, M. Bayoumi, and B. Zavidovique, "SEDLRF: A New Door Detection System for Topological Maps," in *2006 International Workshop on Computer Architecture for Machine Perception and Sensing*, Aug. 2006, pp. 75–80.

[25] R. Sekkal, F. Pasteau, M. Babel, B. Brun, and I. Leplumey, "Simple monocular door detection and tracking," in *2013 IEEE International Conference on Image Processing*, Sep. 2013, pp. 3929–3933, iSSN: 2381-8549.

[26] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, Apr. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7762111/

[27] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE, May 2017, pp. 5774–5781. [Online]. Available: http://ieeexplore.ieee.org/document/7989679/

[28] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 46–56, Sep. 2012, conference Name: IEEE Robotics Automation Magazine.

[29] K. Schmid, P. Lutz, T. Tomić, E. Mair, and H. Hirschmüller, "Autonomous Vision-based Micro Air Vehicle for Indoor and Outdoor Navigation," *Journal of Field Robotics*, vol. 31, no. 4, pp. 537–570, 2014, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21506. [Online]. Available: http://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21506

[30] Y. Sang, Z. Cai, and Y. Wang, "An exploration strategy based on Frontier and safe corridor for indoor flight vehicle," in *2013 Chinese Automation Congress*, Nov. 2013, pp. 244–249.

[31] C. Pravitra, G. Chowdhary, and E. Johnson, "A compact exploration strategy for indoor flight vehicles," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, Dec. 2011, pp. 3572–3577, iSSN: 0743-1546.

[32] 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," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 215–222, Sep. 2017, arXiv: 1703.01416. [Online]. Available: http://arxiv.org/abs/1703.01416

[33] C. Shang, L. Cheng, Q. Yu, X. Wang, R. Peng, Y. Chen, H. Wu, and Q. Zhu, "Micro aerial vehicle autonomous flight control in tunnel environment," in *2017 9th International Conference on Modelling, Identification and Control (ICMIC)*, Jul. 2017, pp. 93–98.

[34] J. W. Starr, "Rangefinding in Fire Smoke Environments," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2015.

[35] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model Predictive Trajectory Tracking and Collision Avoidance for Reliable Outdoor Deployment of Unmanned Aerial Vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid: IEEE, Oct. 2018, pp. 6753–6760. [Online]. Available: https://ieeexplore.ieee.org/document/8594266/

[36] M. Petrlik, T. Baca, D. Hert, M. Vrba, T. Krajnik, and M. Saska, "A Robust UAV System for Operations in a Constrained Environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2169–2176, Apr. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8979150/

[37] 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)*. Atlanta, GA: IEEE, Dec. 2010, pp. 5420–5425. [Online]. Available: http://ieeexplore.ieee.org/document/5717652/

[38] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. Kyoto, Japan: IEEE, Nov. 2011, pp. 155–160. [Online]. Available: http://ieeexplore.ieee.org/document/6106777/

[39] A. Siadat, A. Kaske, S. Klausmann, M. Dufaut, and R. Husson, "An Optimized Segmentation Method for a 2D Laser-Scanner Applied to Mobile Robot Navigation," *IFAC Proceedings Volumes*, vol. 30, no. 7, pp. 149–154, Jun. 1997. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1474667017432551

[40] E. Fernández Perdomo, "Test and Evaluation of the FastSLAM Algorithm in a Mobile Robot," Dec. 2013.

[41] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," p. 16, 2006.

[42] R. R. Labbe Jr., "Kalman and Bayesian Filters in Python," p. 504, 2018. [Online]. Available: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python

CD Content

Table 1 contains names of the root directories on the attached CD.

Directory name	**Description**
thesis	the thesis in pdf format
source	source code of the implemented algorithms
videos	videos of simulations and experiments

Table 1: CD Content

List of Abbreviations

Abbreviations used in this thesis are listed in Table 2.

Abbreviation	Meaning
FOV	Field of View
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IEPF	Iterative End-Point Fit
IMU	Inertial Measurement Unit
IR	infrared
LiDAR	Light Detection and Ranging
KF	Kalman Filter
LTI	Linear Time-Invariant
LKF	Linear Kalman Filter
MBZIRC	Mohamed Bin Zayed International Robotic Challenge
MAV	Micro Aerial Vehicle
MPC	Model Predictive Control
RANSAC	Random Sample Consensus
SLAM	Simultaneous Localization and Mapping
SEF	Successive Edge Following
UAV	Unmanned Aerial Vehicle
USAR	Urban Search and Rescue

Table 2: List of abbreviations