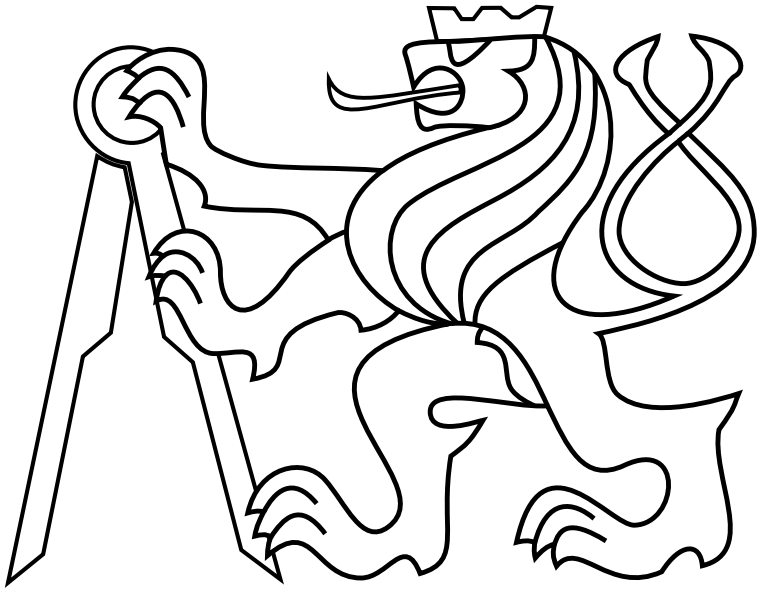
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

BACHELOR'S THESIS



Adrian Filcík

Robotic Helicopter Indoor Localization

May 2023

Department of Cybernetics

Thesis supervisor: **Ing. Jan Chudoba**

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Filcík Adrian** Personal ID number: **483477**

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

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Robotic Helicopter Indoor Localization

Bachelor's thesis title in Czech:

Lokalizace robotické helikoptéry ve vnitřním prostředí

Guidelines:

The aim of the work is analysis of methods for indor localization of small robotic helicopter. Sensors used for localization will be chosen after initial research and consultation with the work supervisor. Primarily suggested sensor is 2-D LIDAR, complemented by other sensors like range-finders or inertial sensors.
• Do a research of corresponding works.
• Design and implement a method for helicopter localization.
• Perform experiments to evaluate the performance and precision of implemented method in the simulator or with appropriate hardware in laboratory conditions.

Bibliography / sources:

[1] Fisher, Robert & Konolige, Kurt.. Handbook of Robotics Chapter 22-Range Sensors, (2008).
[2] Nascimento Tiago P., Saska Martin - Position and attitude control of multi-rotor aerial vehicles: A survey - 2019.
[3] Sobreira, Héber, et al. "Map-matching algorithms for robot self-localization: a comparison between perfect match, iterative closest point and normal distributions transform." Journal of Intelligent & Robotic Systems 93.3-4 (2019).

Name and workplace of bachelor's thesis supervisor:

Ing. Jan Chudoba Intelligent and Mobile Robotics CIIRC

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

Date of bachelor's thesis assignment: **24.02.2022** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **30.09.2023**

Ing. Jan Chudoba	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor'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 bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Author statement for undergraduate thesis:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

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

Signature

Acknowledgements

First of all, I would like to thank my supervisor Ing. Jan Chudoba for his valuable help, Mgr. Barobra Čalkovská for her patient assistance and support and Mrs. Jana Zichová for her kindness. Furthermore, I would like to thank all the open source developers whose software I used in my work, because without them this thesis would not have been possible, especially I would like to thank the team of the Multi-robot Systems Group. Finally, I would like to thank my friends and family for their support. There are more people I would like to thank, but I cannot name them all.

Abstract

This thesis presents a localization system for a small robotic helicopter in an indoor environment. The system consists of helicopter hardware, a 2D LiDAR sensor, and localization software based on the ICP algorithm. To verify the implementation, experiments were conducted in a realistic Gazebo 3D simulator. The performance and accuracy of the presented localization system were compared with the state-of-the-art localization system.

Keywords

UAV, helicopter, drone, localization, LiDAR, SLAM, ICP

Abstrakt

Tato práce představuje lokalizační systém pro malý robotický vrtulník ve vnitřním prostředí. Systém se skládá z hardwaru vrtulníku, 2D LiDARového senzoru a lokalizačního softwaru založeného na algoritmu ICP. Pro ověření implementace byly provedeny experimenty v realistickém 3D simulátoru Gazebo. Výkon a přesnost představeného lokalizačního systému byly porovnány s moderním lokalizačním systémem.

Klíčová slova

UAV, helikoptéra, dron, lokalizace, LiDAR, SLAM, ICP

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For the purposes of this thesis, a small robotic helicopter is an Unmanned Aerial Vehicle (UAV), which is an aircraft that does not carry a pilot. It can be remotely piloted, or it can be autonomous. It can be a single-, double-, or multi-rotor helicopter. The rotors provide the necessary thrust to keep the helicopter in the air and allow it to hover.

In this thesis, the small robotic helicopter cannot be generally defined as a Micro Aerial Vehicle (MAV) because MAVs have been defined [2] as a vehicle with any dimension less than 6 inches (approximately 15 cm) and a mass less than 100 grams. The UAV frames used in this thesis exceed these specifications.

Localization is the process of determining the position and orientation of a selected object in a given reference frame.

This thesis considers an indoor environment to be an environment that does not allow the use of the Global Positioning System (GPS) or other Global Navigation Satellite Systems (GNSS). Therefore, it is also referred to as a GPS-denied or GNSS-denied environment.

The applications of drones are vast. They can be sent into environments where a person cannot go for various reasons. UAVs can also more easily reach places that unmanned ground vehicles (UGV) cannot, making them useful for search and rescue tasks in otherwise impenetrable or dangerous terrain [3], [4], [5]. Additionally, UAVs can facilitate the work of emergency responders, such as locating and extinguishing fires [6], [7]. They are also helpful for law enforcement [8], [9]. Another area where UAVs can be used is in surveying and mapping unknown and inaccessible areas [10], [11]. They are also used to inspect buildings and other structures [12], [13].

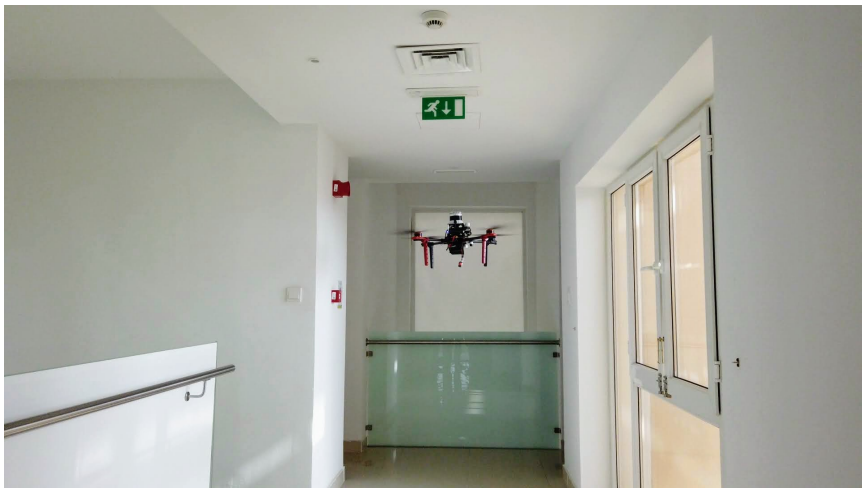


Figure 1.1: UAV flying in an indoor environment. Image originally published by the Multi-robot Systems Group in [1].

As mentioned above, UAVs are also used in indoor environments (Figure 1.1) where GNSS does not work. The most common indoor applications include search and rescue missions, exploration of unknown confined spaces, and building inspections. For these and other reasons, it is useful to ensure reliable navigation and safe, collision-free movement in these indoor environments. It is also possible to apply the same approach to other areas with obstacles and insufficient GNSS signal coverage.

1.1 State of the art

UAV localization in indoor environments remains an active area of research. The following works have addressed this topic.

Visual localization of UAVs in the indoor environment using cameras was the subject of [14]. Problematic in visual localization is the performance in both dark and bright lighting conditions. Therefore, in this work, a camera was selected that can operate in a wide range of lighting conditions.

In [5], two-dimensional (2D) Light Detection And Ranging (LiDAR) sensor was used for indoor UAV localization. The UAVs were operated in underground tunnels. The localization was performed only in the 2D plane because the horizontal cross-section of the tunnels did not change significantly.

Three-dimensional (3D) LiDAR sensors are also used for indoor localization. In [15], such a 3D LiDAR system is used for simultaneous localization and mapping. In addition, a fusion of sensor data from a downward-facing one-dimensional (1D) LiDAR sensor, gyroscope, accelerometer, and barometer is used to increase the robustness of the localization system.

Several localization systems are used for localization. For visual localization, these include simultaneous localization and mapping (SLAM) systems such as Fast Semi-direct Monocular Visual Odometry (SVO) [16], Direct Sparse Odometry (DSO) [17], or Oriented FAST and rotated BRIEF feature-based simultaneous localization and mapping 2 (ORB-SLAM2) [18].

The most commonly used 2D LiDAR localization systems include Hector SLAM [19], Gmapping [20], [21], or Cartographer [22]. The Cartographer localization system can also be used for localization with 3D LiDAR sensors. Other localization systems using 3D LiDAR sensors include Lidar Odometry and Mapping (LOAM) [23] or its modified version, Fast LOAM (F-LOAM) [24].

1.2 Outline

This thesis is primarily concerned with solving the problem of localizing a UAV, in our case, a small robotic helicopter, in an indoor environment. Specifically, it aims to achieve the following key objectives:

1. Conduct research on related work.
2. Design and implement a helicopter localization method.
3. Perform experiments to evaluate the performance and accuracy of the implemented method in the simulator.

Objective 1, research on related work, has already been partially covered in the chapter 1. Further research on UAV platforms, sensors, and known localization methods is done in the chapter 2.

Objective 2, the design and implementation of a helicopter localization method, is covered in the chapter 3.

And objective 3 is covered in the chapter 4, where experiments are performed to evaluate the performance and accuracy of the implemented method.

Chapter 2

Approach

The purpose of this chapter is to present the problem statement and describe the approach to the solution.

2.1 Problem statement

This thesis aims to research and design a method for the localization of a small helicopter in an indoor environment. Multiple aspects needed to be addressed in the design process of the solution. These aspects were:

1. Selection of an appropriate development platform (a helicopter and appropriate software needed for its function).
2. Selection of appropriate sensors.
3. Design and implementation of a suitable algorithm for localization.

Further requirements were derived according to the assignment of this thesis.

For the selection of a development platform, the requirements were divided into software and hardware requirements. The software requirement was to find suitable, actively developed, and supported software to facilitate control of the helicopter hardware, support a realistic simulator, and allow implementation of the required localization algorithm. The hardware requirement was to find a helicopter small enough to fly indoors but with sufficient lift capacity and battery life.

The requirements for the sensors were sufficient measurement speed and accuracy. Furthermore, due to the requirements for the development platform, low weight, low power consumption, and low computational complexity in the pre-processing of the sensor data were added.

The requirements for the localization algorithm were accuracy and speed. In addition, due to the requirements of the development platform, low enough computational complexity was added to make the localization sufficiently energy efficient. Finally, compatibility with the sensor data is required.

2.2 Selection of a development platform

This section focuses on selecting an appropriate development platform. The selection is subject to the requirements in the section 2.1. The goal is to select a suitable software and hardware solution to operate and control a small robotic helicopter and to enable the implementation of a localization algorithm.

2.2.1 Software platform

First, it was necessary to select a suitable software framework. As stated in the section 2.1, the software solution had to enable control of a small helicopter, be actively developed and supported, support a realistic simulator, and allow implementation of the required localization algorithm.

The following software platforms were considered:

- The Robot Operating System (ROS), an open-source robotics middleware platform formerly developed by Willow Garage[1], now developed by Open Robotics[2], as described in [25] and [26].
- The Carnegie Mellon Robot Navigation Toolkit (CARMEN), an open-source mobile robot control software suite developed at Carnegie Mellon University and funded by DARPA's MARS Program. CARMEN has been described in [27].
- The Player Project, formerly the Player/Stage Project, an open-source software project that provides the Player robot server and the Stage 2D simulator, introduced in [28] and described in [29] and [30].
- The Mobile Robot Programming Toolkit (MRPT), a package of libraries and applications for robotics research, as detailed in [31].
- The Microsoft Robotics Developer Studio (MRDS), a robotics suite for robot control and simulation. See [32] for an introduction to MRDS.
- The NVIDIA Isaac platform, a software platform for developing and deploying robots, as introduced in [33].

[1] `http://www.willowgarage.com/`
[2] `http://www.openrobotics.org/`

- The Open Robot Technology Middleware, implemented by the National Institute of Advanced Industrial Science and Technology (OpenRTM-aist), an open source and open architecture implementation of the Robotics Technology Middleware (RT-middleware) and the Object Management Group's (OMG) Robotics Technology Components (RT-Components). RT-middleware is described in [34] and OpenRTM-aist in [35].

- The Open Platform for Robotic Services (OPRoS), an open-source component-based platform that includes a robot control framework, a server, and a test and verification tool, developed by the Korea Association of Robot Industry. OPRoS was introduced in [36].

- The Open Robot Control Software (Orocos), a software framework consisting of C++ libraries for machine and robot control. Its two main components are the Orocos Real-Time Toolkit (RTT) and the Orocos Component Library (OCL). More about the Orocos Project can be found in [37], [38], and [39]. For more information about the Orocos RTT, see [40].

- The Orca, an open-source software framework designed for mobile robotics developed by the KTH Royal Institute of Technology. It evolved from the Orocos Project. It was first used in [41], formally introduced in [42], and further described in [43] and [44].

- The Yet Another Robot Platform (YARP), an open-source robotics middleware, as described in [45].

- The Mission Oriented Operating Suite (MOOS) and the MOOS Interval Programming Helm (MOOS-IvP), software for mobile robotics, primarily focused on autonomous marine vehicle research. MOOS was introduced in [46] and further described in [47], and MOOS-IvP was introduced in [48].

- The Robot Construction Kit (Rock) is a software framework for the development of robotic systems. It is based on the Orocos RTT. The description of Rock can be found in [49] and [50].

Software platforms such as CARMEN, MRDS, OPRoS, or Orca were excluded because they were no longer actively maintained at the time of writing and therefore did not support some of the newer sensors, other hardware, and software standards.

Software platforms such as MRPT or NVIDIA Isaac were excluded from the selection because they function more as toolkits and would not facilitate the implementation of our own localization algorithm. Both provide a set of ready-to-use applications and tools. MRPT can be used alone or in conjunction with ROS or ROS 2. NVIDIA Isaac is based on ROS 2 and is primarily designed to accelerate some robotics algorithms utilizing NVIDIA's proprietary hardware.

To successfully operate and control a small robotic helicopter, the selected software platforms had to be able to communicate with commonly used UAV flight controllers and their autopilot firmware, such as PX4[3] as introduced in [51], or ArduPilot[4]. The MAVLink protocol[5] (described in [52]) is most commonly used for this purpose. Therefore, software platforms without direct support for the MAVLink protocol were excluded. These platforms were the Player Project, YARP, Orocos, and ROCK. All of these platforms can also communicate with ROS through a translation layer.

In [53] and [54], the possibilities of using OpenRTM-aist for UAV control are described. In these papers, a DroneKit-Python[6] application is used as an RT-Component to communicate with the flight controller using the MAVLink protocol. However, no source code is provided for these solutions.

The iPX4 package[7] and the pMavlink translation layer[8] can bridge between MOOS and the PX4 flight controller. They bring MAVLink support to MOOS. However, these solutions lack proper documentation.

The last two software platforms considered were ROS and ROS 2. Although Open Robotics develops both, they are not interoperable.

ROS 1 is still actively maintained. The MAVROS package[9] provides the link between ROS 1 and the various autopilots using the MAVLink protocol. It has broad support for various sensors and other hardware. It is also well documented. There are also many libraries and packages that extend the capabilities of ROS 1, such as the Multi-robot Systems Group UAV system (MRS UAV system) described in [55].

ROS 2 is newer and uses the Data Distribution Service (DDS) as middleware. It is under active development. Bridging between ROS 2 and various autopilots can be achieved using the XRCE-DDS protocol[10], provided, for example, by the eProsima Micro XRCE-DDS library[11] or by MAVROS. It is also well documented. At the time of writing, however, it lacks an extension of the type that is the MRS UAV system for ROS 1.

Ultimately, a combination of the MRS UAV system and ROS 1 was selected.

2.2.2 Robot Operating System

ROS 1 is structured into nodes. Each node is a separate program. All nodes register with the ROS master node. Nodes can communicate with each other using messages or

[3] https://px4.io/
[4] http://www.ardupilot.org/
[5] https://mavlink.io/
[6] https://dronekit.io/
[7] https://github.com/mission-systems-pty-ltd/iPX4
[8] https://github.com/mission-systems-pty-ltd/pMavlink
[9] https://wiki.ros.org/mavros
[10] https://www.omg.org/spec/DDS-XRCE/1.0/About-DDS-XRCE
[11] https://micro-xrce-dds.docs.eprosima.com/en/latest/

services. ROS 1 has interfaces in both Python and C++. We will only focus on the C++ interface here.

Topics, uniquely named buses, are used for message transport. Each node can send and receive messages to and from a selected topic using the classes provided. The class for sending messages is called the Publisher, and the class for receiving messages is called the Subscriber. Topics are typically used for data streams of a particular type, such as a continuous video stream from a camera.

Services are used for direct node-to-node communication. A node can advertise a service using the advertiseService method of the NodeHandle class and call a service using the call method of the ServiceClient class. There is exactly one response to each call. Therefore, services are often used for specific actions, such as retrieving a single image from a camera.

There are also nodelets. A nodelet is a separate program similar to a node. However, unlike nodes, multiple nodelets can be combined into a single node while maintaining separate namespaces, allowing zero-copy pointer passing between publish and subscribe calls between those nodelets.

2.2.3 MRS UAV system

The MRS UAV system is an open-source UAV platform. It is built using ROS Noetic and is intended to run entirely on board. The platform is actively developed, maintained, and well-documented[12].

It consists of several components, of particular importance are the MRS Gazebo simulation[13] and the MRS UAV system core[14]. The simulation environment is based on the realistic Gazebo simulator. The MRS UAV core consists of only the most essential ROS packages required for UAV operation. For our purposes, the ROS package MRS UAV Odometry is particularly important.

MRS UAV Odometry allows estimation of the UAV state using sensor fusion of on-board sensors. Sensor fusion is primarily accomplished by using a bank of Kalman filters [56] to estimate multiple UAV state hypotheses simultaneously. More details about the system architecture can be found in [55].

2.2.4 Hardware platform

Next, a suitable UAV platform had to be selected. The hardware requirements from the section 2.1 were to select an aircraft small enough to fly indoors but with enough lift

[12] `https://ctu-mrs.github.io/`
[13] `https://github.com/ctu-mrs/simulation`
[14] `https://github.com/ctu-mrs/uav_core`

capacity to carry the flight controller, necessary sensors, and other hardware and good battery life to allow extended operation.

According to the above requirements, the selection was made among the smallest helicopters for which the MRS UAV system is pre-configured. The [57], [58], and [59] were used as guides. DJI Flame Wheel F450 and Holybro X500 helicopters were considered.

The Holybro X500 has a 500 mm frame. According to [57], [59], the MRS Group's Holybro X500 setup is equipped with four T-Motor MN3510-13 700 kV motors, 13-inch carbon fiber propellers, and one or two 4S 6750 mAh lithium polymer batteries, allowing a flight time of approximately 20 minutes.

The DJI Flame Wheel F450 has a 450 mm frame. According to [57], [59], the MRS Group's DJI Flame Wheel F450 setup is equipped with four 2312 920 kV motors, 9.4-inch plastic propellers, and a 4S 6750 mAh lithium polymer battery, allowing for approximately 10 to 15 minutes of flight time.

2.2.5 DJI Flame Wheel F450

In the end, the DJI Flame Wheel F450 quadcopter was chosen because it is the smaller of the two helicopters but has a large enough lift capacity and battery life. In addition to the 2312 920 kV motors, 9.4-inch plastic propellers, and a 4S 6750 mAh lithium polymer battery, it was also equipped with the Pixhawk 4 flight controller[15] loaded with MRS Group's custom PX4 autopilot firmware[16] and an Intel NUC computer kit[17].

2.3 Selection of sensors

For this thesis, it was also necessary to select suitable types of sensors for the localization that could be carried by the small robotic helicopter, the DJI Flame Wheel F450. Therefore, according to the section 2.1, the following criteria were considered when selecting sensors: low weight, low power consumption, and accuracy. Speed was not considered a factor in the selection of sensor types, as sufficiently fast sensors are currently available for each sensor category.

Our helicopter has to operate and localize in an indoor environment. However, traditional UAV localization methods, such as GNSS using a magnetometer, do not work there. In general, the radio signal from GNSS satellites does not penetrate the walls of buildings, and the use of various ferromagnetic materials and electrical wires makes the magnetometer's determination of the magnetic north pole less accurate.

[15] `https://docs.px4.io/main/en/flight_controller/pixhawk4.html`

[16] `https://github.com/ctu-mrs/px4_firmware`

[17] `https://www.intel.com/content/www/us/en/products/sku/188808/intel-nuc-10-performance-kit-nuc10i7fnk/specifications.html`

Obstacles are expected in indoor environments. Sensors designed to measure the characteristics of the immediate environment or the internal state of the UAV are not capable of tracking obstacles in the environment. Therefore, they are not suitable as primary localization sensors. They can be used as a complement to other localization sensors to refine the data. Such sensors include a barometer or an inertial measurement unit (IMU). However, the IMU is still widely used because it is essential for UAV stabilization.

That leaves object detection and measurement sensors. Object sensors can be categorized in many ways. They can be categorized by the physical property being measured, whether they are passive or active (whether they require external power), or by the nature of their output. For the sake of simplicity, we have decided to divide the sensors into several categories:

1. Digital cameras.
2. Ranging sensors.
3. Depth cameras.

These categories of sensors are sometimes accompanied by specific examples of the sensors most commonly used by the MRS group as listed in [57].

2.3.1 Digital cameras

Digital cameras are often used in drones because, among other things, they give pilots a first-person view.

The two fundamental types of camera-based localization algorithms are optical flow-based and feature-based algorithms. Optical flow localization is generally not very accurate because it only tracks the velocities of objects in the camera's field of view (FOV) and is thus prone to drift.

Feature-based algorithms can be divided into those that use pre-positioned markers in the environment (like AprilTag [60], [61]) and those that search for significant features of objects captured by the camera and compare them between successive frames [62]. Since we assume that the indoor environment is potentially unknown, the marker-based method is not applicable.

Digital cameras have the advantage that they are mostly passive sensors and therefore consume relatively little power. However, this low power consumption is offset by the increased power consumption of the computing unit due to the high computational complexity of most feature-based algorithms.

Cameras also typically have a limited FOV, which often makes using a multi-camera system desirable (cite). However, this increases both the power consumption and the overall weight.

2.3.2 Ranging sensors

Ranging sensors are sensors that do not require physical contact with the target. A ranging sensor typically measures the distance between the sensor and the objects.

RaDAR

The Radio Detection And Ranging (RaDAR) sensor is a ranging sensor that uses radio waves to determine the distance, angle, and even speed of objects (using the Doppler effect) relative to the measurement location. The advantage of RaDARs is their relatively high accuracy. The disadvantage of RaDARs is their relatively high power consumption and weight.

SoNAR

The Sound Navigation And Ranging (SoNAR) sensor is a sensor that works on the same principle as RaDAR but uses sound instead of radio waves to measure distance. SoNAR sensors for UAVs most commonly use ultrasound. These ultrasonic sensors typically measure in one direction only, are less accurate, and have a relatively limited range. Therefore, they are not suitable for localization. However, they are useful for measuring height from the ground and collision avoidance.

LiDAR

The LiDAR is a sensor that measures distance by illuminating objects with a laser and measuring the time it takes for the reflected light to return. There are 1D, 2D and 3D LiDARs.

1D LiDARs are commonly used in UAVs for ground range or collision avoidance. They are not suitable for localization. 2D LiDARs are most commonly used in UAVs to determine position and orientation within a plane. And 3D LiDARs are most commonly used in UAVs to determine position and orientation in 3D space.

Both 2D and 3D LiDARs are suitable for localization purposes, but 2D LiDARs are relatively lightweight and have low power consumption, while 3D LiDARs are heavier and more expensive.

Depth camera

Depth cameras are a combination of cameras and ranging sensors, most commonly a combination of cameras and LiDAR sensors. They are disadvantaged by the limited FOV of the camera and the limited range of the LiDAR. They are a relatively good choice for indoor localization but they often have higher power consumption than some 2D LiDARs.

Evaluated criteria	Multi-camera	2D RaDAR	2D LiDAR	3D LiDAR	Depth camera
Weight	0	-	+	-	++
Power consumption	+	-	++	0	+
Accuracy	-	0	+	++	0

Table 2.1: Suitable sensors for indoor localization are rated as bad (-), neutral (0), good (+), or best (++) for each criterion.

Based on the research, applicable sensor types for indoor localization were selected. These sensor types were then broadly compared to each other based on the given criteria: weight, power consumption, and accuracy. The orientation results are presented in a straightforward Table 2.1.

Finally, IMU, 1D, and 2D LiDAR sensors were selected, namely Garmin LIDAR-Lite v3 and SLAMTEC RPLIDAR-A3. Garmin LIDAR-Lite v3 for height measurement from the ground and SLAMTEC RPLIDAR-A3 as the primary localization sensor.

The IMU is already included in the non-equipped version of the helicopter, as it is already part of the Pixhawk 4 module. A barometric sensor can be added to account for altitude jumps when flying over uneven ground.

2.3.3 SLAMTEC RPLIDAR-A3

The selected LiDAR SLAMTEC RPLIDAR-A3[18] sensor weighs 190 grams. Its angular range is a full 360 degrees. Its minimum range is 0.2 meters, while the maximum is 25 meters. According to the manufacturer, it can detect a dark object up to 10 meters away. Its maximum sampling rate is given as 16000 times per second, and the scanning frequency can be set between 10 and 20 Hz. Its maximum angular resolution is 0.225 degrees.

We operate the LiDAR sensor at a scan rate of 20 Hz and get 720 samples per scan, which translates to an angular resolution of 0.5 degrees. We also limit the maximum range to 14 meters due to significant measurement errors at longer distances and to maintain compatibility with older sensors such as the SLAMTEC RPLIDAR-A2. It was mounted horizontally on the top of our helicopter. Therefore it allows localization in the horizontal plane (2D position and 1D orientation).

The used SLAMTEC RPLIDAR-A3 LiDAR scanner does not provide intensity value measurements. Intensities can otherwise be used to make point-set registration algorithms more robust by using point feature matching [63].

Its driver in ROS also does not report the time increment between each range measurement. This time increment can otherwise be used in interpolating the position of 3D points when the scanner is moving. The ROS package called laser_geometry provides a

[18] `https://www.slamtec.com/en/Lidar/A3`

C++ class `LaserProjection` whose method `transformLaserScanToPointCloud` provides this functionality.

2.4 Design of an appropriate localization algorithm

A suitable localization algorithm had to be devised. According to part 2.1, the following criteria were considered in the selection of sensors: accuracy, speed, sufficiently low computational complexity, and compatibility with sensor data.

IMU data can be used to estimate translation and rotation displacement. However, IMU data tends to drift and must be compensated. Therefore, an additional localization sensor is required. Our localization sensor of choice is a 2D LiDAR. Thus the sensor data are planar scans that could be represented as a 2D point cloud. In order to account for drift, the mapping must be used. Each new scan is incrementally matched to the map. The map is then updated by storing the matched scans in the map.

There are many localization algorithms that work with ranging sensor data. They can be divided into four main categories: those based on Iterative Closest Point (ICP), those based on Lidar Odometry and Mapping (LOAM), and those based on Normal Distributions Transform (NDT). All of these algorithms are used to align two point clouds and determine the resulting transformation between them.

2.4.1 Initial alignment estimation

Many scan-matching algorithms require a good estimate of the initial alignment between the LiDAR scan and the map because they tend to get stuck in the local optimum.

Feature-based registration[19] can be used to obtain an initial estimate. Typically, keypoints need to be selected first for such registration. A keypoint is generally a point of interest that has a distinctive property.

Features are then extracted from the keypoints. Feature descriptors are used for this purpose. Feature descriptors are generally mathematical functions that analyze and quantify properties around given keypoints (such as various geometric patterns) and assemble them into a feature vector. Feature vectors are extracted from both scans (scan and map). Then, the correspondences between the feature vectors from the two sets of feature vectors can be estimated using different algorithms. The transformation between the scans (scan and map) is then estimated based on the correspondences. This transformation can then be used as an estimate of the initial alignment.

In our work, however, this approach cannot be used because, for feature extraction, the data from both scans (scan and map) must be of the same type and have the same number of dimensions.

[19] `https://pcl.readthedocs.io/en/latest/registration_api.html`

2.4.2 NDT-based algorithms

The first category is NDT-based algorithms. NDT was first introduced in [64]. The original NDT worked only with 2D data, but later a version working with 3D data was proposed and introduced in [65].

NDT essentially works as follows: Points from the first point cloud are divided into regular cells using a grid. Then, the distribution of points in each cell is estimated using a normal distribution. An optimization problem is then solved to find the transformation that, when applied to the second point cloud, maximizes the sum of the point likelihoods from the second point cloud over the distributions obtained from the first point cloud. For this optimization, iterative optimization algorithms are often used, and thus NDT can also be accelerated by using multithreading. Such a solution was used in [66].

The grid size setting is essential for NDT. If the grid size is set unsuitably, NDT may perform poorly. However, the appropriate grid size is primarily determined by the geometric properties of the perceived environment and thus cannot be reliably determined without prior knowledge of the environment. Therefore, NDT cannot be used reliably for our purposes.

2.4.3 LOAM-based algorithms

LOAM is an algorithm for simultaneous localization and mapping (SLAM). LOAM was introduced in [23]. LOAM-based algorithms include:

- Advanced implementation of LOAM (A-LOAM) introduced in [67].
- Lightweight and Ground-Optimized LiDAR Odometry and Mapping (LeGO-LOAM) introduced in [68].
- Fast LiDAR Odometry and Mapping (F-LOAM) introduced in [24].
- Optimized-SC-F-LOAM, Tightly Coupled 3D LiDAR Inertial Odometry and Mapping (LIO-Mapping) introduced in [69].
- And Tightly-coupled LiDAR Inertial Odometry via Smoothing and Mapping (LIO-SAM) introduced in [70].

All these algorithms work only with 3D LiDAR data because they extract features from geometric structures in the scene and thus cannot be used for our purposes.

2.4.4 ICP-based algorithms

ICP is an iterative optimization algorithm that seeks to find the transformation that, when applied to one of the point clouds, minimizes the sum of the least squares between the given point clouds. ICP takes one point cloud as the fixed target and the other point cloud as the source to which the estimated transformation is applied. A map must be constructed and used as the target point cloud to account for potential drift between scans. ICP was first introduced in [71] and [72].

ICP-based algorithms include the following:

- ICP point-to-plane introduced in [71].
- Generalized-ICP (GICP) introduced in [73].
- Fast GICP, Voxelized GICP (VGICP), Fast VGICP, Fast VGICP CUDA introduced in [74], and many others.

All of these algorithms use surface normal estimation to approximate the neighborhood of each point in the point cloud using a plane. As a result, the distances to these approximated surfaces can be measured instead of just the distances between points. Generally, this improves the speed of convergence.

However, 2D LiDAR data is planar, so fitting a plane to this data to obtain the normals results in all normals being perpendicular to the plane where the scan was taken. There is also no guarantee that the resulting map will be sufficiently dimensional (non-planar). Therefore, these variants of ICP are not suitable for our purposes.

Finally, it was decided that the localization algorithm would be based on the ICP algorithm.

2.4.5 Iterative Closest Point

Although ICP has already been briefly introduced in section 2.4.4, the pseudocode is provided and further described in the Algorithm 1 for convenience.

The input to the ICP is a source point cloud, and a target point cloud. Optional input is the initial estimate of rotation and translation. If both point clouds are 2D, the rotation matrix is of the form $R \in \mathbb{R}^{2\times2}$ and the translation vector is of the form $t \in \mathbb{R}^{2\times1}$. If at least one point cloud is 3D, the rotation matrix is of the form $R \in \mathbb{R}^{3\times3}$ and the translation vector is of the form $t \in \mathbb{R}^{3\times1}$.

N is a constant whose value corresponds to the number of elements of the source point cloud. ϵ represents the current mean squared error of the distances and MSE_Threshold is a threshold that represents the maximum acceptable mean squared error. The variable named Iterations has a value corresponding to the number of already completed cycles, and

Algorithm 1 Iterative Closest Point

Input:
Source point cloud: S
Target point cloud: T
Initial guess of rotation and translation: $\{R,t\}$
A constant equal to the number of source point cloud elements: N
Maximum acceptable mean squared error: MSE_Threshold
Maximum number of iterations: Maximum_Iterations
Output:
The transformation matrix estimate: Transformation

$\epsilon \leftarrow \infty$ ▷ Variables are initialized.
Iteration $\leftarrow 0$
for $n = 1 \rightarrow N$ **do** ▷ The initial guess is applied.
    $S_n = R \cdot S_n + t$
end for
Transformation $\leftarrow \begin{bmatrix} R & t \\ \vec{0}^T & 1 \end{bmatrix}$
while $\epsilon >$ MSE_Threshold **and** Iteration $<$ Maximum_Iterations **do**
    $C \leftarrow$ **CorrespondenceEstimation**(S,T)
    $\{R,t\} \leftarrow$ **TransformationEstimation**(C)
    for $n = 1 \rightarrow N$ **do**
        $S_n = R \cdot S_n + t$
    end for
    Transformation $\leftarrow \begin{bmatrix} R & t \\ \vec{0}^T & 1 \end{bmatrix} \cdot$ Transformation
    $\epsilon \leftarrow \frac{1}{N} \sum_{n=1}^{N} \|T_n - S_n\|^2$
    Iteration++
end while

Maximum_Iterations is a threshold that denotes the number of maximum iterations. If the ϵ value exceeds the MSE_Threshold or the Iteration value exceeds the Maximum_Iterations threshold, the algorithm terminates. The output is the resulting transformation estimate in the form of a transformation matrix. Successive transformations are accumulated in this transformation matrix every cycle.

For each point in the source point cloud S, the function **CorrespondenceEstimation**(S,T) assigns the nearest point in the target point cloud T. These pairs are then stored in the correspondence list C. In general, several types of search methods can be used for this nearest neighbor search. Brute-force nearest neighbor search is generally not used because of the poor operational complexity of the algorithm. Therefore, nearest neighbor search methods such as k-d tree (introduced in [75]) or Octree (introduced in [76]) are used. For even faster searches, methods that only estimate nearest neighbors are also used. Such methods include algorithms from the Fast Library for Approximate Nearest Neighbors (FLANN) as described in [77].

The **TransformationEstimation**(C) function estimates the transformation required to align the source point cloud with the target point cloud.In this pseudocode, the transformation is returned in the form of a rotation matrix and a translation vector. The function solves the problem of minimizing a point-to-point distance metric to estimate a transformation, the formulation of this minimization problem follows:

$$\{R,t\} = \arg\min_{\{R,t\}} \frac{1}{N} \sum_{n=1}^{N} \|T_n - R \cdot S_n - t\|^2 \tag{2.1}$$

Chapter 3

Implementation

This chapter lists used libraries, discusses implementation details and limits of used methods, and describes some simplifications and innovative changes.

3.1 Selection of software frameworks

The code for this thesis was written as a ROS nodelet in C++. For the work with point clouds and simplifying the implementation of ICP, it was necessary to choose a suitable library that works well with ROS.

Point Cloud Library (PCL)[1] is an open-source library that specializes in image and point cloud processing. It provides both base implementations and state-of-the-art implementations of algorithms used in filtering, feature estimation, surface reconstruction, registration, model fitting, and segmentation. A wrapper for implementation in ROS is also provided. The PCL was introduced in [78].

The C(canonical) Scan Matcher (CSM)[2] is an open-source library written in C. It provides an implementation of an ICP variant using a point-to-line metric. The CSM library has been created for the purposes of [79]. The package called laser_scan_matcher[3] builds upon the CSM and integrates well inside ROS. It also uses PCL for point cloud representation. It can be run as a ROS node or nodelet.

Another open-source library that implements ICP is libpointmatcher[4]. It uses YAML[5] files for its point-matching pipeline, allowing developers to change parameters without recompiling the code. The libpointmatcher was introduced in [80] and [81]. The libpointmatcher library was first introduced in [80] and was further evaluated in [81]. The module

[1] `https://pointclouds.org/about/`
[2] `https://github.com/AndreaCensi/csm`
[3] `https://github.com/CCNYRoboticsLab/scan_tools`
[4] `https://github.com/ethz-asl/libpointmatcher`
[5] `https://yaml.org/`

Auto-tuned ICP (AICP)[6] builds upon the libpointmatcher library. It provides laser-based localization and mapping functions and includes a wrapper for ROS. The AICP was introduced in [82] and further used in [83] and [84].

Ultimately, the PCL was chosen because it provides base classes for point cloud registration that can be easily edited for each project. The Eigen[7] library was chosen for the work with vectors and matrices. Eigen is an open-source C++ template library for linear algebra, and it integrates well into ROS and PCL. MRS libraries[8] were used to simplify the work with ROS, primarily for converting representations of 3D orientation. They are a part of the MRS UAV system.

3.2 Filtering

Sometimes the data can be significantly affected by measurement errors. Therefore, it is often necessary to filter the data before further processing of point clouds. In our case, these errors manifest themselves as noise in the distance measurements and are most often modeled using a normal distribution with zero mean and a standard deviation of approximately 0.01 m or 1 cm. Therefore, a filter node was incorporated into our design. For this purpose, we used the ROS package laser_filters. The median filter is particularly useful for noise removal. However, our tests showed that the use of filters is not necessary on our platform and that the more significant problem is the added time overhead.

3.3 Localization algorithms

Subsequently, it was necessary to implement the localization algorithm itself. This section walks through the implementation process, then explains the specific solutions and some limitations.

3.3.1 Implementation of the ICP algorithm

The base ICP implementation in PCL uses a singular value decomposition (SVD) [85] compared to the original work of Paul J. Besl and Neil D. McKay [72] that used transformation estimation based on quaternion optimization. Paul J. Besl and Neil D. McKay also discussed in [72] the use of SVD as the transformation estimation method inside the ICP algorithm. They state that their quaternion-based algorithm is adequate for up to three-dimensional spaces but that SVD can be generalized to work in higher dimensions. This transformation estimation is provided by the `TransformationEstimationSVD` class.

[6] `https://github.com/ori-drs/aicp_mapping`
[7] `https://eigen.tuxfamily.org/`
[8] `https://ctu-mrs.github.io/docs/software/uav_core/mrs_lib/`

The SVD-based transformation estimation estimates rotation and translation, and its implementation in PCL cannot be constrained to fewer than six degrees of freedom (6DOF). This leads to problems when 2D point clouds like planar scans are combined with 3D transformations. Points in the source point cloud tend to be aligned with the target point cloud in the same plane, even if this does not correspond to the actual orientation in space during the measurement. This is solved by the class `WarpPointRigid3D` that, when used with the `TransformationEstimationLM`, allows the number of degrees of freedom to be locked to three degrees of freedom (3DOF).

3.3.2 Constrained nonlinear ICP algorithm

The class `IterativeClosestPointNonLinear` mainly differs from the base class `IterativeClosestPoint` by using a different transformation estimation class, `TransformationEstimationLM`, instead of `TransformationEstimationSVD`. The `TransformationEstimationLM` uses the Levenberg-Marquardt algorithm (LM) [85] to align given correspondences, as proposed in [86]. The LM algorithm in the `TransformationEstimationLM` class is using the `LevenbergMarquardt` class from the Eigen library.

The `WarpPointRigid3D` class can be used with `TransformationEstimationLM` to limit the rigid transformation to only 3DOF (1D rotation + 2D translation). This is done by optimizing the equation 3.1 using the transformation matrix from equation 3.2 as the transformation estimation:

$$\mathbf{T} = \arg\min_{\mathbf{T}} \frac{1}{N} \sum_{n=1}^{N} \|T_n - \mathbf{T} \cdot S_n\|^2 \tag{3.1}$$

$$\mathbf{T} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 & t_x \\ \sin(\psi) & \cos(\psi) & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

S_n is the nth point from the source point cloud and T_n the nth point from the target point cloud and N is the number of source point cloud points. The transformation matrix $\mathbf{T}$ is determined by only three parameters t_x, t_y, and ψ. t_x represents the translation along the X axis, t_y the translation along the Y axis, and ψ the rotation around the Z axis in radians.

In our case, this allows the transformed scans (point clouds), among other things, to remain at the correct height and not be squeezed into the same plane as the reference point cloud, as is the case with `IterativeClosestPoint` and its SVD-based backend. The `TransformationEstimationLM` also allows changing the default distance function. It can be changed by creating a subclass of `TransformationEstimationLM` and overriding the `computeDistance` method.

3.3.3 Transformation estimation with custom distance metric

We propose a new distance function to be minimized to speed up the alignment of the given correspondences:

$$d(S'_n, T_n) = \left\| \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \cdot (T_n - S'_n) \right\|_2 \tag{3.3}$$

$$\mathbf{T} = \arg\min_{\mathbf{T}} \frac{1}{N} \sum_{n=1}^{N} d(\mathbf{T} \cdot S_n, T_n)^2 \tag{3.4}$$

The distance function is the equation 3.3 and the resulting minimization problem is in the equation 3.4. S_n is the nth point from the source point cloud and T_n the nth point from the target point cloud. $\mathbf{T}$ is the estimated transformation matrix. S'_n denotes the nth point from the transformed source point cloud, thus $S'_n = \mathbf{T} \cdot S_n$. And N is the number of source point cloud points.

Because our input point clouds are planar, our distance function omits the Z element of the Cartesian coordinate system. This method does not change the metric used to estimate correspondences and, therefore, should not lead to more outliers where points are close together in the XY plane and far apart on the Z axis. If outliers are detected, they can be removed by using outlier rejection classes from PCL, such as `CorrespondenceRejectorDistance` and `CorrespondenceRejectorMedianDistance`.

This change should be equivalent to using the point-to-line metric if all lines were perpendicular to the XY plane in the Cartesian coordinate system. This change should also yield similar results to using the point-to-plane metric if we accept the assumption that most measured points belong to surfaces perpendicular to the XY plane in the Cartesian coordinate system and the density of points along the XY plane of the target point cloud is sufficiently high.

This assumption should hold for surfaces such as walls that are close enough to the sensor. Pathways and corridors in buildings should generally satisfy this assumption, while cluttered or big spaces and natural structures do not. When this assumption is not met, the results should still be similar to those using the default point-to-point metric.

This pseudo-point-to-plane matching could be improved by introducing a new method of calculating normals and curvature, where normals would be calculated only in the XY plane.

3.3.4 Correction of displacement

Our method did not always converge to the global optimum when there was a significant change in position or orientation between two scans. Therefore, we decided to use a random sample consensus (RANSAC) [87] to improve the initial guess of the position and orientation before the first iteration of our ICP method.

We decided to use the PCL class `CorrespondenceRejectorSampleConsensus`, which uses the RANSAC to reject outliers. It has two configurable parameters: an inlier threshold and a maximum number of iterations. The inlier threshold is a distance, and it is in the same units as the source and target data sets. The class uses RANSAC to iteratively choose a given number of point pairs from a provided list of correspondences. A transformation is estimated given the point pairs chosen. The source point cloud is then transformed using the estimated transformation, and the number of points whose corresponding points are closer than the inlier threshold is recorded. After a fixed number of iterations set by the maximum number of iterations parameter, the pairs of points with the highest number of recorded correspondence distances below the inlier threshold are returned as inliers (filtered correspondences). This solution is similar to the outlier detection algorithm found in [88].

The `CorrespondenceRejectorSampleConsensus` class uses SVD to estimate transformations and thus suffers from the same problem as the base PCL ICP implementation. However, this effect is much smaller in this case because the transformations are only estimated on the given correspondence list, and only the list of inliers (remaining correspondences) is returned at the end. The improved initial estimate is obtained by passing the list of inliers to our `TransformationEstimationLM2D` class.

After extensive parameter testing, it was found that our algorithm for refining the initial guess may run too long before finding a good enough transformation. Two approaches provide the necessary speedup: reducing the number of correspondences by using another correspondence rejection algorithm and subsampling the data. We decided to subsample the data. The following PCL subsampling filters were tested:

- `VoxelGrid`.
- `ApproximateVoxelGrid`.
- `UniformSampling`.

The `VoxelGrid` class creates a 3D voxel grid [89] over the input point cloud. Then in each voxel, all points are approximated by their centroid. The `ApproximateVoxelGrid` class uses some approximations to create an output similar to subsampling using a 3D voxel grid. The `UniformSampling` class also creates a 3D voxel grid, but it downsamples the data by approximating each voxel with the closest point to the voxel's center.

In our testing, the `VoxelGrid` performed the best, with the `ApproximateVoxelGrid` being the second and the `UniformSampling` being last. We think the `VoxelGrid` performs best because it preserves the underlying structure well.

Other downsampling classes like the `RandomSample` or `FarthestPointSampling` could work too. The `RandomSample` class randomly selects points from the provided point cloud with uniform probability. The algorithm used in the class was based on Algorithm A from [90]. The `FarthestPointSampling` class selects points from the provided point cloud, starting with a random point and then applying the farthest point sampling (FPS) [91], [92] using the Euclidean distance [93]. We could not use the `FarthestPointSampling` class because it was not yet implemented in the version of PCL used while writing this thesis.

3.4 Mapping

It was also necessary to implement a method for mapping. The map had to be able to store data in 3D. The data structures that were considered to be suitable for the representation of the 3D map were:

- Point clouds.
- VoxelGrid.
- VoxelGridOcclusionEstimation[9].
- Octree.

Using only point clouds to represent the map seems to be the simplest solution. However, as the number of points increases, the memory requirements increase linearly. Therefore, using such a map representation is undesirable.

`VoxelGrid` is more memory efficient because when the map is updated with new point cloud data, the point clouds are downsampled.

`VoxelGridOcclusionEstimation` uses the same data structure as `VoxelGrid` but additionally implements a raytracing method to estimate occluded voxels. This method has been presented in [94].

Octree is a tree-like data structure where each internal node has eight children. Octree trees can be used to partition 3D space. The tree's root represents a cube bounding box surrounding a subset of the given 3D space. The space is then partitioned by recursively dividing the nodes into eight smaller cubes, or octants. Such an octree-based mapping method, called Octomap, was introduced in [95]. Like `VoxelGridOcclusionEstimation`, Octomap also implements a raytracing method. Using the octree as an internal data structure makes the map even more memory efficient than `VoxelGrid`. It also has the advantage of partially filtering out moving objects from the map due to the raytracing method used.

[9] `http://pointclouds.org/documentation/classpcl_1_1_voxel_grid_occlusion_estimation.html`

In the end, Octomap was chosen as the mapping method because it has the best properties for our purposes among all the other methods mentioned. The octomap_server[10] implementation, available as a ROS package, was configured and deployed as a ROS node.

3.5 Overview of the proposed localization system

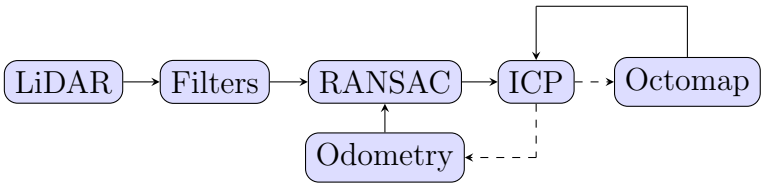


Figure 3.1: Diagram of the proposed localization pipeline.

Figure 3.1 shows a simplified diagram of the proposed localization algorithm. The inputs are LiDAR sensor data and MRS UAV Odometry data. The LiDAR data can be filtered in the Filter node. Then the sensor data is converted into a point cloud. The resulting point cloud, the map from the Octomap server, and the odometry data are used to estimate 2D translation and 1D rotation. The resulting estimated position and orientation can be sent to the MRS UAV Odometry. The aligned point cloud can be sent to the Octomap server to update the map.

The map update is controlled by thresholds. These thresholds are in the XY plane shift, Z axis shift, and rotation around the Z axis. If any of these thresholds are exceeded, the map is updated with the point cloud that has the lowest mean squared error of the point-to-point distance metric since the last update.

[10] `http://wiki.ros.org/octomap_server`

Chapter 4

Experiments

The proposed localization system was tested in a simulation. The simulations were conducted in the realistic Gazebo simulator[1][2]. The 3D visualization tool RViz[3] was used to display the acquired ROS data. For the purpose of data collection, a ROS node was programmed to collect data returned from the localization systems for later processing in Matlab[4]. It was also necessary to obtain the reference position and orientation of the helicopter for data comparison purposes, for which the ground truth pose is used. The ground truth pose is obtained by the Gazebo simulator and forwarded by the Real-time kinematic positioning (RTK) node (GNSS RTK pose estimation). The simulations were performed on a laptop with an Intel® Core™ i7-8565U processor, NVIDIA GeForce GTX 1050 4GB GDDR5 graphics card and 16GB of memory running Ubuntu 20.04.6 LTS operating system.The following tests are performed without a feedback loop with the MRS UAV Odometry node because a suitable Kalman filter for our localization algorithm has not been implemented.

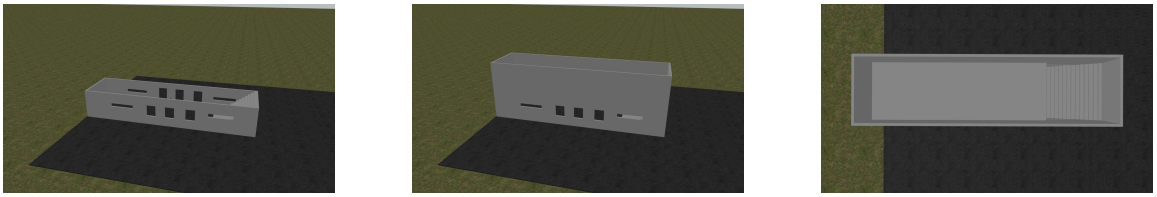
4.1 Test environments

To verify the functionality of the localization system, it was necessary to create test environments. The models of the buildings were created in the building editor go the Gazebo simulator. It was necessary to design them so that different characteristics of the tested algorithms could be verified. The result is two environments, which for the purposes of this thesis will be called the building and the loop.

[1] `https://gazebosim.org/`
[2] `https://github.com/ctu-mrs/simulation`
[3] `http://wiki.ros.org/rviz`
[4] `https://www.mathworks.com/products/matlab.html`

4.1.1 The building

The building was designed to test two challenging parameters for indoor localization: rotation and flight over stairs to a higher floor. The Figure 4.1 show a model of the building as seen from the Gazebo simulator.



(a) First floor. (b) Second floor. (c) Top view.

Figure 4.1: The environment for testing rotation and flight to the higher floor.

The rotation test is designed to verify the ability of the localization algorithms to respond to a large change in the displacement of distant points between two measurements caused by rotation. In this test, a small robotic helicopter takes off and gradually makes a full rotation around its vertical axis. The resulting map from the Octomap node is shown in Figure 4.2a as shown in RViz.

The flight to the higher floor test is designed to verify the ability of localization systems to perform localization during an upward flight in an environment with a horizontal cross-sectional variation. In this test, a small robotic helicopter takes off, reaches the stairs, climbs above them to the second floor and travels a short distance. The ground truth flight path is shown in the Figure 4.3 and the resulting map from the Octomap node is shown in Figure 4.2 as shown in RViz.

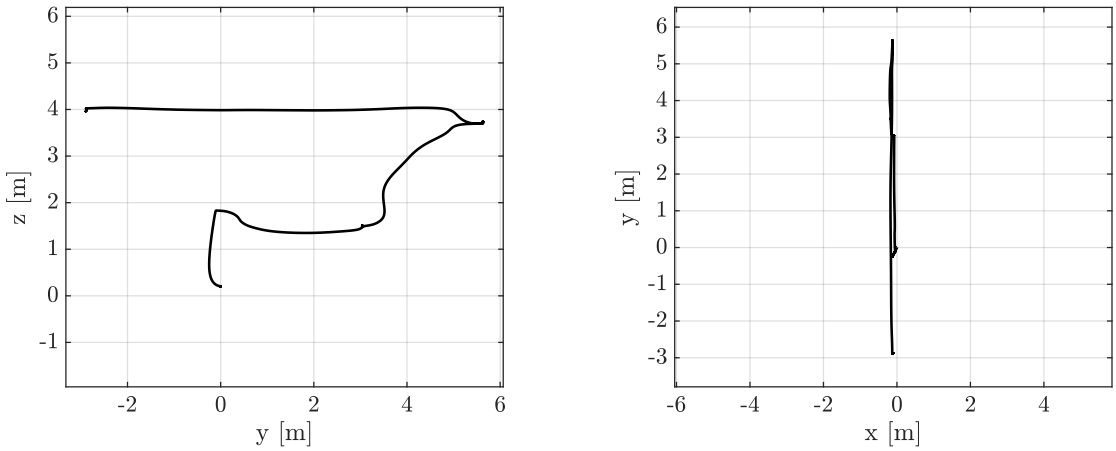
4.1.2 The loop

The loop was designed to test one of the important parameters of localization systems, loop closure. Figure 4.4a shows a model of the loop as seen from the Gazebo simulator.

The loop closure test is designed to verify the ability of the localization systems to close the reported flight path when visiting a previously visited location. In this test, the helicopter takes off, flies around the entire circuit and returns to the starting point. The ground truth flight path is shown in the Figure 4.5 and the resulting map from the Octomap node is shown in Figure 4.4b as shown in RViz.

(a) Rotation test. (b) Flight to the higher floor.

Figure 4.2: Octomap maps visualized with RViz.

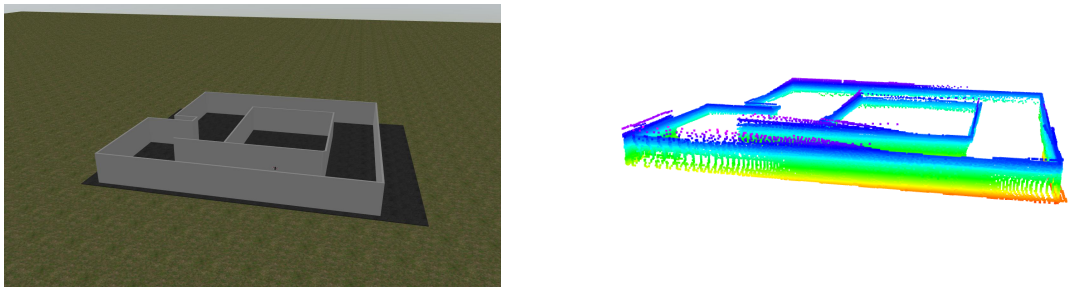


(a) Side view. (b) Top view.

Figure 4.3: Flight path for the flight to the higher floor test.

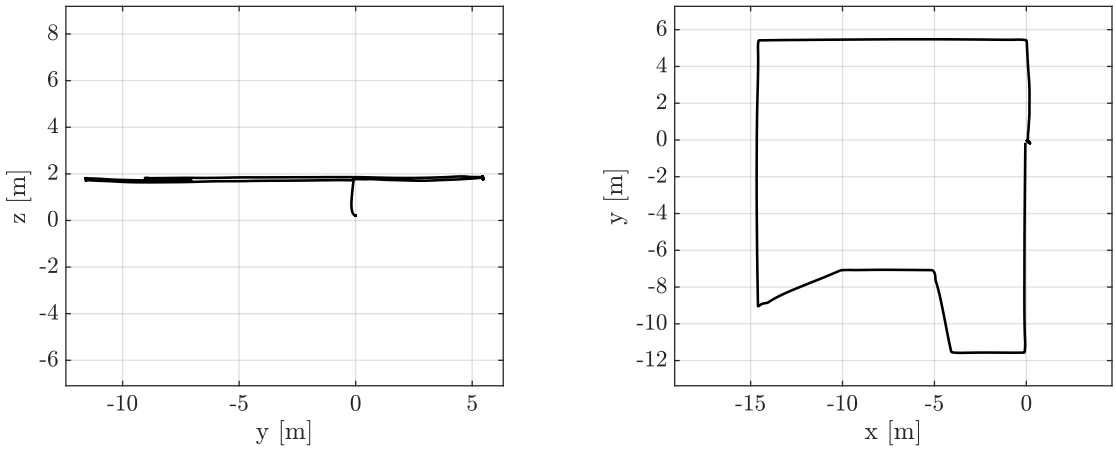
4.2 Testing the RANSAC displacement correction

In this section, a version of our proposed localization algorithm is tested both with and without the RANSAC displacement correction method. The localization system without the RANSAC component is called ICP and the localization system with the RANSAC component is called ICP+RANSAC. The purpose is to compare the performance of the whole system as well as the performance of its individual parts.



(a) The environment visualized in Gazebo. (b) Octomap map visualized with RViz.

Figure 4.4: The environment for testing loop closure.



(a) Side view. (b) Top view.

Figure 4.5: Flight path for the loop closure test.

4.2.1 The rotation test

In the Figure 4.6 is the flight path reported by both systems. The flight path figure is dense because the helicopter was only rotating in place. The Figure 4.7 shows the absolute deviation of position from ground truth and the Figure 4.8 shows the absolute deviation of orientation from ground truth. The box plots in Figure 4.9 show the performance of ICP and ICP+RANSAC in terms of both absolute position deviation and absolute orientation deviation.

It can be seen that the two systems have similar performance in determining the heading angle with the same absolute deviation of about 2.5 radians around the time of 25 seconds. The positioning performance is comparable, but the ICP performs slightly better here. However, precise positioning is not the main focus of the rotation test.

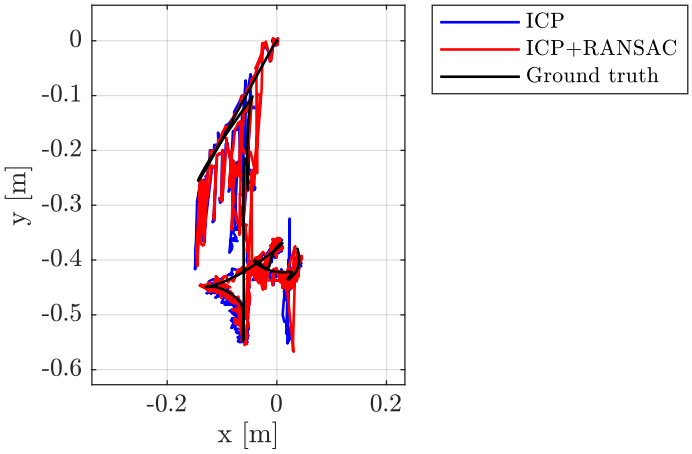


Figure 4.6: The rotation test flight path.

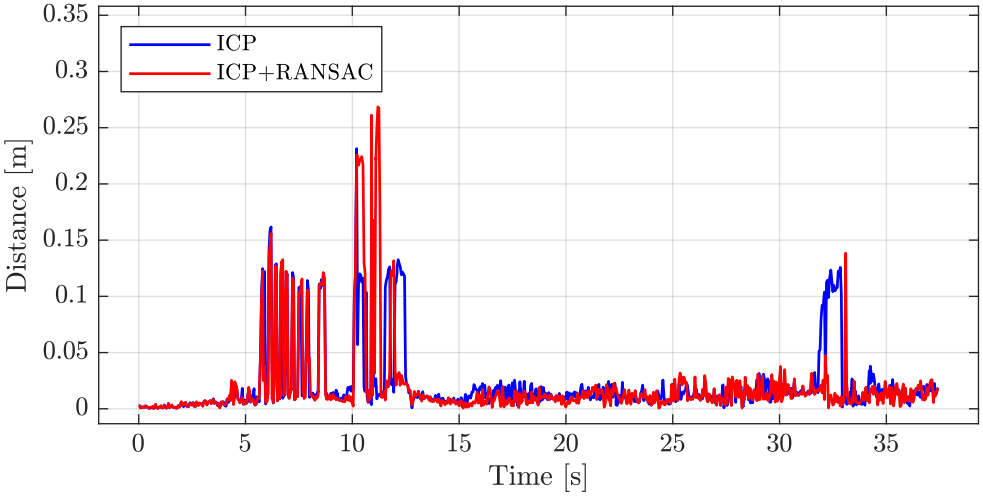


Figure 4.7: Absolute position deviation from the ground truth.

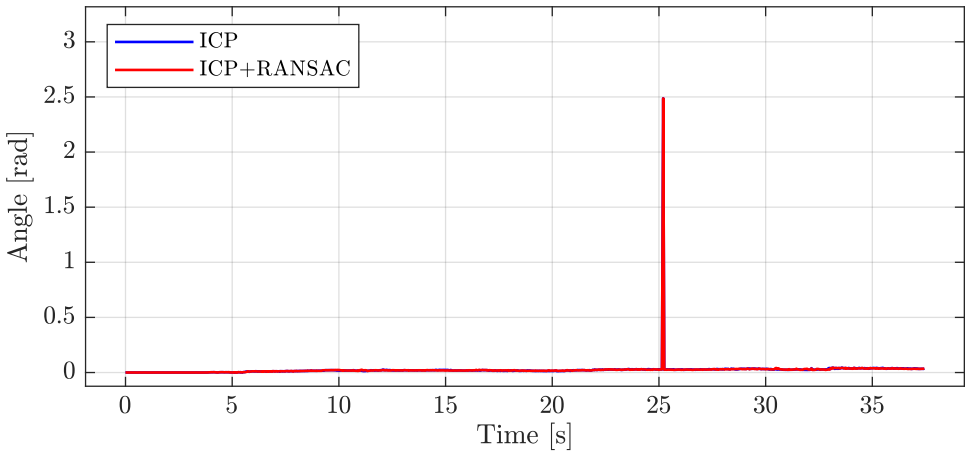
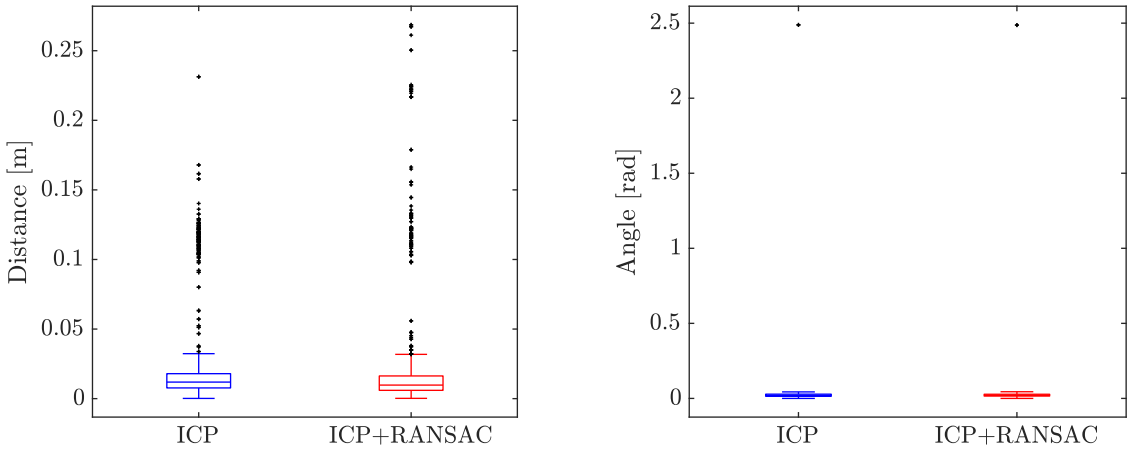


Figure 4.8: Absolute orientation deviation from the ground truth.



(a) Absolute position deviation from the ground truth.

(b) Absolute orientation deviation from the ground truth.

Figure 4.9: Comparison of ICP and ICP+RANSAC in the rotation test.

4.2.2 The flight to the higher floor test

In the Figure 4.10 is the flight path reported by both systems. The image of the flight path looks like a line because it is a top view, so the altitude change to reach the second floor is not visible. The Figure 4.11 shows the absolute deviation of position from ground truth and the Figure 4.12 shows the absolute deviation of orientation from ground truth. The box plots in Figure 4.13 show the performance of ICP and ICP+RANSAC in terms of both absolute position deviation and absolute orientation deviation.

Again it can be seen that the two systems have similar performance in determining the heading angle with the same absolute deviation of about 0.034 radians around the time of 45 seconds. The positioning performance is similar, but again the ICP performs slightly better here. Both systems report maximum absolute position errors of about 35 cm, which is at the limit of what can be used for reliable indoor positioning.

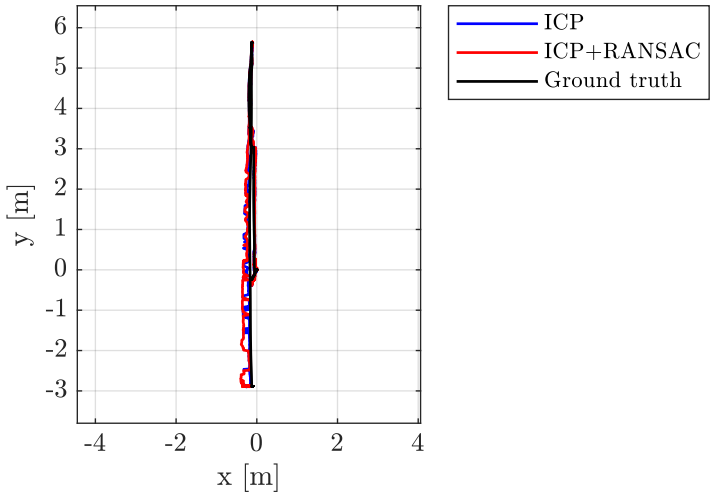


Figure 4.10: The flight to the higher floor test flight path.

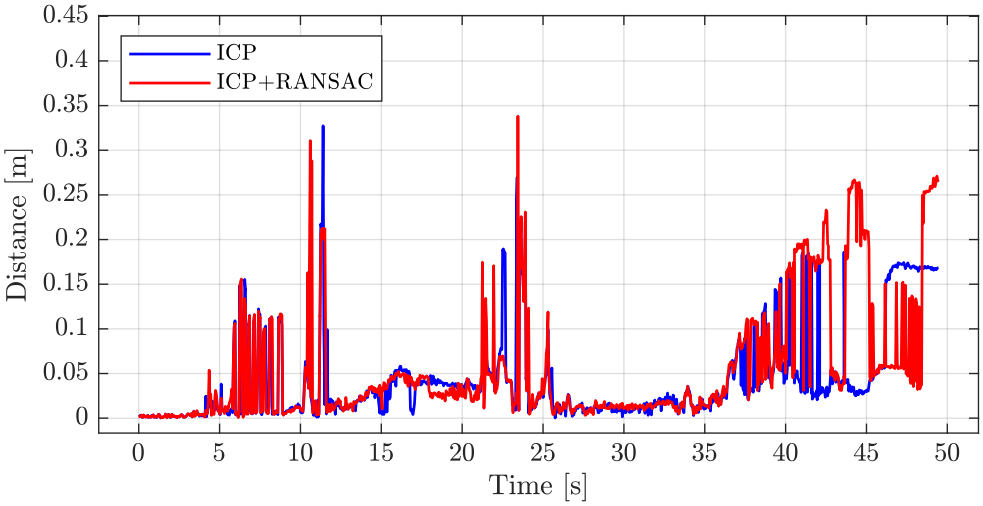


Figure 4.11: Absolute position deviation from the ground truth.

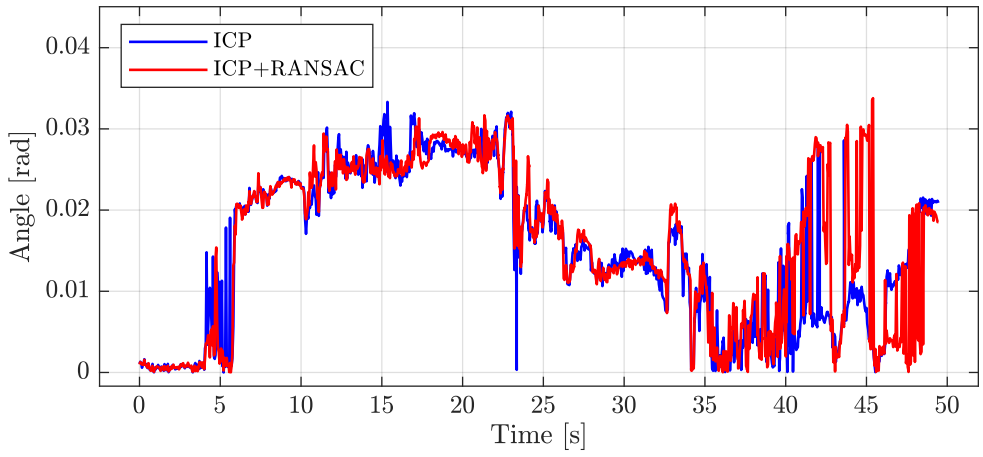
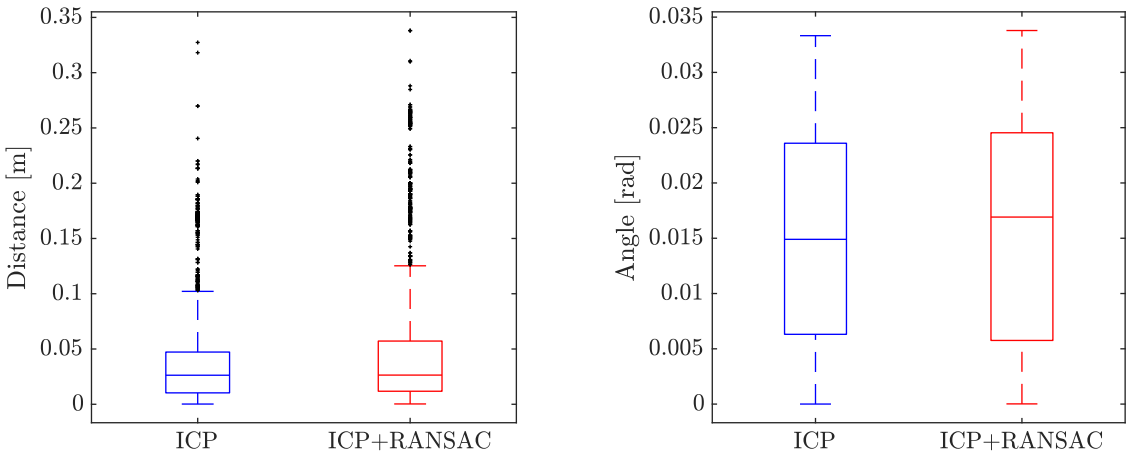


Figure 4.12: Absolute orientation deviation from the ground truth.



(a) Absolute position deviation from the ground truth.

(b) Absolute orientation deviation from the ground truth.

Figure 4.13: Comparison of ICP and ICP+RANSAC in the flight to the higher floor test.

4.2.3 The loop closure test

In the Figure 4.14 is the flight path reported by both systems. The Figure 4.15 shows the absolute deviation of position from ground truth and the Figure 4.16 shows the absolute deviation of orientation from ground truth. The box plots in Figure 4.17 show the performance of ICP and ICP+RANSAC in terms of both absolute position deviation and absolute orientation deviation.

Here, ICP achieves significantly better position estimation results. Its maximum absolute deviation is about 0.5 m, while ICP+RANSAC has an absolute positioning deviation of about 1.5 m. As can be seen from the figure and the figure, the error in the reported position of the ICP+RANSAC positioning system is due to a jump of about 5 seconds, which occurred during takeoff. Both systems perform similarly in determining the heading angle, with ICP+RANSAC performing slightly better.

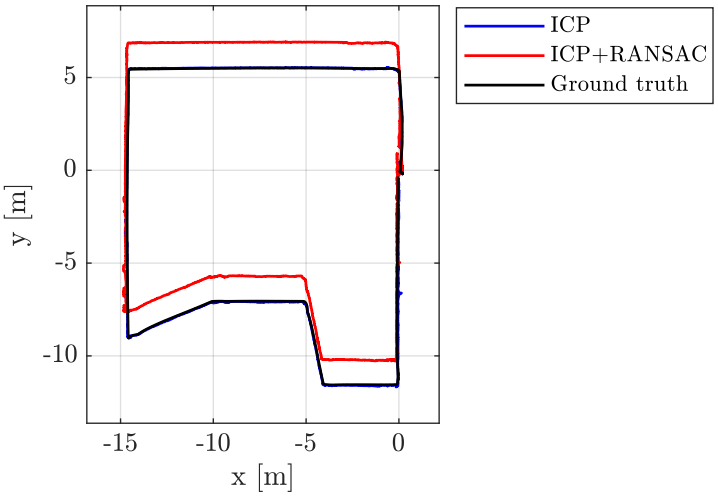


Figure 4.14: The loop closure test flight path.

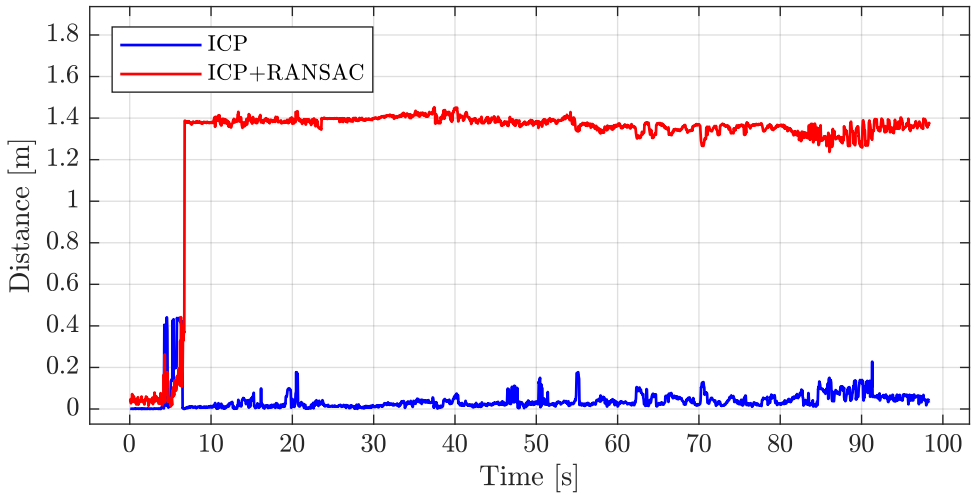


Figure 4.15: Absolute position deviation from the ground truth.

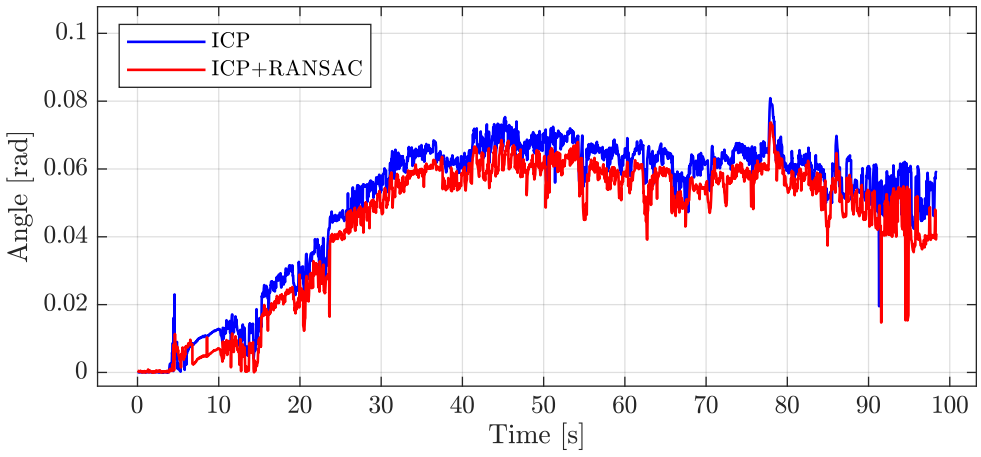
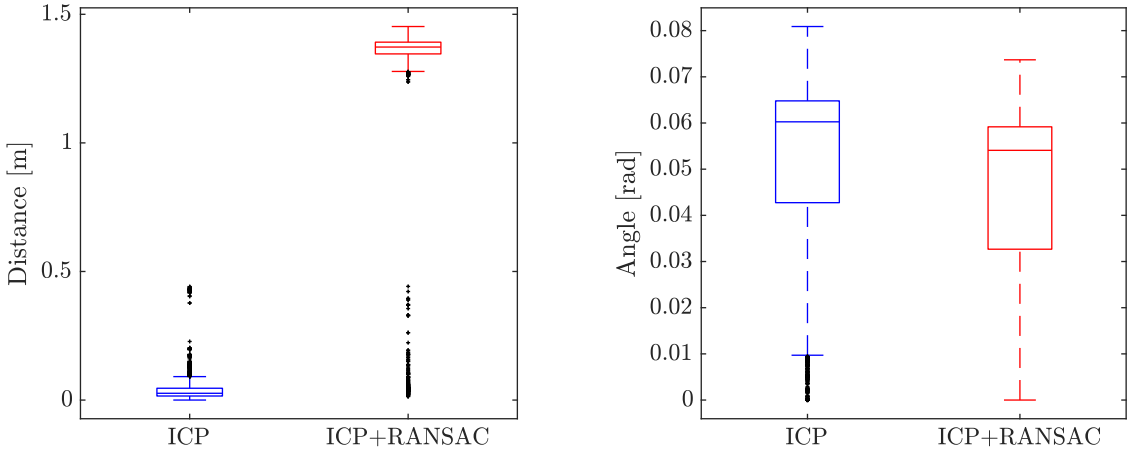


Figure 4.16: Absolute orientation deviation from the ground truth.



(a) Absolute position deviation from the ground truth.

(b) Absolute orientation deviation from the ground truth.

Figure 4.17: Comparison of ICP and ICP+RANSAC in the loop closure test.

4.3 Comparison with the Hector SLAM

For further testing, it was decided to run the tests without the RANSAC displacement correction method, as previous tests have shown that the performance of ICP and ICP+RANSAC is comparable, and in some cases the ICP results were even better. Tests were conducted at low flight speeds, and we discuss that the errors that the RANSAC displacement correction method would handle better would be more apparent at higher flight speeds.

The following tests compare the parameters of our proposed localization system without the RANSAC displacement correction method with the Hector SLAM. Hector SLAM is still considered to be a state-of-the-art localization system, mostly used by UAVs and UGVs. Hector SLAM is also used by the MRS Group[5].

4.3.1 The rotation test

In the Figure 4.18 is the flight path reported by both systems. The Figure 4.19 shows the absolute deviation of position from ground truth and the Figure 4.20 shows the absolute deviation of orientation from ground truth. The box plots in Figure 4.21 show the performance of ICP and Hector SLAM in terms of both absolute position deviation and absolute orientation deviation.

Both systems performed similarly, with the ICP performing worse. The maximum absolute deviation in both position and orientation is acceptable.

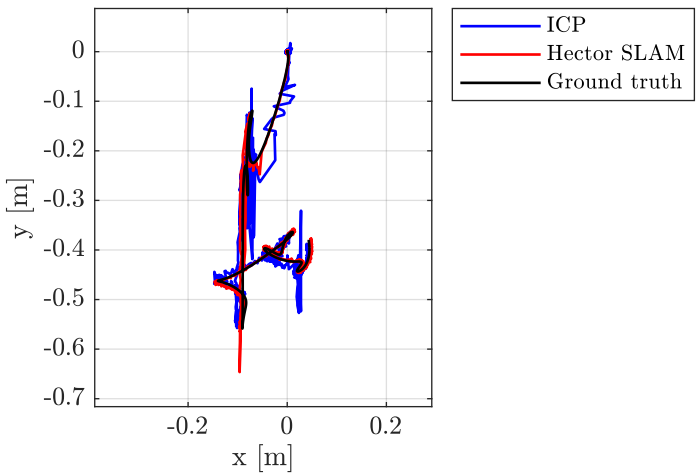


Figure 4.18: The rotation test flight path.

[5] https://github.com/ctu-mrs/hector_slam

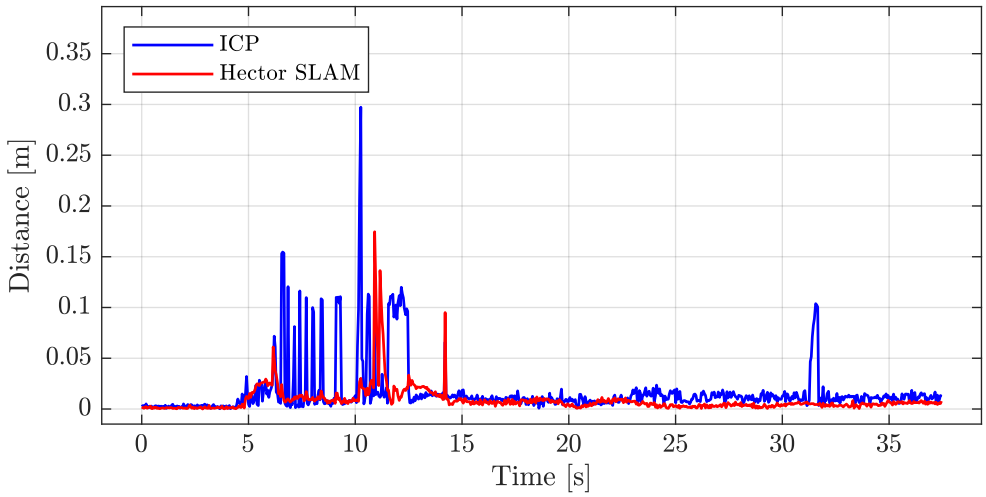


Figure 4.19: Absolute position deviation from the ground truth.

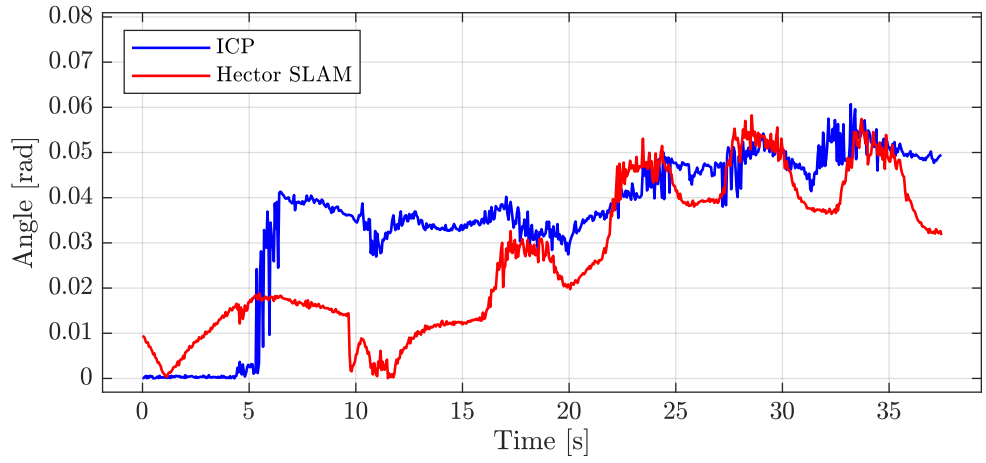
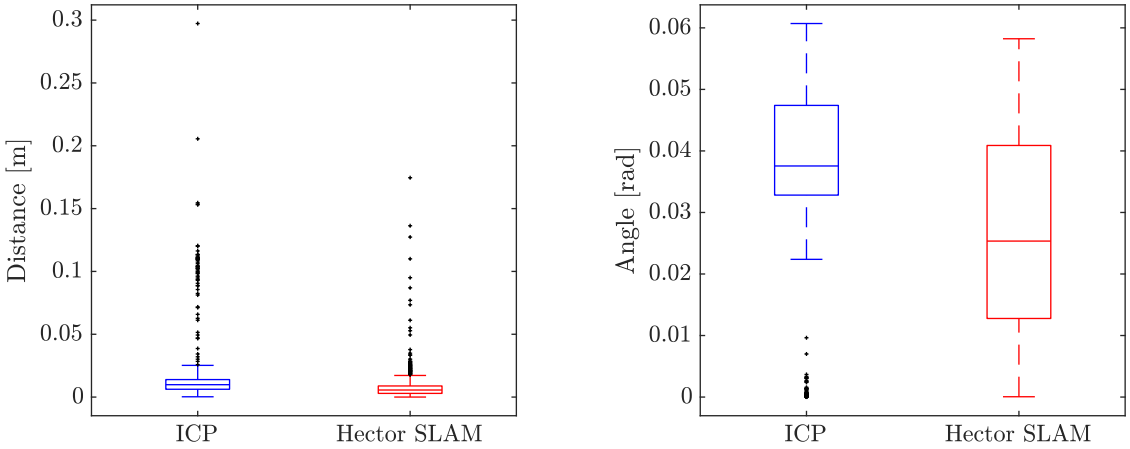


Figure 4.20: Absolute orientation deviation from the ground truth.



(a) Absolute position deviation from the ground truth.

(b) Absolute orientation deviation from the ground truth.

Figure 4.21: Comparison of ICP and Hector SLAM in the rotation test.

4.3.2 The flight to the higher floor test

In the Figure 4.22 is the flight path reported by both systems. The Figure 4.23 shows the absolute deviation of position from ground truth and the Figure 4.24 shows the absolute deviation of orientation from ground truth. The box plots in Figure 4.25 show the performance of ICP and Hector SLAM in terms of both absolute position deviation and absolute orientation deviation.

Here Hector SLAM had considerably better results. The maximum absolute deviation in position for ICP was over 70 cm, when Hector SLAM had only 30 cm. However, the median values are comparable between the two methods.

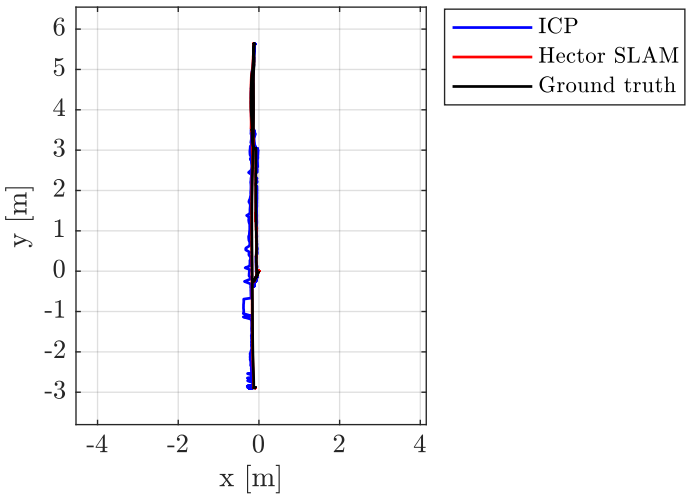


Figure 4.22: The flight to the higher floor test flight path.

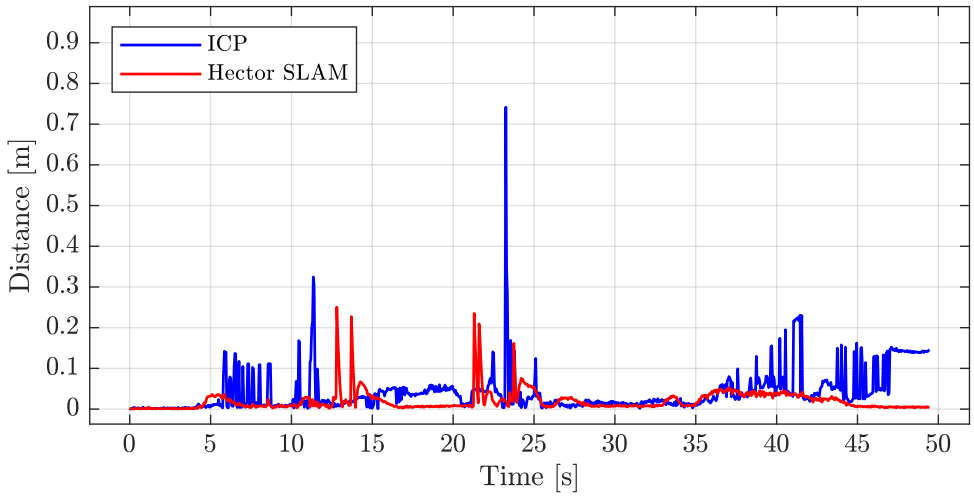


Figure 4.23: Absolute position deviation from the ground truth.

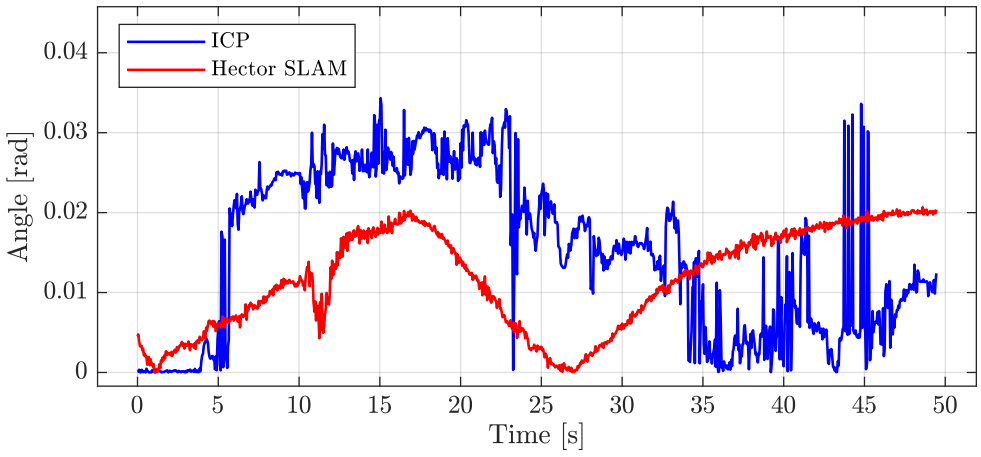
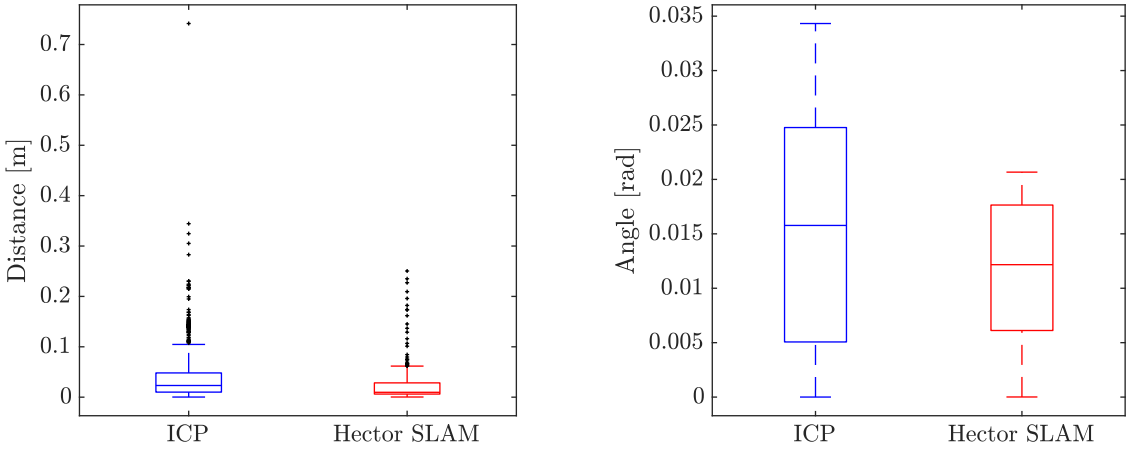


Figure 4.24: Absolute orientation deviation from the ground truth.



(a) Absolute position deviation from the ground truth.

(b) Absolute orientation deviation from the ground truth.

Figure 4.25: Comparison of ICP and Hector SLAM in the flight to the higher floor test.

4.3.3 The loop closure test

In the Figure 4.26 is the flight path reported by both systems. The Figure 4.27 shows the absolute deviation of position from ground truth and the Figure 4.28 shows the absolute deviation of orientation from ground truth. The box plots in Figure 4.29 show the performance of ICP and Hector SLAM in terms of both absolute position deviation and absolute orientation deviation.

Although neither system has a loop closure detection algorithm, both were able to report approximately the correct position at loop closure. The ICP localization method performed quite well in this test. Its maximum absolute position error was less than 25 cm and its maximum absolute orientation error was less than 0.09 radians. On the other hand, the Hector SLAM was better in both of these parameters.

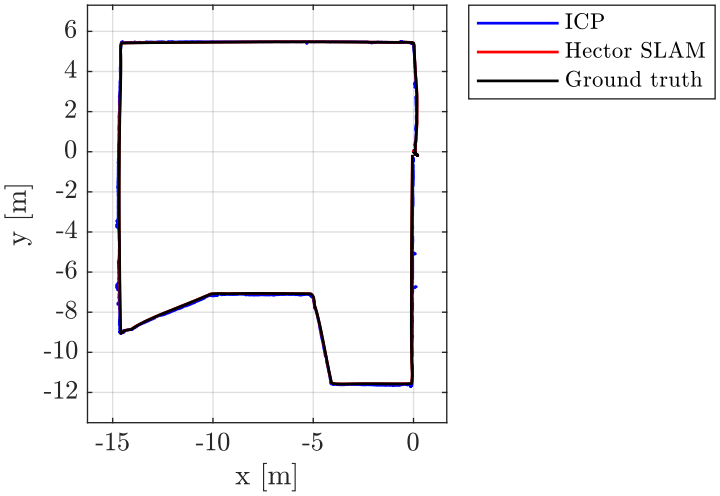


Figure 4.26: The loop closure test flight path.

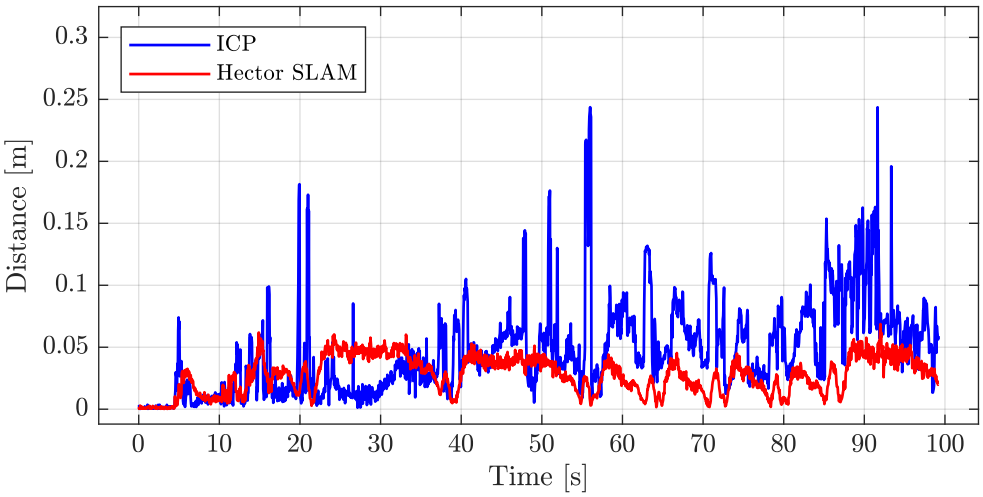


Figure 4.27: Absolute position deviation from the ground truth.

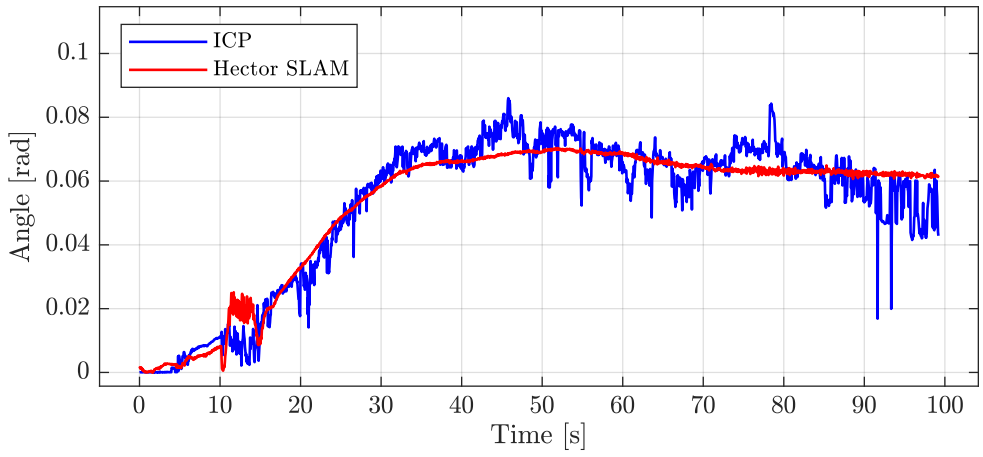
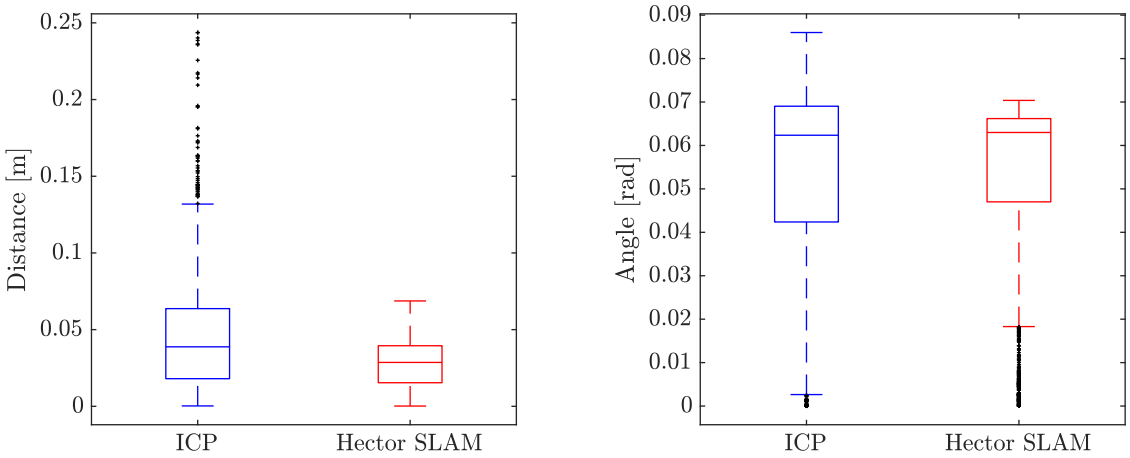


Figure 4.28: Absolute orientation deviation from the ground truth.



(a) Absolute position deviation from the ground truth.

(b) Absolute orientation deviation from the ground truth.

Figure 4.29: Comparison of ICP and Hector SLAM in the loop closure test.

4.4 Odometry feedback loop

The previous tests were performed without feedback with MRS UAV Odometry because although our localization system is compatible with MRS UAV Odometry because it uses the same message types as Hector SLAM, the position and orientation data are noisy and have different parameters. For a good cooperation between the two systems, it would be necessary to add a new Kalman filter to the bank of Kalman filters (in the MRS UAV Odometry).

This section shows how the system might work approximately when it has a good initial position estimate from odometry. This is done by using the Hector SLAM as the transformation estimator and passing the estimated position and rotation estimate to our proposed localization system.

In the Figure 4.30, we can see that ICP+Odometry has fewer outliers in terms of absolute distance error in position, but its accuracy has not improved, on the contrary, it is slightly worse than that of ICP. Where its performance has improved is in absolute orientation error, as can be seen in the Figure 4.31. However, it still does not achieve the same results as Hector SLAM, which has the best results in both measured parameters.

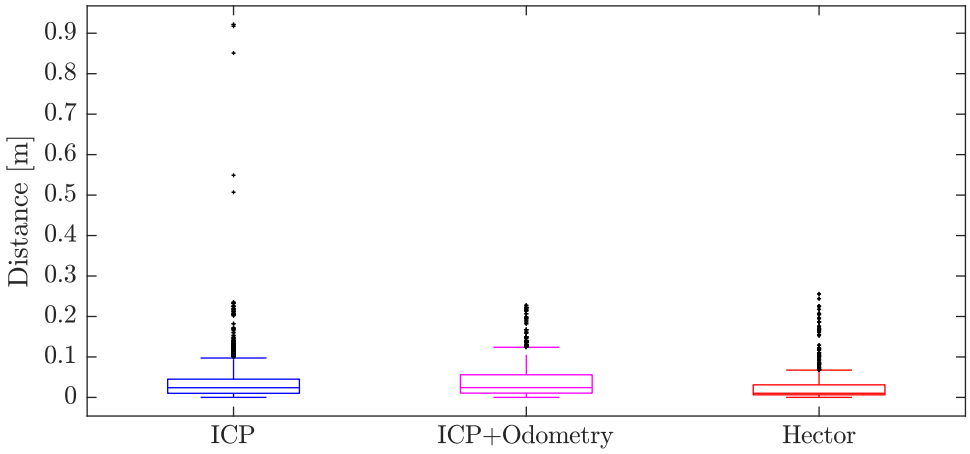


Figure 4.30: Absolute position deviation from the ground truth, comparison of ICP, ICP+Odometry, and Hector SLAM.

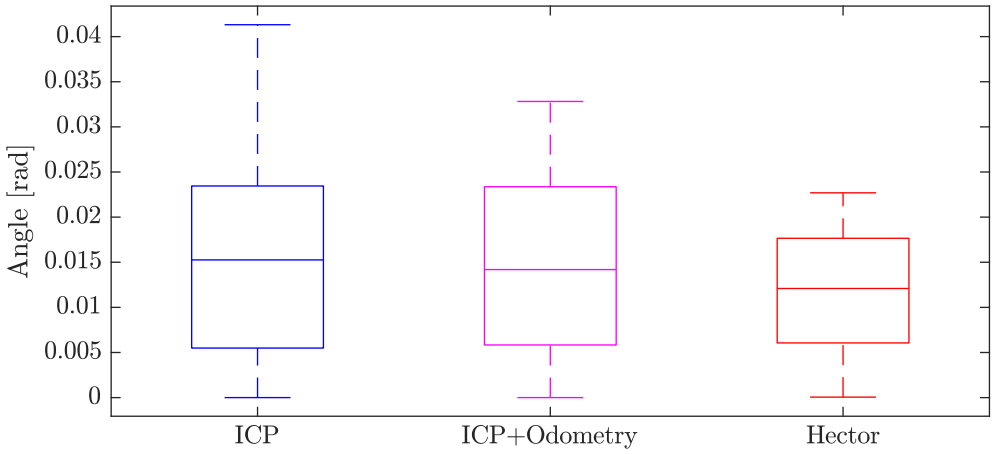


Figure 4.31: Absolute orientation deviation from the ground truth, comparison of ICP, ICP+Odometry, and Hector SLAM.

Chapter 5

Conclusion

This work introduced a localization system for a small robotic helicopter in an indoor environment. Other localization methods were also researched. Experiments were conducted in a realistic Gazebo 3D simulator. The performance and accuracy of the presented localization system were compared with the state-of-the-art localization system Hector SLAM. Although the performance of the presented system did not outperform Hector SLAM, it should be applicable for indoor localization.

Future work could focus on the following areas. Design and implement rotation correction such as by cross-correlation of two consecutive LiDAR scans or rotation estimation using angle histograms [96]. Furthermore, better integrate the introduced localization system with MRS UAV Odometry, configure and add a new Kalman filter to the bank of Kalman filters.

Bibliography

[1] V. Pritzl, M. Vrba, C. Tortorici, R. Ashour, and M. Saska, "Adaptive estimation of uav altitude in complex indoor environments using degraded and time-delayed measurements with time-varying uncertainties," *Robotics and Autonomous Systems*, vol. 160, p. 104315, 2023.

[2] J. Hall and K. Mohseni, *Micro Aerial Vehicles*. New York, NY: Springer New York, 2015, pp. 1770–1778. [Online]. Available: https://doi.org/10.1007/978-1-4614-5491-5_892

[3] 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, 2012.

[4] M. Atif, R. Ahmad, W. Ahmad, L. Zhao, and J. J. Rodrigues, "Uav-assisted wireless localization for search and rescue," *IEEE Systems Journal*, vol. 15, no. 3, pp. 3261–3272, 2021.

[5] M. Petrlík, T. Báča, D. Heřt, M. Vrba, T. Krajník, 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, 2020.

[6] C. Yuan, Y. Zhang, and Z. Liu, "A survey on technologies for automatic forest fire monitoring, detection, and fighting using unmanned aerial vehicles and remote sensing techniques," *Canadian journal of forest research*, vol. 45, no. 7, pp. 783–792, 2015.

[7] V. Spurny, V. Pritzl, V. Walter, M. Petrlik, T. Baca, P. Stepan, D. Zaitlik, and M. Saska, "Autonomous firefighting inside buildings by an unmanned aerial vehicle," *IEEE Access*, vol. 9, pp. 15 872–15 890, 2021.

[8] D. W. Murphy and J. Cycon, "Applications for mini vtol uav for law enforcement," in *Sensors, C3I, information, and training technologies for law enforcement*, vol. 3577. SPIE, 1999, pp. 35–43.

[9] M. Vrba, D. Heřt, and M. Saska, "Onboard marker-less detection and localization of non-cooperating drones for their safe interception by an autonomous aerial system," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3402–3409, 2019.

[10] H. Qin, Z. Meng, W. Meng, X. Chen, H. Sun, F. Lin, and M. H. Ang, "Autonomous exploration and mapping system using heterogeneous uavs and ugvs in gps-denied environments," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1339–1350, 2019.

[11] P. Štibinger, T. Báča, and M. Saska, "Localization of ionizing radiation sources by cooperating micro aerial vehicles with pixel detectors in real-time," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3634–3641, 2020.

[12] J. Wells, B. Lovelace *et al.*, "Improving the quality of bridge inspections using unmanned aircraft systems (uas)," Minnesota. Department of Transportation, Tech. Rep., 2018.

[13] P. Petráček, V. Krátkỳ, and M. Saska, "Dronument: System for reliable deployment of micro aerial vehicles in dark areas of large historical monuments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2078–2085, 2020.

[14] J. Chudoba, M. Saska, T. Báča, and L. Přeučil, "Localization and stabilization of micro aerial vehicles based on visual features tracking," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2014, pp. 611–616.

[15] V. Pritzl, M. Vrba, C. Tortorici, R. Ashour, and M. Saska, "Adaptive estimation of uav altitude in complex indoor environments using degraded and time-delayed measurements with time-varying uncertainties," *Robotics and Autonomous Systems*, vol. 160, p. 104315, 2023.

[16] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 15–22.

[17] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[18] M. J. M. M. Mur-Artal, Raúl and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[19] 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*. IEEE, 2011, pp. 155–160.

[20] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, 2005, pp. 2432–2437.

[21] ——, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[22] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1271–1278.

[23] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, vol. 2, no. 9. Berkeley, CA, 2014, pp. 1–9.

[24] H. Wang, C. Wang, C.-L. Chen, and L. Xie, "F-loam : Fast lidar odometry and mapping," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sep. 2021, pp. 4390–4396.

[25] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, Jan. 2009, p. 5.

[26] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[27] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 3. IEEE, Nov. 2003, pp. 2436–2441.

[28] B. P. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. S. Sukhatme, and M. J. Mataric, "Most valuable player: A robot device server for distributed control," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, vol. 3. IEEE, 2001, pp. 1226–1231.

[29] B. Gerkey, R. T. Vaughan, A. Howard *et al.*, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1. Citeseer, 2003, pp. 317–323.

[30] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proceedings of the Australasian conference on robotics and automation (ACRA 2005)*. Citeseer Citeseer, 2005, p. 145.

[31] J. L. B. Claraco, "Development of scientific applications with the mobile robot programming toolkit," Oct. 2010. [Online]. Available: http://www.mrpt.org/downloads/mrpt-book.pdf

[32] S. M. Rea, *Programming microsoft® robotics studio*. Microsoft Press, Mar. 2008.

[33] NVIDIA Corporation, "Nvidia launches isaac sdk, a new platform to accelerate development of ai-powered robotics," https://blogs.nvidia.com/blog/2018/03/27/isaac-robotics-sdk/, Sep. 2018.

[34] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W.-K. Yoon, "Rt-middleware: distributed component middleware for rt (robot technology)," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sep. 2005, pp. 3933–3938.

[35] N. Ando, T. Suehiro, and T. Kotoku, "A software platform for component based rt-system development: Openrtm-aist," in *Simulation, Modeling, and Programming for Autonomous Robots: First International Conference, SIMPAR 2008 Venice, Italy, November 3-6, 2008. Proceedings 1*. Springer, 2008, pp. 87–98.

[36] B. Song, S. Jung, C. Jang, and S. Kim, "An introduction to robot component model for opros (open platform for robotic services)," in *Proceedings of the International Conference Simulation, Modeling Programming for Autonomous Robots Workshop*, Jan. 2008, pp. 592–603.

[37] H. Bruyninckx, "Open robot control software: the orocos project," in *Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164)*, vol. 3. IEEE, Feb. 2001, pp. 2523–2528.

[38] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *IEEE International Conference on Robotics and Automation*, 2003, pp. 2766–2771.

[39] P. Soetens and H. Bruyninckx, "Realtime hybrid task-based control for robots and machine tools," in *IEEE International Conference on Robotics and Automation*, 2005, pp. 260–265.

[40] P. Soetens, "RTT: Real-Time Toolkit," http://www.orocos.org/rtt.

[41] W. Li, H. I. Christensen, A. Oreback, and D. Chen, "An architecture for indoor navigation," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 2. IEEE, 2004, pp. 1783–1788.

[42] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback, "Towards component-based robotics," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 163–168.

[43] A. Makarenko, A. Brooks, and T. Kaupp, "Orca: Components for robotics," in *International Conference on Intelligent Robots and Systems (IROS)*. Citeseer, 2006, pp. 163–168.

[44] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Orebäck, "Orca: A component model and repository," *Software engineering for experimental robotics*, vol. 30, pp. 231–251, 2007.

[45] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 8, 2006.

[46] D. P. Eickstedt, "Adaptive sampling in autonomous marine sensor networks," WOODS HOLE OCEANOGRAPHIC INSTITUTION MA, Tech. Rep., 2006.

[47] P. M. Newman, "Moos-mission orientated operating suite," 2008.

[48] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard, "Nested autonomy for unmanned marine vehicles with moos-ivp," *Journal of Field Robotics*, vol. 27, no. 6, pp. 834–875, 2010.

[49] S. Joyeux and J. Albiez, "Robot development: from components to systems," in *6th National Conference on Control Architectures of Robots*, 2011, pp. 15–p.

[50] S. Joyeux, J. Schwendner, T. M. Roehr, and R. I. Center, "Modular software for an autonomous space rover," in *The 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2014)*, 2014.

[51] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 6235–6240.

[52] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "Micro air vehicle link (mavlink) in a nutshell: A survey," *IEEE Access*, vol. 7, pp. 87 658–87 680, 2019.

[53] M. Takaaki, Y. Yuichi, N. Keitaro, and N. Yoshiaki, "Developing a scheme of controlling drones using openrtm-aist," *The Proceedings of JSME annual Conference on Robotics and Mechatronics (Robomec)*, vol. 2016, no. 0, pp. 2P2–02b3, 2016.

[54] Y. Yaguchi, Y. Nitta, S. Ishizaka, T. Tannai, T. Mamiya, K. Naruse, and S. Nakano, "Formation control for different maker drones from a game pad," in *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2017, pp. 1373–1378.

[55] T. Báča, M. Petrlík, M. Vrba, V. Spurný, R. Pěnička, D. Hert, and M. Saska, "The mrs uav system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 1, pp. 1–28, 2021.

[56] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995.

[57] D. Hert, T. Baca, P. Petracek, V. Kratky, V. Spurny, M. Petrlik, M. Vrba, D. Zaitlik, P. Stoudek, V. Walter, P. Stepan, J. Horyna, V. Pritzl, G. Silano, D. Bonilla Licea, P. Stibinger, R. Penicka, T. Nascimento, and M. Saska, "MRS Modular UAV Hardware Platforms for Supporting Research in Real-World Outdoor and Indoor Environments," in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, Jun. 2022, pp. 1264–1273.

[58] Multi-robot Systems Group, "Unmanned aerial vehicles," https://github.com/ctu-mrs/mrs_uav_system#unmanned-aerial-vehicles, 2023.

[59] ——, "Micro aerial vehicles - platforms," http://mrs.felk.cvut.cz/research/micro-aerial-vehicles, 2023.

[60] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 3400–3407.

[61] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4193–4198.

[62] Y. Wu, F. Tang, and H. Li, "Image-based camera localization: an overview," *Visual Computing for Industry, Biomedicine, and Art*, vol. 1, no. 1, pp. 1–13, 2018.

[63] Y. Tian, X. Liu, L. Li, and W. Wang, "Intensity-assisted icp for fast registration of 2d-lidar," *Sensors*, vol. 19, no. 9, p. 2124, May 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/9/2124

[64] P. Biber and W. Straßer, "The normal distributions transform: A new approach to laser scan matching," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 3. IEEE, Nov. 2003, pp. 2743–2748.

[65] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan registration for autonomous mining vehicles using 3d-ndt," *Journal of Field Robotics*, vol. 24, no. 10, pp. 803–827, Oct. 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20204

[66] K. Koide, J. Miura, and E. Menegatti, "A portable three-dimensional lidar-based system for long-term and wide-area people behavior measurement," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1729881419841532, Feb. 2019. [Online]. Available: https://doi.org/10.1177/1729881419841532

[67] T. Quin and S. Cao, "A-LOAM: Advanced implementation of LOAM," Apr. 2018, accessed: 2023-02-13. [Online]. Available: https://github.com/HKUST-Aerial-Robotics/A-LOAM

[68] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2018, pp. 4758–4765.

[69] L. Liao, C. Fu, B. Feng, and T. Su, "Optimized sc-f-loam: Optimized fast lidar odometry and mapping using scan context," 2022. [Online]. Available: https://arxiv.org/abs/2204.04932

[70] H. Ye, Y. Chen, and M. Liu, "Tightly coupled 3d lidar inertial odometry and mapping," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, May 2019, pp. 3144–3150.

[71] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and Vision Computing*, vol. 10, no. 3, pp. 145–155, Jan. 1992, range Image Understanding. [Online]. Available: https://www.sciencedirect.com/science/article/pii/026288569290066C

[72] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[73] A. V. Segal, D. Hähnel, and S. Thrun, "Generalized-icp," in *Robotics: Science and Systems*, J. Trinkle, Y. Matsuoka, and J. A. Castellanos, Eds., vol. 2, no. 4, Seattle, WA. The MIT Press, Jun. 2009, p. 435. [Online]. Available: http://dblp.uni-trier.de/db/conf/rss/rss2009.html#SegalHT09

[74] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, "Voxelized gicp for fast and accurate 3d point cloud registration," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2021, pp. 11 054–11 059.

[75] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[76] R. P. I. I. P. Laboratory and D. Meagher, *Octree Encoding: a New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*, Oct. 1980. [Online]. Available: https://books.google.cz/books?id=CgRPOAAACAAJ

[77] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[78] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[79] A. Censi, "An icp variant using a point-to-line metric," in *2008 IEEE International Conference on Robotics and Automation*. Ieee, Jun. 2008, pp. 19–25.

[80] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP Variants on Real-World Data Sets," *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, Feb. 2013.

[81] F. Pomerleau, S. Magnenat, F. Colas, M. Liu, and R. Siegwart, "Tracking a depth camera: Parameter exploration for fast icp," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, 2011, pp. 3824–3829.

[82] S. Nobili, R. Scona, M. Caravagna, and M. Fallon, "Overlap-based ICP tuning for robust localization of a humanoid robot," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2017.

[83] S. Nobili, M. Camurri, V. Barasuol, M. Focchi, D. Caldwell, C. Semini, and M. Fallon, "Heterogeneous sensor fusion for accurate state estimation of dynamic legged robots," in *Robotics: Science and Systems (RSS)*, Jul. 2017.

[84] S. Nobili, G. Tinchev, and M. Fallon, "Predicting alignment risk to prevent localization failure," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[85] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[86] A. W. Fitzgibbon, "Robust registration of 2d and 3d point sets," *Image and Vision Computing*, vol. 21, no. 13-14, pp. 1145–1153, Apr. 2003, british Machine Vision Computing 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885603001835

[87] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[88] P. H. Torr and D. W. Murray, "Outlier detection and motion segmentation," in *Sensor Fusion VI*, vol. 2059. SPIE, 1993, pp. 432–443.

[89] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer graphics*, 2nd ed., ser. Addison-Wesley systems programming series. London, England: Addison Wesley, Jun. 1990.

[90] J. S. Vitter, "Faster methods for random sampling," *Communications of the ACM*, vol. 27, no. 7, pp. 703–718, 1984.

[91] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi, "The farthest point strategy for progressive image sampling," *IEEE Transactions on Image Processing*, vol. 6, no. 9, pp. 1305–1315, 1997.

[92] C. Moenning and N. A. Dodgson, "Fast marching farthest point sampling for point clouds and implicit surfaces," University of Cambridge, Computer Laboratory, Tech. Rep., Jun. 2003.

[93] K. J. Smith, *Precalculus: A functional approach to graphing and problem solving*. Jones & Bartlett Publishers, 2013. [Online]. Available: https://books.google.com/books?id=ZUJbVQN37bIC&pg=PA8

[94] J. Amanatides, A. Woo *et al.*, "A fast voxel traversal algorithm for ray tracing." in *Eurographics*, vol. 87, no. 3, 1987, pp. 3–10.

[95] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at https://octomap.github.io. [Online]. Available: https://octomap.github.io

[96] G. Weiß, C. Wetzler, and E. Von Puttkamer, "Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, vol. 1. IEEE, 1994, pp. 595–601.

Appendices

List of abbreviations

In the Table 1 and 2 are listed abbreviations used in this thesis.

Abbreviation	Meaning
UAV	Unmanned Aerial Vehicle
MAV	Micro Aerial Vehicle
GPS	Global Positioning System
GNSS	Global Navigation Satellite System
UGV	Unmanned Ground Vehicle
1D	One-dimensional
2D	Two-dimensional
3D	Three-dimensional
LiDAR	Light Detection And Ranging
RaDAR	Radio Detection And Ranging
SoNAR	Sound Navigation And Ranging
SLAM	Simultaneous Localization and Mapping
SVO	Fast Semi-direct Monocular Visual Odometry
DSO	Direct Sparse Odometry
ORB-SLAM2	Oriented FAST and rotated BRIEF feature-based SLAM 2
ROS	Robot Operating System
CARMEN	Carnegie Mellon Robot Navigation Toolkit
MRPT	Mobile Robot Programming Toolkit
MRDS	Microsoft Robotics Developer Studio
AIST	National Institute of Advanced Industrial Science and Technology
OpenRTM-aist	Open Robot Technology Middleware, implemented by AIST
RT-middleware	Robotics Technology Middleware
OMG	Object Management Group
RT-Components	Robotics Technology Components

Table 1: Lists of abbreviations

Abbreviation	Meaning
Orocos	Open Robot Control Software
RTT	Real-Time Toolkit
OCL	Orocos Component Library
YARP	Yet Another Robot Platform
MOOS	Mission Oriented Operating Suite
MOOS-IvP	MOOS Interval Programming
Rock	Robot Construction Kit
DDS	Data Distribution Service
MRS	Multi-robot systems
MRS UAV system	Multi-robot Systems Group UAV system
FOV	Field of view
ICP	Iterative Closest Point
LOAM	Lidar Odometry and Mapping
NDT	Normal Distributions Transform
MCL	Monte Carlo Localization
A-LOAM	Advanced implementation of LOAM
LeGO-LOAM	Lightweight and Ground-Optimized Lidar Odometry and Mapping
F-LOAM	Fast lidar odometry and mapping
LIO-Mapping	Tightly Coupled 3D Lidar Inertial Odometry and Mapping
LIO-SAM	Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping
PCL	Point Cloud Library
CSM	The C(canonical) Scan Matcher
AICP	Auto-tuned ICP
YAML	YAML Ain't Markup Language
IMU	Inertial measurement unit
SVD	Singular Value Decomposition
6DOF	Six degrees of freedom
3DOF	Three degrees of freedom
LM	Levenberg-Marquardt algorithm
RANSAC	Random sample consensus
FPS	Farthest Point Sampling
RTK	Real-time kinematic positioning

Table 2: Lists of abbreviations