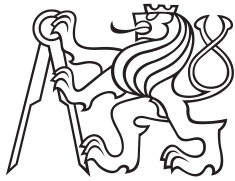


Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Indoor mobile robot localization using up-looking camera

Bc. Eslam Elgourany

Supervisor: Ing. Karel Košnar, Ph.D.

Field of study: Cybernetics and Robotics

Subfield: Cybernetics and Robotics

August 2020

I. Personal and study details

Student's name: **Elgourany Eslam** Personal ID number: **472436**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Control Engineering**
Study program: **Cybernetics and Robotics**
Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Indoor robot localization using up-looking camera

Master's thesis title in Czech:

Lokalizace robotu pro vnitřní prostředí s využitím vzhůru namířené kamery

Guidelines:

1. Study the existing methods for visual localization of mobile robot from up-looking camera
2. Design and implement method for localization of mobile robot in a known map
3. Perform experiments with the real robot
4. Evaluate properties of the implemented method using a referential localization system

Bibliography / sources:

- [1] Mobile Robot Localization using Ceiling Landmarks and Images Captured from an RGB-D Camera, Wen-Tsai Huang, Chun-Lung Tsai and Huei-Yung Lin, International Conference on Advanced Intelligent Mechatronics, 2012
[2] Ceiling-Based Visual Positioning for an Indoor Mobile Robot With Monocular Vision, De Xu, Liwei Han, Min Tan, and You Fu Li, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 56, NO. 5, MAY 2009
[3] CV-SLAM: a new ceiling vision-based SLAM technique, WooYeon Jeong, IROS 2005

Name and workplace of master's thesis supervisor:

Ing. Karel Košnar, Ph.D., Intelligent and Mobile Robotics, CIIRC

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

Date of master's thesis assignment: **13.02.2020** Deadline for master's thesis submission: _____

Assignment valid until: **30.09.2021**

Ing. Karel Košnar, Ph.D.
Supervisor's signature

prof. Ing. Michael Šebek, DrSc.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

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

Date of assignment receipt

Student's signature

Acknowledgements

I wish to express my sincere appreciation to my supervisor, Professor Karel Kosnar, who has the substance of a genius: he convincingly guided and encouraged me to be professional and do the right thing even when the road got tough. Without his persistent help, the goal of this project would not have been realized.

I wish to show my gratitude to Dr. Gaël Ecorchard, who was always offering help in the algorithmic part. Thanks to all the members of the Intelligent and Mobile Robotics department who helped in creating such a great environment for the thesis to take place.

I would like to acknowledge the effort of Vojtěch Pánek who helped a lot in the localization part.

Thanks to my twin sister Dr. Israa Elgourany who invested a lot of her time in the proofreading. She showed patience that cannot be underestimated.

Finally, I am extremely grateful to my father, Ing. Ibrahim Elgourany and my mother. They supported me not only during my stay in the Czech Republic but throughout my life. They kept me going on, and this work would not have been possible without their input with me since I was a child till now. Thanks to my wife for her presence always by my side.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, August , 2020

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, . srpna 2020

Abstract

In this thesis, an effective vision-based system is proposed to accurately track a robot's true position and orientation using an overhead camera attached to the robot and looking on the ceiling. The ground truth is the Vicon system which is a motion capture system. The robot used in the experiment is TurtleBot, which is a low-cost, personal robot kit with open-source software. The selection of suitable visual features and methods is necessary to achieve efficient results. A map of features is created, and a localization algorithm is implemented to locate the robot. The ceiling is chosen as a reference for the algorithm because it is the most stable part of any indoor environment. The camera is chosen in localization because it is the most flexible and low cost approach.

Keywords: features detection, visual localization, particle filter, robotics

Supervisor: Ing. Karel Košnar, Ph.D.
E225b,
Jugoslávských partyzánů 1580/3,
160 00 Prague 6,
Czech Republic

Abstrakt

Tato diplomová práce navrhuje efektivní vizuální systém, který přesně sleduje skutečnou polohu a orientaci robota pomocí stropní kamery připevněné k robotu a snímající strop. K zachycení pohybu je využit systém Vicon. V experimentu byl použit TurtleBot, což je nízkonákladový osobní robot s otevřeným softwarem. K dosažení efektivních výsledků bylo nutné zvolit vhodné nástroje a metody. Na základě experimentů je vytvořena mapa prvků a následně použit lokalizační algoritmus implementovaný k nalezení robota. Jako reference pro algoritmus je vybrán strop, protože je nejstabilnější součástí jakéhokoli vnitřního prostoru. Kamera byla vybrána s ohledem na její flexibilitu a nízkonákladovost.

Klíčová slova: vizuální lokalizace, robotika, detekce příznaků, filtr částic

Contents

List of Abbreviations	1	4.2 Experiments	39
1 Introduction	3	4.2.1 Experiments setup	39
1.1 Motivation	3	4.2.2 Mapping results	39
1.2 Aim and objective of the thesis . .	4	4.2.3 Localization results	39
1.3 Structure of the thesis	4	5 Conclusion and future work	43
2 Localization in mobile robots	5	Bibliography	45
2.1 Mapping definition	5		
2.2 Types of maps	5		
2.2.1 Occupancy grid	5		
2.2.2 Polygonal map	6		
2.2.3 Graph map	7		
2.2.4 Topological map	7		
2.3 Localization definition	7		
2.4 Types of localization algorithms .	8		
2.4.1 Iterative closest point	9		
2.4.2 Kalman filter	10		
2.4.3 Particle filter	11		
2.4.4 Visual SLAM	12		
2.5 Related work	13		
3 Algorithms description	17		
3.1 Software tools	17		
3.1.1 Linux	17		
3.1.2 ROS	17		
3.1.3 Python	18		
3.1.4 Open CV	18		
3.2 Implementation	18		
3.2.1 Camera calibration	18		
3.2.2 Features definition	20		
3.2.3 Features matching	22		
3.3 Mapping algorithm	22		
3.3.1 Projection matrix	22		
3.3.2 Brute force matching	23		
3.3.3 Triangulation of points	24		
3.3.4 Visualizing the map	24		
3.4 Localization algorithm	26		
3.4.1 Particles distribution	26		
3.4.2 Motion model	26		
3.4.3 Sensor model	29		
3.4.4 Resampling method	31		
4 Experimental results	35		
4.1 Hardware configuration	35		
4.1.1 Robot description	36		
4.1.2 Intel NUC computer	36		
4.1.3 Camera	37		
4.1.4 Vicon system	38		

Figures

<p>2.1 G-mapping example [1] 6</p> <p>2.2 Polygon map representation [2] . . 6</p> <p>2.3 Graph map representation [2] . . . 7</p> <p>2.4 Topological Map of Prague metro station 8</p> <p>2.5 General scheme for mobile robot localization [3]. 9</p> <p>2.6 ICP algorithm aligning two scans [4] 10</p> <p>2.7 $\hat{x}_{k k-1}$ denotes the estimate of the system's state at time step k before the $k - th$ measurement y_k has been taken into account; $P_{k k-1}$ is the corresponding uncertainty[5] 11</p> <p>2.8 Example of particle filter result[6] 13</p> <p>2.9 Visual SLAM System[7] 13</p> <p>2.10 Markers used in the related work [8] 14</p> <p>2.11 Iterative closest point registration algorithm [8] 15</p> <p>3.1 Camera calibration patterns 19</p> <p>3.2 Feature detection example[9] . . . 20</p> <p>3.3 AKAZE displayed on the ceiling of the laboratory 21</p> <p>3.4 Features matching between two images 22</p> <p>3.5 Triangulation scheme with non-parallel cameras [10] 24</p> <p>3.6 General architecture of proposed mapping pipeline 26</p> <p>3.7 Motion model[11] 27</p> <p>3.8 Angle γ between the ray to the point (x, y) 27</p> <p>3.9 Gaussian distribution example[12] 28</p> <p>3.10 Sensor model 29</p> <p>3.11 Roulette wheel selection methods 32</p> <p>4.1 Left: The ceiling of the laboratory which was used for the mapping and localization algorithms. Right: The laboratory where the experiments took place. 35</p> <p>4.2 Turtlebot dimensions 36</p>	<p>4.3 NUC Intel computer attached to the robot 37</p> <p>4.4 Basler camera lens 37</p> <p>4.5 Vicon camera and tracker markers 38</p> <p>4.6 Markers attached on the robots appearing on the Vicon tracker software 38</p> <p>4.7 Robot different trajectories 39</p> <p>4.8 Map of features for the rectangular trajectory 40</p> <p>4.9 Map of features for the straight line trajectory 40</p> <p>4.10 Estimated position of the robot from the particle filter 41</p> <p>4.11 Weights assigned to the particles using $W = 0.001 + \frac{N^2}{\sum_i d}$ Sensor model 42</p> <p>4.12 Weights assigned to the particles using $W = 0.001 + N^2$ Sensor model 42</p>
--	---

Tables


4.1 NUC specifications.	37
4.2 Sensor model comparison	41





List of Abbreviations

Abbrevation	Full form
ROS	Robot Operating system.
SLAM	Simultaneous localization and mapping.
RGB	Red Green Blue.
RGB-D	Red Green Blue - Depth.
RFID	Radio Frequency Identification.
IR	Infra Red.
Open CV	Open Source Computer Vision Library.
ICP	Iterative closest point.
NUC	Next Unit of Computing.
GPS	Global Positioning System.
OS	Operating System.



Chapter 1

Introduction

Within this chapter, the general idea which surrounds this thesis will be discussed. It starts with the motivation section, and the aim of the thesis to be fulfilled, and a summary of the thesis structure follows.



1.1 Motivation

Workers in warehouses walk many kilometers everyday carrying some goods from one place to another. Nevertheless, in newer warehouses outfitted with robots, much of that walking has been eliminated. Now companies are using robots to do such tasks.

To create an autonomous robust robotic system that works efficiently, the major requirements that any autonomous robot system needs are localization and planning of the robot. In other meaning, the ability of the robot to know where it is on the map and to think and plan an optimized path with running an avoid collision algorithm besides a frequent updating of its map so that the robot doesn't get lost. Sensors are needed to overcome those challenges. Sensors allow robots to collect data about objects' geometric and physical properties in their surroundings, such as position, orientation, velocity, acceleration, distance, size, force, moment, temperature, weight, etc. The types of sensors used in robotics vary across different applications of robots.

Sensors can be divided into two groups: internal sensors and external sensors. Internal sensors obtain information about the robot itself such as position sensor, velocity sensor, acceleration sensors, motor torque sensor, etc. while external sensors such as cameras, range sensors (IR sensor, laser range finder, and ultrasonic sensor) contact proximity sensors (photodiode, IR detector, RFID, touch, etc.) and force sensors gather information related to the surrounding environment.

■ 1.2 Aim and objective of the thesis

The thesis goal is to design and implement a method for mobile robots in a known map using ceiling monitoring. In underlying meaning, A camera looking at the ceiling will be mounted on a robot, the robot should know the exact position and orientation of itself on the map using only the camera. The camera is chosen as a sensor because it is readily available and not expensive. The ceiling is chosen to be a reference for the algorithms because it is a stable part of any indoor environment.

■ 1.3 Structure of the thesis

Chapter 2, **Localization in mobile robots**: this chapter provides a brief explanation about mapping and localization algorithms. There will be discussed some algorithms that are widely used in the field of navigation.

Chapter 3, **Algorithms description**: this chapter describes the software tools and details the methods and algorithms used to achieve the work presented in the thesis.

Chapter 4, **Experimental results**: this chapter defines the hardware configurations, environment description, and the experiments done. Also, the results of the algorithms proposed at the thesis will be presented to the reader in this chapter.

Chapter 5, **Conclusion and future work**: this chapter describes the conclusion of the experiments and future work that can be done to improve the usability of the proposed system.

Chapter 2

Localization in mobile robots

In this chapter, various mapping and localization algorithms will be briefly explained.

2.1 Mapping definition

Robotic mapping [13] deals with the problem of acquiring spatial models of physical environments through mobile robots. It is one of the most important tasks for robots to achieve. Maps are used for robot navigation like localizing the robot and planning a particular path or task. For a robot to see or define the map, it should have sensors. Sensors can include cameras, range finders using sonar, laser, and infrared technology, radar, tactile sensors, compasses, and GPS. However, all these sensors are subject to errors, often referred to as measurement noise. More importantly, most robot sensors are subject to range limitations.

There are a lot of algorithms and methods that can solve such tasks for robots. One of the most used algorithms is Simultaneous localization and mapping (SLAM) [14]. It is the task of constructing the map and as well keep track of the robot location in it.

2.2 Types of maps

Different types of maps will be presented in this section. Different maps help in many ways, from navigation to establishing ownership, to presenting specific information.

2.2.1 Occupancy grid

Occupancy grid mapping [15] usually addresses the problem of generating maps from noisy and uncertain sensor measurement, with the assumption that the robot pose is known. An example of an occupancy grid shown in Figure 2.1.

The goal of an occupancy mapping algorithm is to estimate the posterior probability that the space is occupied given the data:

$$p(m \mid z_{1:t}, x_{1:t}) \quad (2.1)$$

where m is the map, $z_{1:t}$ is the set of measurements from time 1 to t , and $x_{1:t}$ is the set of robot poses from time 1 to t .

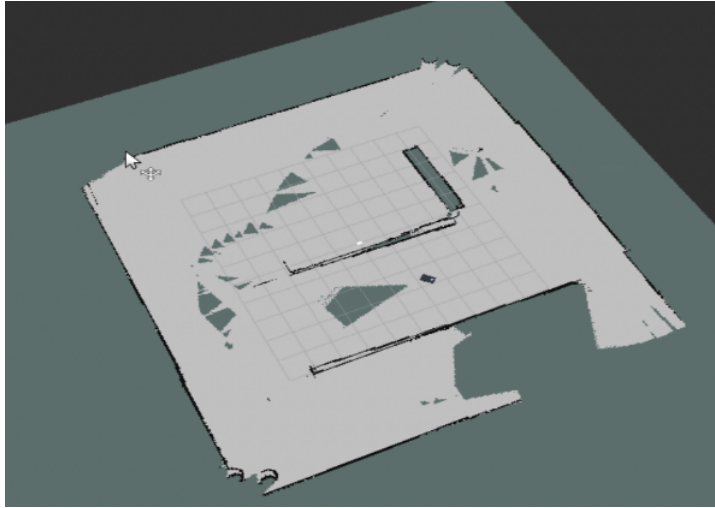


Figure 2.1: G-mapping example [1]

2.2.2 Polygonal map

Polygonal maps are one of the most common representations alternative to occupancy grids. It is represented in the shape of information that consists of a collection of vertices, edges, and faces. If the movement between large areas is uniform, then instead of following a grid, a polygonal map representation can be used. The polygons can either present free spaces or obstacles.

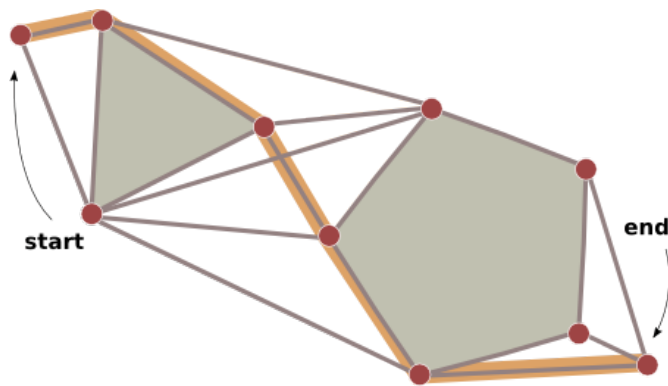


Figure 2.2: Polygon map representation [2]

2.2.3 Graph map

Usually, a graph map is formed from many connected and internally disjoint regions of the euclidean plane. The graph maps are more general and don't include many details, as shown in Figure 2.3.

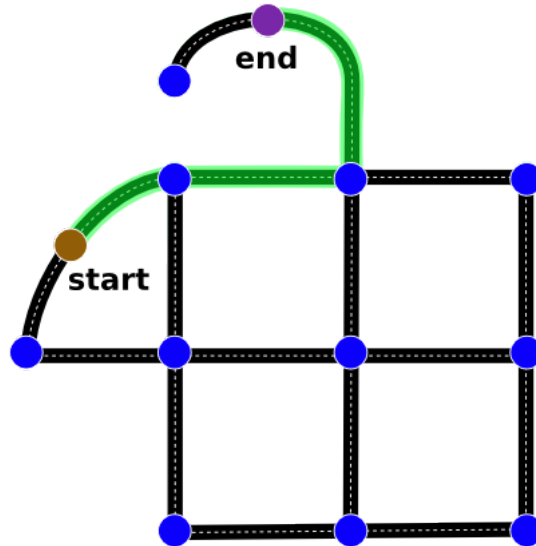


Figure 2.3: Graph map representation [2]

2.2.4 Topological map

It is a type of simplified diagram that includes only vital information and excludes other unnecessary details. These maps lack scale, whereas distance and direction are subject to changes, but the relationship between points is maintained. The name topological map [16] is derived from topology. It is the branch of mathematics that studies the properties of objects that do not change as the object is deformed. An example of a topological map is the Prague metro map, which is represented in Figure 2.4.

2.3 Localization definition

Robot localization [17] means the robot's ability to determine its position in its frame of reference. For the robot to navigate in its environment, the robot requires information to react with, for example, a map of the environment and the ability to understand that map. The localization of a mobile robot denotes the robot's ability to establish its position and orientation within the frame of reference.

Localization is one of the most crucial competencies needed by an autonomous robot as the robot's location is a necessary precursor to forming decisions about future actions. In a typical robot localization situation, a map of the environment is available. The robot is provided with sensors that

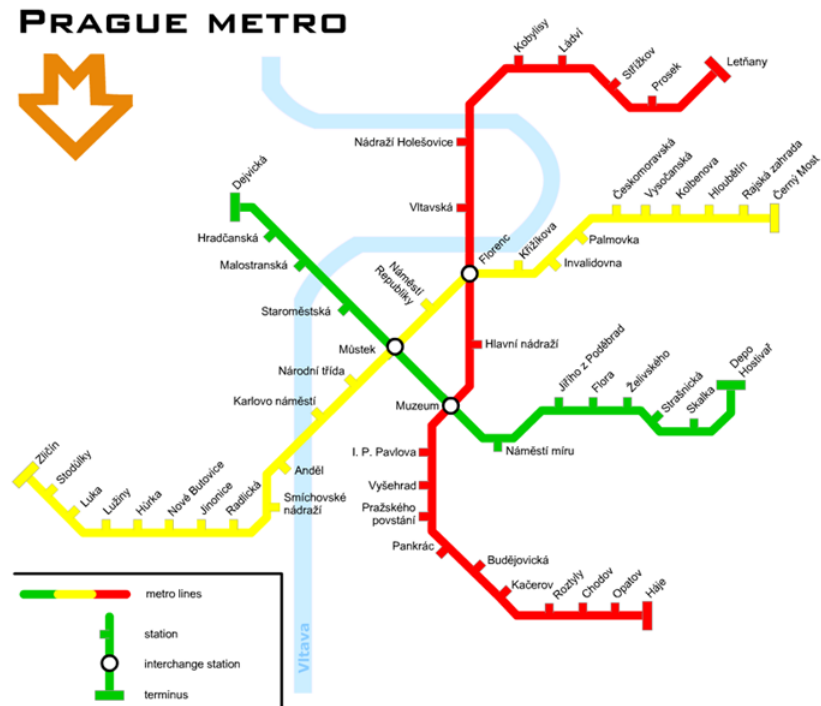


Figure 2.4: Topological Map of Prague metro station

observe the surroundings and observe its own motion. The localization difficulty then becomes one of determining the robot position and orientation within the map using these sensors' information. Robot localization methods need to be able to deal with noisy observations and defining not only an estimate of the location of the robot but also to define the uncertainty of the position estimation.

2.4 Types of localization algorithms

There are several methods for localizing the robot. Several factors can influence the algorithms or the methods used; some of them are mentioned below.

- Localization in static and dynamic environment.

There are no moving objects around the robot in a static environment, and if there is, it will be following a specific trajectory without changing it with time. The environment is constant for the robot. However, in a dynamic environment, the robot should consider other objects that are not part of the map and can move or change their properties with time.

- Localization in known and unknown environment.

In a known environment, the robot would localize itself using the map given as input to it. However, for the unknown environment, the robot

must build a map by itself and use it for its own localization like in SLAM [18].

- Relative and absolute localization.

Relative localization [19] determines the robot position based on the previous sensor measurement. On the other hand, absolute localization estimates the position without any need for the previous data measurement as it can rely on external sensors.

- Passive and active localization.

In passive localization, the robot receives data from the sensors and estimates its position without influencing the robot's behavior. In the active localization, the robot behavior can be affected by the sensors to do active sensing like, for example, moving the camera in some specific direction to discover a particular area.

The general idea of robot localization is shown in Figure 2.5 . In the next sections, there will be introduced the principles of some localization algorithms in robotics.

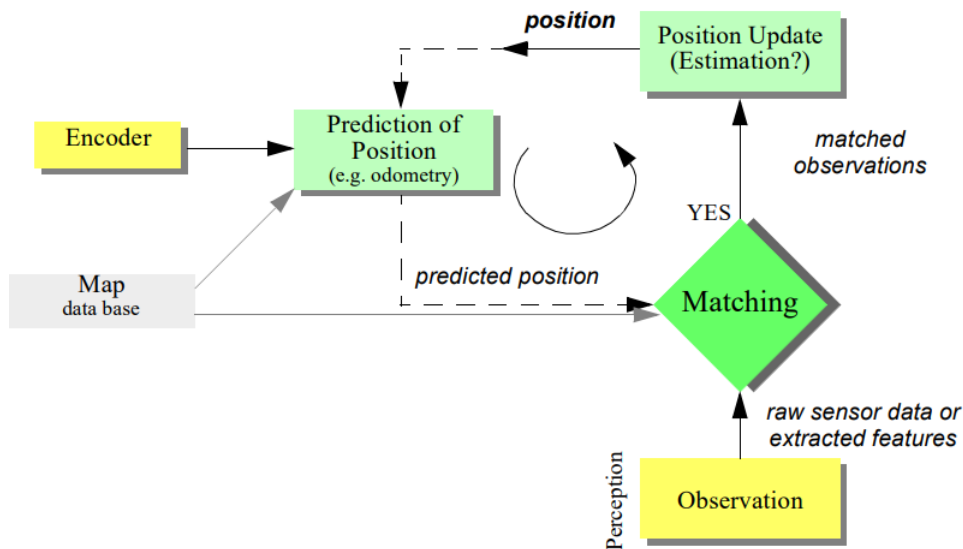


Figure 2.5: General scheme for mobile robot localization [3]

■ 2.4.1 Iterative closest point

Iterative closest point (ICP) [20] is an algorithm that is used to minimize the difference between two clouds of points. Usually, the map is known, and it is kept fixed as a reference for the robot. The laser sensor attached to the robot provides readings of the surroundings with some specific range. Initially, the sensor readings that represent the borders of the map and edges are not

matching with the reference map, as shown in Figure 2.6(a). ICP algorithm wishes to align the two scans.

The ICP algorithm always checks three steps: association, transformation, and error evaluation. These are repeated until the scans are aligned satisfactorily [4] as shown in Figure 2.6(b).

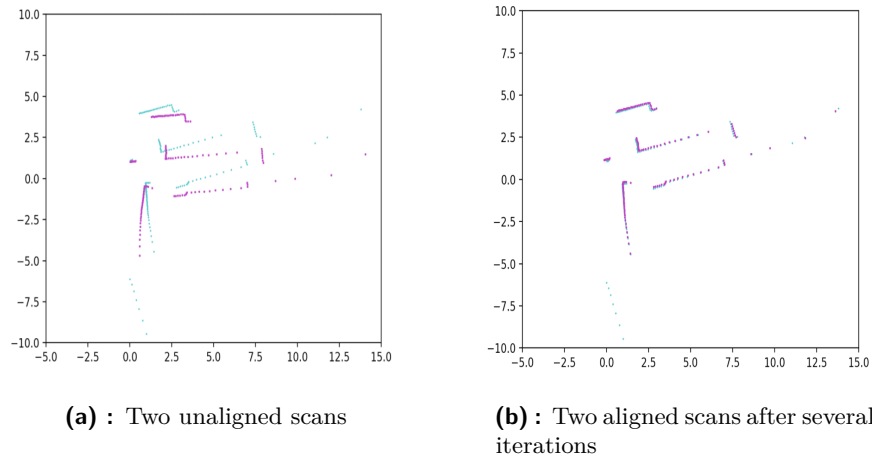


Figure 2.6: ICP algorithm aligning two scans [4]

2.4.2 Kalman filter

Kalman filter [21] is an optimal estimator which is often used for mobile robot localization. It is used in any application where uncertain information about the system exists. It makes an educated guess about what the system is going to do next. Kalman filters are ideal for systems that are continuously changing. They do not need to keep any history other than the previous state, and they are very fast, making them well suited for real-time problems and embedded systems.

The algorithm depends on the sensors' measurements, which can be used to observe the surroundings like cameras or laser sensors. It also relies on the movement of the robot from one state to another concerning time. The Kalman filter combines the uncertainties regarding the current state of the robot and the sensor measurements. Ideally, the uncertainty of the robot should be decreased. A Gaussian probability distribution represents both uncertainties. Thus, the mean represents what value of the distribution has the highest probability to be accurate, and the variance expresses how uncertain the measurements are concerning this mean value.

The filter keeps track of the system's estimated state and the variance or uncertainty of the estimate. The estimate is updated using the robot movement and sensor measurements. The scheme of the Kalman filter is shown in Figure 2.7.

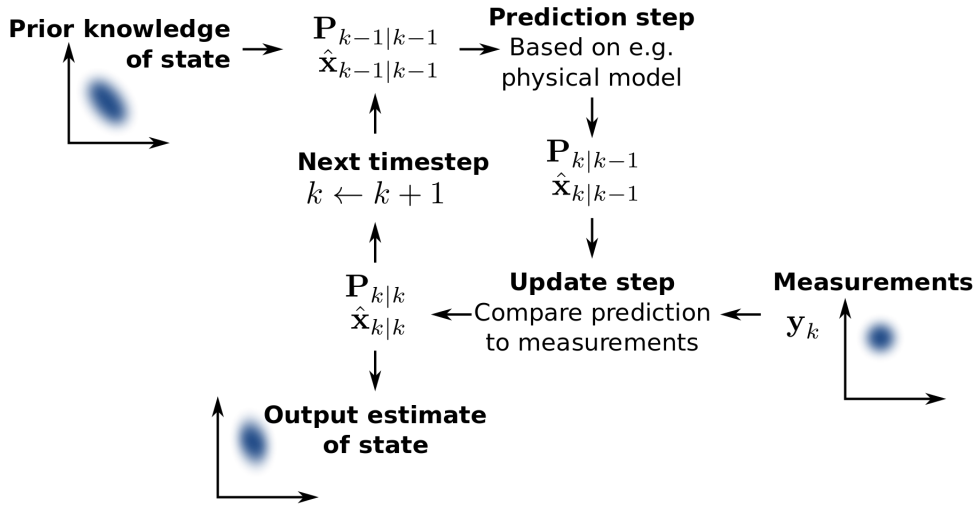


Figure 2.7: $\hat{x}_{k|k-1}$ denotes the estimate of the system’s state at time step k before the k -th measurement y_k has been taken into account; $P_{k|k-1}$ is the corresponding uncertainty[5]

2.4.3 Particle filter

A particle filter is an algorithm for estimating the state of a dynamic system. It takes the current belief to be updated based on motion information and control commands along with observations from the sensors. It has a prediction step and a correction step in order to estimate how the states develop.

The algorithm creates a large number of particles in which each particle position represents a possible belief of where the robot is. The particles are scattered uniformly over the map only in the beginning. So if ten thousand particles exist, it means that there are ten thousand hypotheses where the robot can be in the given space. The number of particles is an input that can be defined by the user. Particles can be distributed differently if prior information exists.

Each particle is assigned a weight; the density of the particles represents the distribution of probability. Initially, the weight is $1/N$ for N particles. $1/N$ is used so that the sum of all probabilities is equal to one.

The next part of the algorithm is to predict the step using the control commands. Suppose the robot received a command to move 0.3 meters while turning by 0.005 radians. It could be possible to move every particle with this amount, But an error is expected in such a case because the controls of the robot are not perfect, so the robot would not move exactly as commanded. Therefore, adding noise to the particle’s movements is necessary to have a better chance of determining the robot’s actual movement.

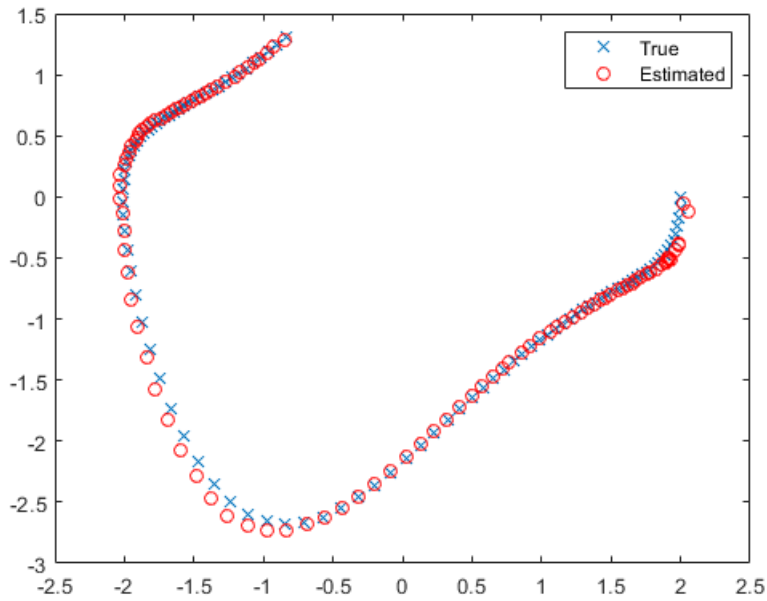


Figure 2.8: Example of particle filter result[6]

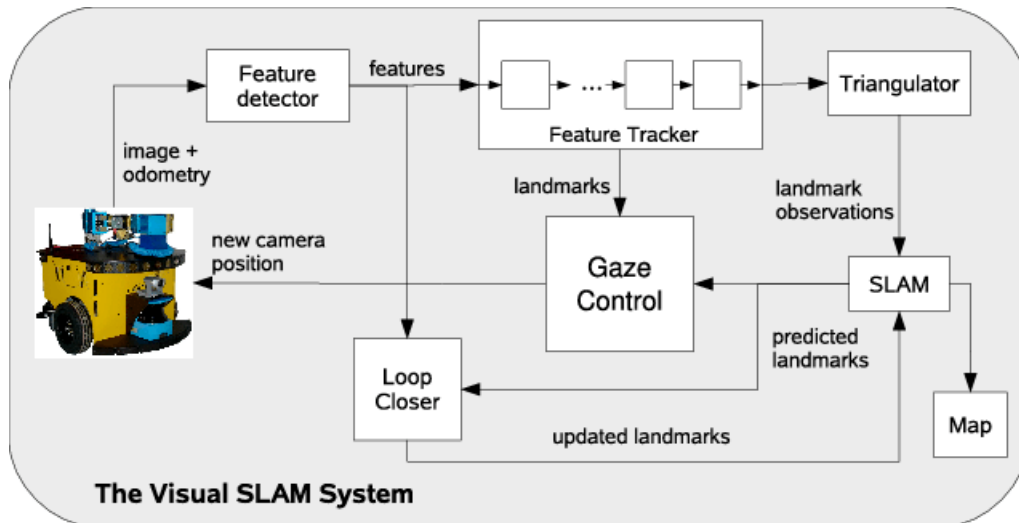


Figure 2.9: Visual SLAM System[7]

2.5 Related work

Wen-Tsai Huang, Chun-Lung Tsai, and Huei-Yung Lin proposed two mobile robot localization techniques for the indoor environment [8]. First, some images of some markers attached to the ceiling with known positions to calculate the robot's location and orientation, like a global method. Second, an RGB-D camera mounted on the robot is adapted to acquire the color and

depth images of the environment, like a local method.

Regarding the first method, the marker used in the proposed system was a black square pattern containing few different combinations of solid white circles. The marker is designed to provide unique localization information. This is achieved by assigning three out of four corners as solid white circles, as shown in Figure 2.10. When the marker image was captured by the camera, it was immediately identified without the influence of the camera's viewpoint. Furthermore, the positions of the corner circles are used to rectify the image and calculate the rotation information of the robot. As for the other circles in the pattern, they are used to indicate and distinguish the locations of the markers in the environment. Since the acquired images are colored and the markers are designed as black and white patterns, they are converted to grayscale images and then binarized for further processing. The color information is also filtered to avoid false detection. Sobel edge detection algorithms [22] are adopted to extract the markers from the white ceiling on the background.

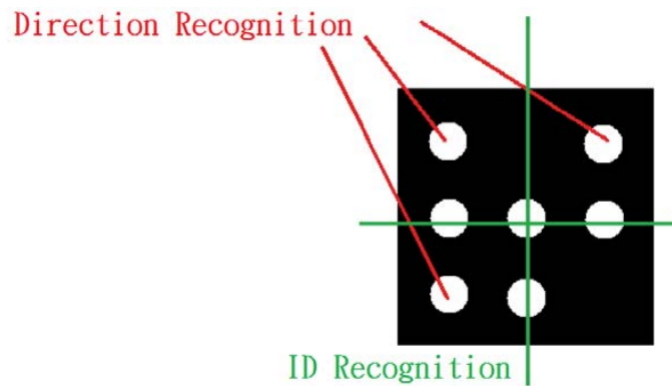


Figure 2.10: Markers used in the related work [8]

In the second method [8] they used an RGB-D camera. An RGB-D camera refers to a camera system that can capture the colored image (Red- Green-Blue) and the associated depth map simultaneously. It usually consists of a color digital camera and a range sensor that is capable of providing the depth information of the scene. Given two sets of 3D points, the iterative closest point algorithm was used to find their relative rotation and translation by registering the two data sets in the same coordinate system. In this algorithm, one data set was defined as model, and the other was defined as data. The objective is to fit the data points to model points on the overlapping part and find the transformation. As shown in Figure 2.11, the points marked in red and green in the left figure are the model and data, respectively, prior to registration. On the right figure, the data points (marked in blue) were transformed and overlapped with the model points after carrying out the ICP registration algorithm. Feature matching algorithm was also implemented as counting only on the registration of 3D point clouds acquired by the range

sensor might be inaccurate due to the noise of insufficient overlapping parts. More details can be accessed from the reference mentioned above.

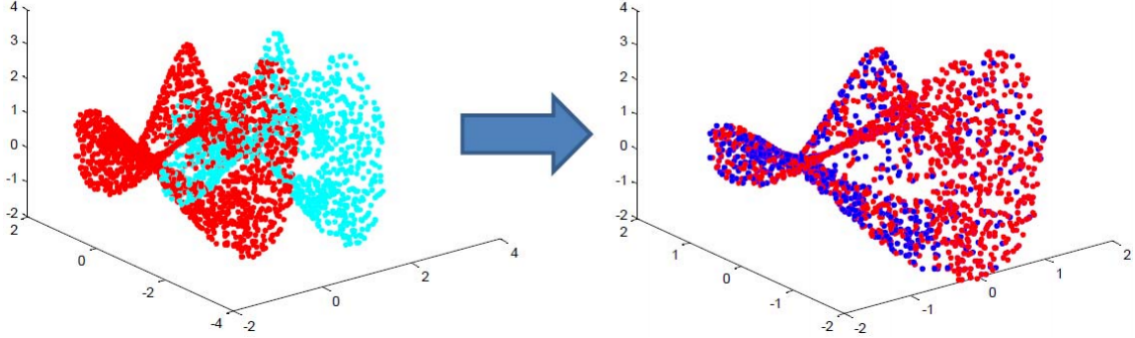


Figure 2.11: Iterative closest point registration algorithm [8]

Another work [23], De Xu, Liwei Han, Min Tan, and You Fu Li dealt with parallels and corner points on the ceiling as it can serve features for visual positioning for an indoor mobile robot. Based on the natural features on the ceiling, a new visual positioning method is proposed. A camera is mounted on the top of the mobile robot and pointed to the ceiling. At the beginning of visual positioning, the initial orientation and position of the mobile robot in the world frame is estimated by a specified block on the ceiling via (PnP) perspective-n-point-based positioning method. With the motion of the mobile robot, its global orientation is calculated from the main and secondary lines feature when the ceiling has parallels. In other cases, its global orientation is estimated by point features on the ceiling. Then, its position is recursively computed with the point features. The error analysis and experiments verify the effectiveness of this method.

Another researchers [24], WooYeon Jeong and Kyoung Mu Lee, illustrated a new Ceiling Vision-based SLAM technique. Fast and robust CV-SLAM (Ceiling Vision-based Simultaneous Localization and Mapping) technique using a single ceiling vision sensor. The proposed algorithm is suitable for a system that demands very high localization accuracy, such as an intelligent robot vacuum cleaner. A single-camera looking upward direction (called ceiling vision system) is mounted on the robot, and salient image features are detected and tracked through the image sequence. The ceiling vision has an advantage in tracking since it involves only rotation and affine transform without scale change. Moreover, in this paper, the researchers solved the rotation and affine transform problems using a 3D gradient orientation estimation method and a multi-view description of landmarks. By applying these methods to the solution for data association, the 3D landmark map was constructed in realtime through the Extend Kalman filter based SLAM framework. Furthermore, the relocation problem was solved efficiently by using a wide baseline matching between the reconstructed 3D map and a 2D

ceiling image. Experimental results demonstrate the accuracy and robustness of the proposed algorithm in real environments.

Chapter 3

Algorithms description

3.1 Software tools

In this section, the software tools, including the operating system and libraries used in the thesis, will be described. Also, the methods of proposed algorithms will be explained in details.

3.1.1 Linux

Linux [25] is a family of open source operating systems. Popular Linux distributions include Debian, Fedora, and Ubuntu. Ubuntu 14.04 distribution was installed on the computer of the robot and used as an interface for implementing the algorithms.

3.1.2 ROS

Robot Operating System (ROS) [26] is a collection of frameworks for robot software development. It provides services designed for various computer clusters such as hardware abstraction, low level device control, message passing between processes, and package management. Running sets of ROS based processes are represented in a graph architecture where processing takes place in nodes that can receive, post, and multiplex various kinds of messages.

The most used distributions of ROS are Indigo, Kinetic, Lunar and Melodic. They can be used mainly on Unix operating systems such as Ubuntu or Mac OS. ROS allows implementation in the most modern programming languages such as Python, C++, Lisp, Java, and Lua. ROS Indigo was installed on the (Next Unit of Computing) NUC PC on the robot.

One of the core properties in ROS is passing messages between nodes. This is done by a Publisher-Subscriber architecture, where a publisher node creates a topic (or uses an existing one), and all nodes subscribed to this topic receive its messages. Topics are the buses in which nodes can exchange messages through it. One topic can be both published into and subscribed by multiple nodes.

One important feature in the ROS is the ROS bag [27] which is a backup of data from ROS messages. It is like a container used for storing the data sent between sensors at a specific time interval. This feature is very practical when implementing an algorithm because it can be recorded once and then played back to simulate a specific scenario as much as needed. The ROS bag file format is very efficient for recording and playback, as messages are stored in the same representation used in the network transport layer of ROS. ROS bags were used during the experiment.

■ 3.1.3 Python

Python programming language [28] is used in the thesis to install the packages needed and for the implementation of the algorithms.

■ 3.1.4 Open CV

OpenCV (Open Source Computer Vision Library) [29] is a collection of open-source libraries for computer vision and machine learning. It has more than 47,000 users besides a lot of worldwide researchers. OpenCV libraries contain more than 2500 algorithms and methods supporting the field of computer vision and machine learning. OpenCV was built to provide an infrastructure for computer vision applications and accelerate the use of machine perception in commercial products.

These algorithms include 2D and 3D feature tool kits, gesture recognition, object identification, segmentation and recognition, structure from motion, motion tracking, augmented reality, Human-computer interaction, etc. This interface supports C++, Python, Java, Matlab, and a wide range of operating systems such as Windows, Linux, Android, and Mac OS. There is also an active community forum for troubleshooting.

Several libraries from OpenCV are used in the thesis, like camera calibration which can be described in details here ([30], [31]), triangulation of points [32] and, feature detectors and matchers [33].

■ 3.2 Implementation

In this section, the implementation of the algorithms used in the thesis, will be described.

■ 3.2.1 Camera calibration

Camera calibration is the process of establishing the correct parameters of the camera taking images during the calibration process. These parameters are focal length, format size, principal point, and lens distortion. The camera should be calibrated to achieve higher accuracy and low distortion, which helps in achieving the most accurate representation of the real world in the

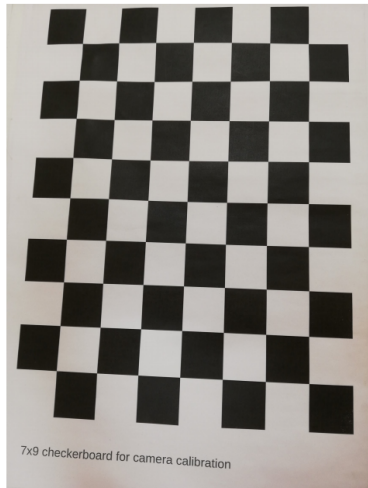
captured images. It is an essential step in the computer vision pipeline because many subsequent algorithms require camera parameter knowledge as an input. Shown in Equation 3.1 a representation for the camera matrix, which is the output of the calibration process. The result of the camera calibration was added to the camera configuration file.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

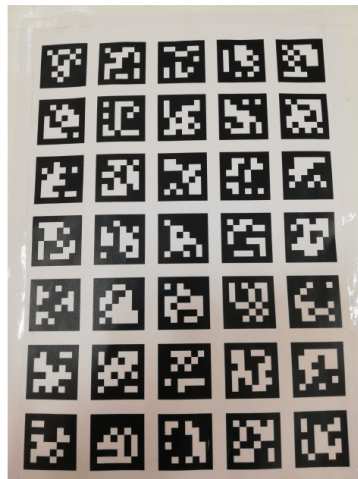
where,
 f_x and f_y are the focal lengths expressed in units of pixels.
 c_x and c_y are the principle points that are at the image center.

Chessboards are used for camera calibration as they are simple to construct, and the structure of their planar grid defines many natural interest points in an image. Camera calibration package was used which is implemented in the Open CV library.

Several pictures were taken for the chessboard pattern from different angles. The implemented calibration script was run. The values obtained by this calibration were proved to be inaccurate. Several colleagues with a similar type of camera had dramatically different values. So a different calibration method was chosen, which was using April tag calibration [34]. April tag calibration method evaluates the calibration process online after each frame. It gives hints about how the camera should be oriented for the next frames and guides the user for better calibration matrix as well as distortion coefficients. Shown in Figure 3.1 the patterns used in the calibration process.



(a) : Chess pattern used for camera calibration



(b) : April tag patterns for camera calibration

Figure 3.1: Camera calibration patterns

The result of the calibration is as follows:

$$\text{Camera matrix} = \begin{bmatrix} 1089.811186 & 0.000000 & 642.894805 \\ 0.000000 & 1093.843267 & 478.331197 \\ 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

$$\text{Distortion coefficients} = [0.205646 \quad 0.039258 \quad 0.003634 \quad 0.001089 \quad 0.000000]$$

3.2.2 Features definition

Image features are the unique places in any image frame. They correspond to local regions in the image and are fundamental in many applications in image analysis like recognition, matching, and reconstruction of 2D images. Image features illustrate two different types of problems: the detection of an area of interest in the image, and the classification of local regions in the image, typically for matching in different images.

The best way to find the features is to look for the regions in images which have maximum variation when moved in all regions around it. Finding these image features is called feature detection. In the Figure 3.2, the blue patch is a flat area and difficult to find and track. Wherever the blue patch is moved, it looks the same. The black patch has an edge, if it is moved in the vertical direction (i.e., along the gradient), it changes, however, if it is moved along the edge (parallel to edge), it will look the same. And for the red patch, it is a corner. Wherever the patch is moved, it looks different, which means it is unique[9].

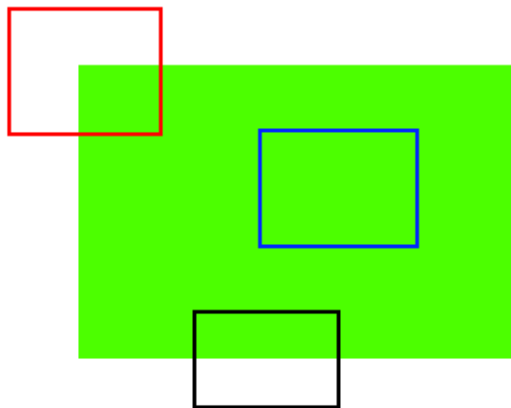


Figure 3.2: Feature detection example[9]

The computer can process such features' definition using a feature descriptor. It is an algorithm that takes an image and outputs feature vectors.

Feature descriptors encode interesting information into a set of numbers and act as a kind of numerical fingerprint that can be used to distinguish individual features from another.

Open CV provides several feature descriptors like SIFT [35], ORB [36], AKAZE [37], SURF [38] and more. A comparison between several features descriptors is described in this article ([39]). AKAZE descriptor will be used according to the previous results achieved in the master thesis done by our colleague Ing. Jiří Koktan in CIIRC [40]. His thesis proved that it is effective and the most proper to use with such systems.

Shown in Figure 3.3, the camera's frame after visualizing the AKAZE feature descriptor. It is obvious that it detects unique parts in the ceiling like corners of the light frames, edges of the ventilator, and the corners of the ceiling patterns.

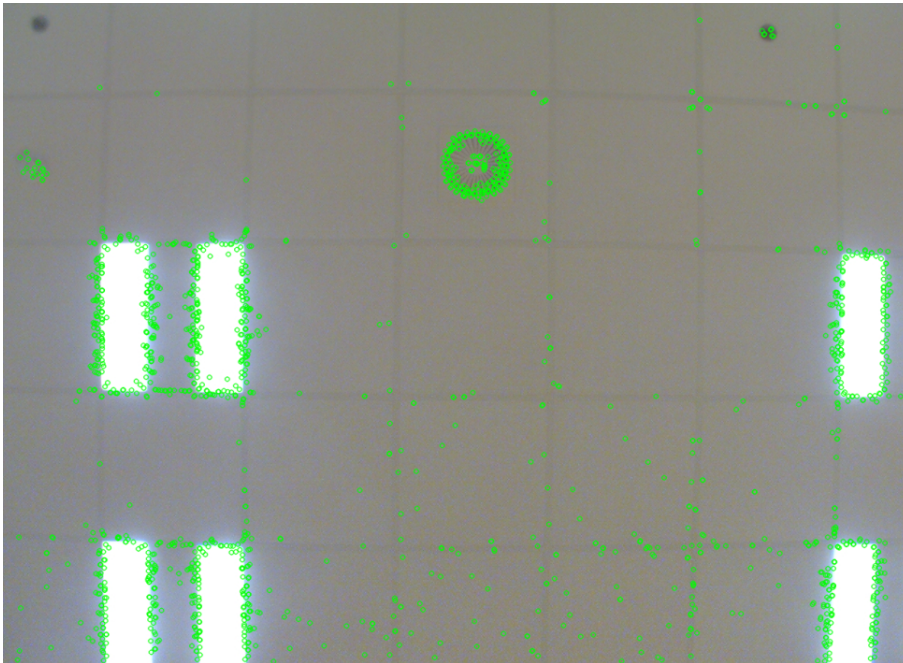


Figure 3.3: AKAZE displayed on the ceiling of the laboratory

In order to illustrate more what those features represent, we can pick one of those detected features on the ceiling and analyze it. A keypoint and a descriptor define such features. Keypoint can be called as an interesting point. It has a spatial location or point in the image that defines what is interesting or what stands out in the image. What makes keypoints different between frameworks is the way those keypoints are described. These are what are known as descriptors. Each detected keypoint has an associated descriptor that accompanies it. A descriptor is represented in the form of finite vector which summarizes properties for this keypoint.

The keypoints are special as no matter how the image changes, whether the image rotates, shrinks, or expands, or even is translated; it will be still

possible to find the same keypoints in this modified image when comparing with the original image as explained in Section 3.2.3.

■ 3.2.3 Features matching

Open CV offers libraries for matching the features in two images. The algorithm is based on comparing and analyzing point correspondences between the reference image and the target image, as shown in Figure 3.4. If any part of the image shares similarities greater than some specified threshold then that part of the image is targeted and considered to include the reference object [41]. In the thesis, the Brute-Force matcher algorithm [42] is used.

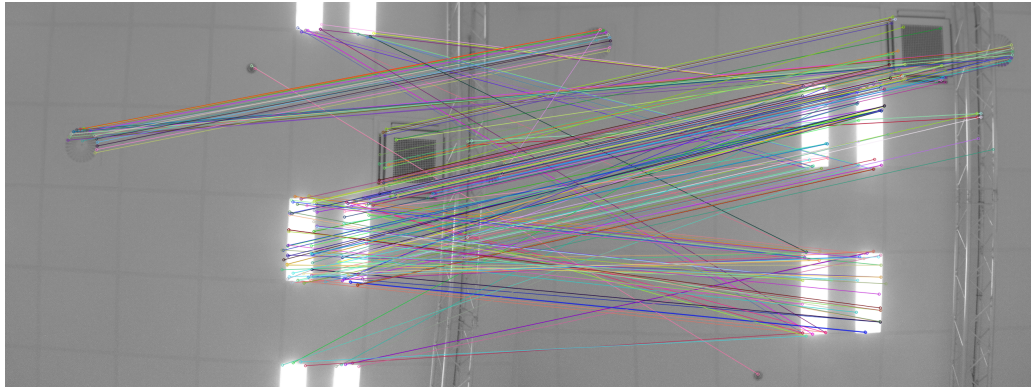


Figure 3.4: Features matching between two images

■ 3.3 Mapping algorithm

In this section, we will illustrate how the mapping of the features detected on the ceiling will be achieved. The mapping is necessary to be included as an input for the localization algorithm. The robot will be moving in a specific trajectory, and the mapping algorithm will be doing the calculations during the robot movement.

■ 3.3.1 Projection matrix

A projection matrix describes the mapping of some point from 3D world coordinates to 2D image coordinates. It is necessary to calculate two projection matrices, one for the previous camera frame, and the second is for the current camera frame. They were calculated by using equation 4.2.

$$P = K \times [R|t] \quad (3.2)$$

where,

P is output 3x4 projection matrix.

K is input 3x3 camera matrix.

R is input 3x3 rotation matrix.

T is input 3x1 translation vector.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.3)$$

where,
 f_x and f_y are the focal lengths expressed in units of pixels.
 c_x and c_y are the principle points that are at the image center.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (3.4)$$

where the rotation matrix is a sequence of three rotations, everyone around each principle axis.

$$t = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (3.5)$$

where t is the position vector of the robot.

■ 3.3.2 Brute force matching

Brute-Force KNN (k-Nearest Neighbors) matching algorithm takes the descriptor of one feature in the first camera frame and tries to match it with all other features in the second camera frame recursively with respect to some distance threshold which is given as an input, and the closest one is returned. The performance of the algorithm for the task is quite effective, and no issues are encountered. The pseudocode is shown at Algorithm 1.

Algorithm 1: Brute Force KNN Algorithm

inputs : Q , a set of query points and R , a set of reference point;
output : A list of points (k) reference points for each query;
1 **foreach** $querypoint(q) \in Q$ **do**
2 Compute distances between q and all r in R
3 Sort the computed distances;
4 Select the nearest k reference points corresponding to k smallest distances;
5 **end foreach**

3.3.3 Triangulation of points

Triangulation is the idea of identifying the position of a point by forming triangles to it from other known points, as shown in Figure 3.5. It refers to determining a point in 3D space by knowing its projections onto a certain number of images.

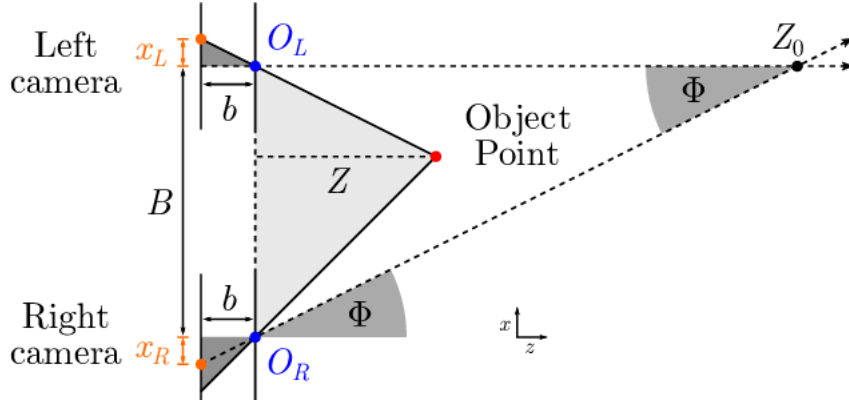


Figure 3.5: Triangulation scheme with non-parallel cameras [10]

$$\text{Triangulation} = \text{Fn}(\text{ProjMat1}, \text{ProjMat2}, \text{ProjPoints1}, \text{ProjPoints2}) \quad (3.6)$$

where,

ProjMat1 is a 3x4 projection matrix of the previous camera frame.

ProjMat2 is a 3x4 projection matrix of the current camera frame.

ProjPoints1 is a 2xN array of feature points of previous camera frame.

ProjPoints2 is a 2xN array of corresponding matched feature points in the current camera frame.

3.3.4 Visualizing the map

A trajectory is done by the robot while the camera mounted on the robot is looking at the ceiling. The current position is known from the Vicon system as a ground truth. When the robot moves a distance of 30 centimeters, the algorithm starts to process the calculations, and the known position becomes the previous position. In contrast, the current position is still tracked via the Vicon system. The projection matrices are calculated, brute force matching between the features of the first and second frame is done, putting into consideration filtering the matches according to some given distance threshold, and the triangulation function is applied. Finally, the output of the triangulate function is plotted. Shown in Algorithm 2 the pseudocode for the mapping algorithm.

At line 16, the triangulate function is the output needed to be visualized. The output represents the coordinates of the features in X , Y , and Z directions besides the orientation of the features w . From lines 20 till 23, swapping

Algorithm 2: Mapping Algorithm

```

input : Vicon system node.
         Camera node.
         Robot movement with keyboard.
output : Map of features.
1 if distance between two consecutive frames > 30 cm then
2   current keypoints, descriptors = DetectAndcompute AKAZE
   Features(image)
3   if previous keypoints, descriptors = None then
4     Previous keypoints= Current keypoints
5     Previous descriptors = Current descriptor
6   end if
7   current translation matrix = Matrix(x,y,z)
8   Current rotation matrix = Rotation (Orientation(x, y, z, w))
9   Current projection matrix = camera matrix . [Rotation matrix |
   Translation matrix]
10  Brute Force Matching Algorithm (Previous Frame, current
   Frame)
11  foreach match in Brute Force Matching do
12    if match.distance < 50 then
13      Take those "good" matches ;           // Filter the matches
14      current projection points = current keypoints for m in
        good matches
15      previous projection points = previous keypoints for m in
        good matches
16      Triangulate (Projection matrices, Projection points)
17      Plot(Triangulate)
18    end if
19  end foreach
20  Previous position = Current position
21  Previous orientation = Current orientation
22  Previous Keypoints = Current Keypoints
23  Previous descriptors = Current descriptors
24 end if

```

of the variables is necessary so that after doing the calculations, the robot will consider its current position as a previous position and then move, and the "real" current position will be accessed from the ground truth, which is the Vicon system. The algorithm is repeated recursively until the robot does not receive any new observation from the Vicon system or the camera, or if the robot did not move a distance of 30 cm. The general architecture of the mapping algorithm can be seen in Figure 3.6.

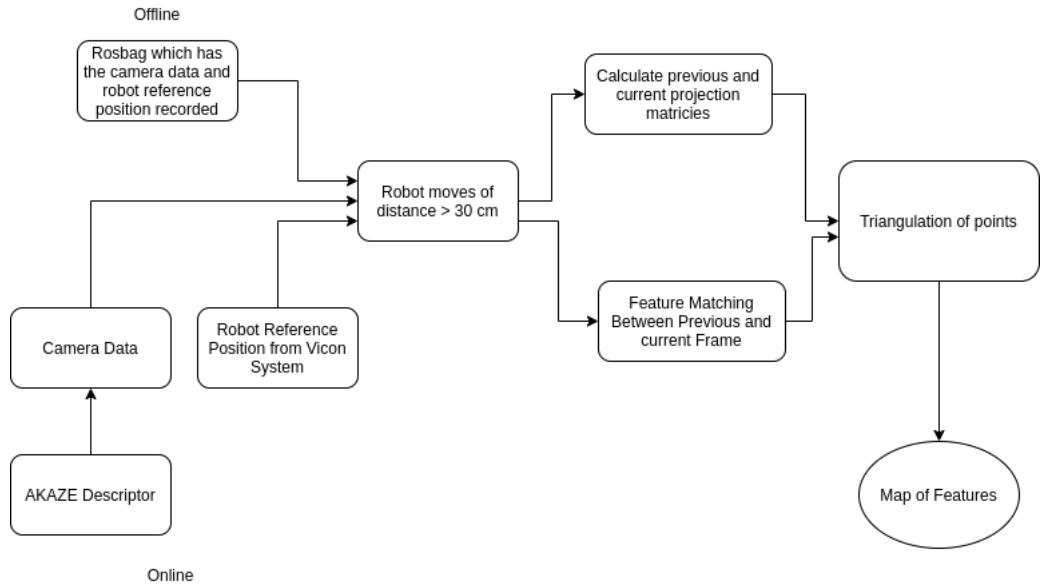


Figure 3.6: General architecture of proposed mapping pipeline

3.4 Localization algorithm

The particle filter is chosen to be used in the localization part. In the next section, the implementation of the particle filter will be explained.

3.4.1 Particles distribution

The algorithm starts with a uniform random distribution of particles around all the map.

3.4.2 Motion model

The motion model ensures the movement of the particles according to the odometry. Noise is added to the motion, to represent any possible irregular deviations resulting from the robot movement. The noise added is represented using four parameters, α_1 , α_2 , α_3 , and α_4 .

$$\alpha_1 = 0.004 \quad (3.7)$$

$$\alpha_2 = 0.004 \quad (3.8)$$

$$\alpha_3 = 0.4 \quad (3.9)$$

$$\alpha_4 = 0.004 \quad (3.10)$$

The values of these parameters for the Turtlebot robot were determined on the subject B3M33MKR and appeared to give a reasonable noise for the

odometry. An example of how the motion model works is shown in Figure 3.7.

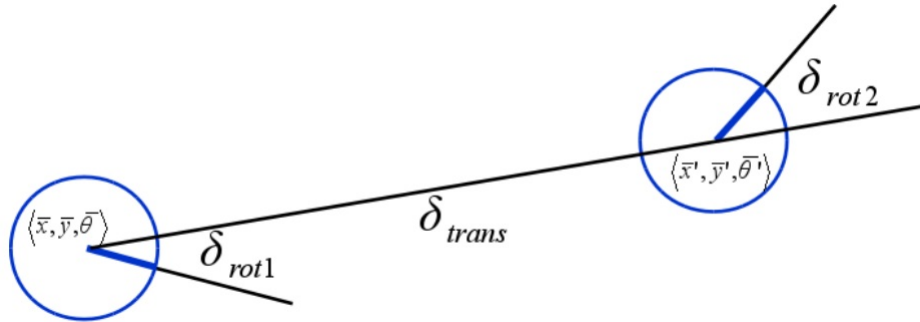


Figure 3.7: Motion model[11]

In Figure 3.7,
 $\langle \bar{x}, \bar{y}, \bar{\theta} \rangle$ represents the position and orientation of the robot at $time_1$.
 $\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$ represents the position and the orientation of the robot at $time_2$.
 δ_{trans} is distance between two successive positions of the robot.
 δ_{rot1} is the orientation of the robot around the Z axis in the first position at $time_1$.
 δ_{rot2} is the orientation of the robot around the Z axis in the second position at $time_2$.

In order to calculate δ_{rot1} , it is necessary to calculate an angle γ using arctan 2 function which is presented in Figure 3.8. The calculations for the γ , δ_{trans} , δ_{rot1} and δ_{rot2} are shown in equations (3.11, 3.12, 3.13, 3.14) respectively.

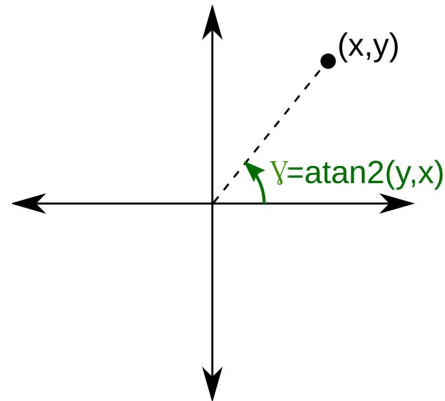


Figure 3.8: Angle γ between the ray to the point (x, y)

$$\gamma = \arctan 2((\bar{y}' - \bar{y}), (\bar{x}' - \bar{x})) \quad (3.11)$$

$$\delta_{trans} = \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2} \quad (3.12)$$

$$\delta_{rot_1} = \gamma - \theta_1 \quad (3.13)$$

$$\delta_{rot_2} = \bar{\theta}' - \bar{\theta} - \delta_{rot_1} \quad (3.14)$$

A Gaussian distribution is applied with $\mu = 0$, as the *sample* part in the equations (3.15, 3.16, 3.17) represents the standard deviation σ of the normal distribution. Example of the distribution with $\mu = 0$ is shown in Figure 3.9.

$$\hat{\delta}_{rot_1} = \delta_{rot_1} + \text{sample}(\alpha_1|\delta_{rot_1}| + \alpha_2|\delta_{trans}|) \quad (3.15)$$

$$\hat{\delta}_{trans} = \delta_{trans} + \text{sample}(\alpha_3|\delta_{trans}| + (\alpha_4(|\delta_{rot_1}| + |\delta_{rot_2}|))) \quad (3.16)$$

$$\hat{\delta}_{rot_2} = \delta_{rot_2} + \text{sample}(\alpha_1|\delta_{rot_2}| + \alpha_2|\delta_{trans}|) \quad (3.17)$$

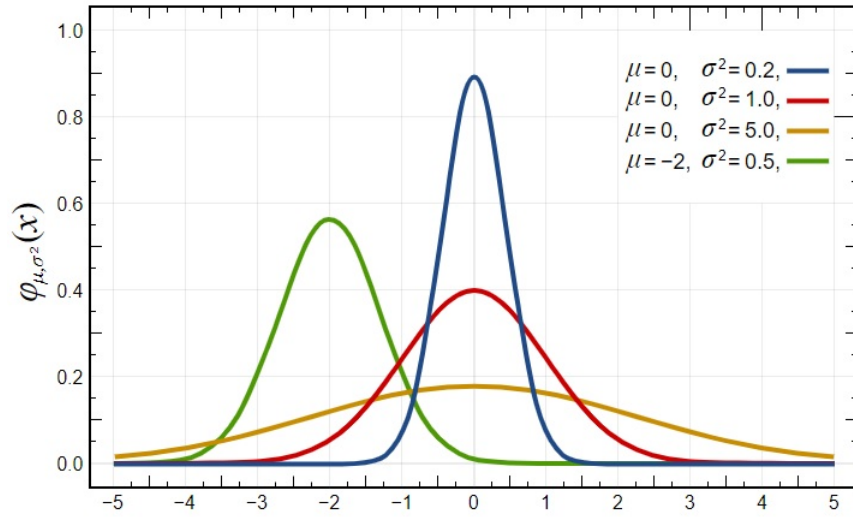


Figure 3.9: Gaussian distribution example[12]

Finally, we add the previous calculated values to the position of the particles as shown in equations (3.18, 3.19, 3.20).

$$x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot_1}) \quad (3.18)$$

$$y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot_1}) \quad (3.19)$$

$$\bar{\theta}' = \theta + \hat{\delta}_{rot_1} + \hat{\delta}_{rot_2} \quad (3.20)$$

where,
 x is the current position of particle in X direction.

y is the current position of particle in Y direction.

θ is the current heading of particle.

The $\hat{\delta}_{rot_1}$ and $\hat{\delta}_{rot_2}$ represent the orientation of the robot at $time_1$ and $time_2$ respectively after adding the noise factors to them as well as $\hat{\delta}_{trans}$ which is the distance that the robot moves. x' , y' and θ' are the coordinates of the particle and its orientation respectively after adding to it noise as well.

3.4.3 Sensor model

When the robot observes the environment using the sensors attached to it, it updates its particles to more accurately reflect where it is, depending on a specific sensor model. The sensor model determines the measured data reliability and assigns weights to the particles according to that. In the proposed algorithm, the only sensor used is the camera looking at the ceiling. So, after the distribution of the particles, each particle will be assigned a virtual camera. The virtual cameras will have the same camera matrix and resolution as the real one. The particles would scan some particular area of the ceiling and see which features exist in its camera frame. The area that the particle covers is equal to the same area that the real camera covers. The camera resolution is 1.2 Megapixels, which means that the camera frame covers a width of 1280 pixels and a height of 960 pixels. A demonstration of how the sensor model works is shown in Figure 3.10.

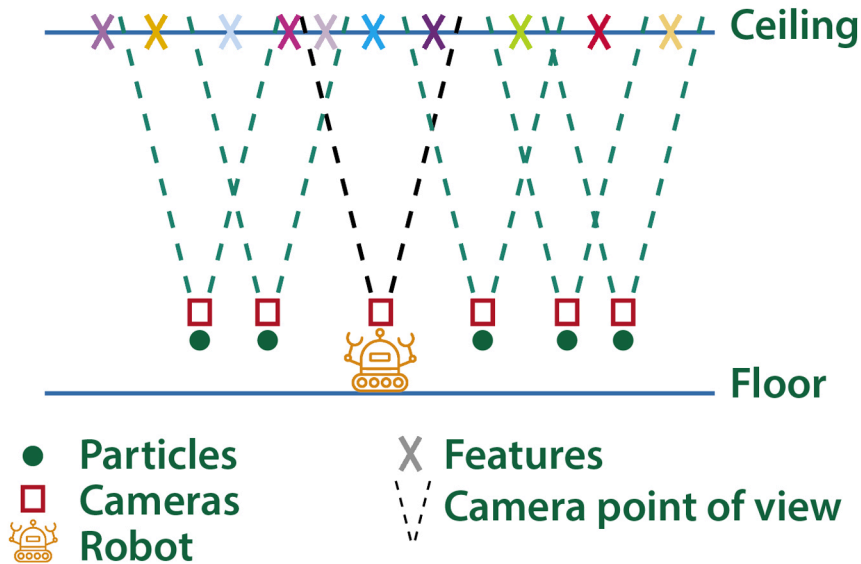


Figure 3.10: Sensor model

In Figure 3.10, the green dots are the particles distributed in the robot

environment. The red squares represent the cameras, and each camera covers the same area on the ceiling.

Projection points of each particle were calculated as shown in the following equations.

$$s \begin{bmatrix} u \\ v \\ w \end{bmatrix} = P \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (3.21)$$

where,

$$P = K \times [R|t] \quad (3.22)$$

More detailed equation,

$$s \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & t_1 \\ \sin\theta & \cos\theta & 0 & t_2 \\ 0 & 0 & 1 & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (3.23)$$

where,

(u, v) are the coordinates of the projection point in pixels.

(f_x, f_y) are the focal lengths expressed in pixel units.

(c_x, c_y) are the principal points that is usually at the image center.

(θ) is the heading angle of the particle.

(t_1, t_2, t_3) are the x, y and z coordinates of the particle respectively.

(X, Y, Z) are the coordinates of a 3D point in the world coordinate space which refer to the features coordinates.

A brute force matching algorithm is applied to match the descriptors of the features that the virtual cameras on the particles can see with the real camera's current descriptors in real time. Naively, if those descriptors are close enough, then the particles which can see that feature will be close to the robot position and should be assigned a high weight.

From the mapping algorithm. A file is saved which has the coordinates of all the features. The file structure is shown in 3.24.

$$\text{Mapping output file} = \begin{bmatrix} X_1 & Y_1 & Z_1 & W_1 & Descriptor_1 \\ X_2 & Y_2 & Z_2 & W_2 & Descriptor_2 \\ X_3 & Y_3 & Z_3 & W_3 & Descriptor_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & W_n & Descriptor_n \end{bmatrix} \quad (3.24)$$

where,

X is the position of the feature in X direction.

Y is the position of the feature in Y direction.

Z is the position of the feature in Z direction.

W is the heading of the feature.

Descriptor is the descriptor of the feature. It is an array of shape $(M \times 61)$.

n is the maximum number of features.

From the equation, $s \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = P \times \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$, the output u_i and v_i should cor-

respond in the triangulate file to the *descriptor_i* which is related to X_i and Y_i and try to match this descriptor with the current seen descriptors by the camera. A distance threshold is defined for the algorithm so that if it is not within the specified range, the feature will be discarded and try to match another one so that the closest feature will be returned. The distance is set to be as minimum as possible, so that it only returns the very similar features. A weight is assigned to the particles according to the formula 3.25,

$$W = 0.001 + \frac{N^2}{\sum_i d} \quad (3.25)$$

where,

W is the weight of the particle.

N is the number of matches.

d is an Euclidean distance between the matched features.

This sensor model shows that the W is $\propto \frac{1}{d}$, which means if the distance between the matched features decreases the weight will increase which refers to a higher probability that the particle is somewhere very close to the robot. Other sensor model will be tested and evaluated in Chapter 4.

3.4.4 Resampling method

Re-sampling is the method of replacing unlikely particles with the most likely ones. There are several ways to do such a process. One of the used algorithms is the roulette wheel selection. This algorithm is used in applications to select an item proportional to its probability. Imagining a roulette wheel and the size of the pockets are proportional to the weight of each particle, as shown in Figure 3.11.

Several ways can be used with such an algorithm. In the first method of the roulette wheel selection algorithm as shown in Figure 3.11(a), during the wheel spinning if the pointer ends up pointing at pocket where for example w_3 is, then it will be picked and put in a new sample set. This process is repeated n times. It can be described as a binary search algorithm.

$$O(n \log n) \quad (3.26)$$

So if we have a million particles, the search will be represented in the equation as $10^6 \log 10^6$, which means it is computationally demanding but can still work if the number of particles is not that huge.

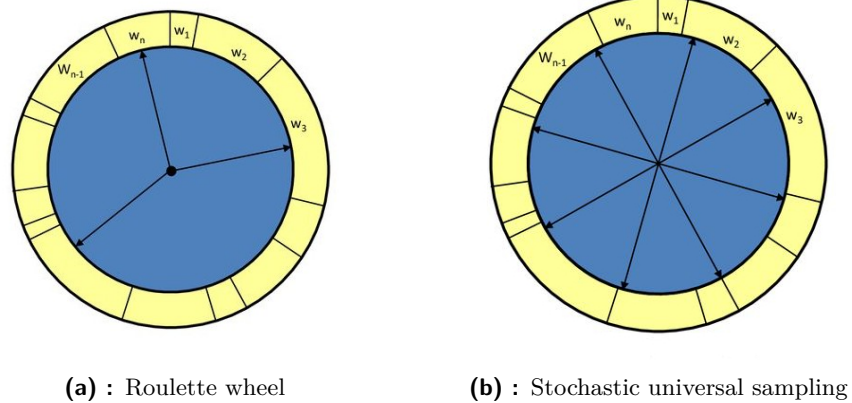


Figure 3.11: Roulette wheel selection methods

However, Stochastic universal sampling takes a uniform spacing of the pointers used to point to the pockets. When we spin the roulette wheel once, we pick the samples which correspond to the pocket at which the pointers are pointing, as shown in Figure 3.11(b). It is named as a low variance resampling. Its advantage is that it can be done in a linear time, which is not computational demanding.

$$O(n) \tag{3.27}$$

Another advantage is that it can help if a set of samples have the same weight, this can happen if the sensor observation does not help much to identify which sample is better than the other one, in such case, it will guarantee to obtain precisely the same sample set as that we had before. As if the samples have the same weight, it is not necessary to do the resampling and better to keep the same samples as it is. Shown in Algorithm 3 the low variance resampling algorithm which is used as a part of the localization algorithm in the thesis.

Line 3 in Algorithm 3 shows the drawing of the random number in the interval between $0; M^{-1}$. However, the while loop selects the particles by repeatedly adding the fixed amount of M^{-1} to r by choosing the particle that corresponds to the resulting number.

After the resampling method, the position of the robot is estimated and can be known. The localization algorithm pseudocode is shown in Algorithm 4.

Line 4 in Algorithm 4 shows that each particle is a hypothesis as to what the true world state may be at time t , However at line 5, the importance weight incorporates the measurement into the particle set. Normalizing of the weights are necessary, and then the resampling algorithm is applied.

Algorithm 3: Low variance resampling algorithm

```

1 Resample ( $X_t, W_t$ ):
2    $\bar{X}_t = \phi$ 
3    $r = \text{rand}(0; M^{-1})$ 
4    $c = w_t^{[1]}$ 
5    $i = 1$ 
6   for  $m = 1:M$  do
7      $U = r + (m-1) M^{-1}$ 
8     while  $U > c$  do
9        $i = i + 1$ 
10       $c = c + w_t^{[i]}$ 
11    end while
12    add  $x_t^{[i]}$  to  $\bar{X}_t$ 
13  end for
14  return  $\bar{X}_t$ 

```

Algorithm 4: Localization Algorithm

```

1 Particle Filter ( $X_{t-1}, u_t, z_t$ ):
2    $\bar{X}_t = X_t = \phi$ 
3   for  $i = 1:N$  do
4      $x_t^{(i)} \sim p(x_t | x_{t-1}^{(i)}, u_t)$            //Motion Model
5      $w_t^{(i)} \sim p(z_t | x_t^{(i)})$            //Sensor Model
6      $\bar{X}_t = \bar{X}_t + \langle x_t^{(i)}, w_t^{(i)} \rangle$ 
7   end for
8   for  $i = 1:N$  do
9      $w_t^{(i)} = t^{-1} w_t^{(i)}$            //Normalize
10  end for
11  Resample using algorithm 3:
12   $\{x_t^{(i)}, w_t^{(i)}, -\}_{i=1}^N = \text{RESAMPLE}[\{x_t^{(i)}, w_t^{(i)}\}_{i=1}^N]$ 
13  return  $X_t$ 

```

Chapter 4

Experimental results

In this chapter, the hardware configuration, experiments setup, and the results of the algorithms will be explained.

4.1 Hardware configuration

The experiments and implementation environment took place in the Czech Institute of Informatics, Robotics and Cybernetics (CIIRC) on the third floor in the laboratory of Intelligent and Mobile Robotics. The laboratory and ceiling are shown in Figure 4.1, where the experiments took place.



Figure 4.1: Left: The ceiling of the laboratory which was used for the mapping and localization algorithms. Right: The laboratory where the experiments took place.

4.1.1 Robot description

The robot used is called TurtleBot [43]. TurtleBot is a low-cost, personal robot kit with open-source software. TurtleBot was created at Willow Garage by Melonee Wise and Tully Foote in November 2010.

The robot has a flat cylindrical shape with a diameter of 354 mm and a height of 89 mm, as shown in Figure 4.2. Its weight is approximately 6.3 kg, and the maximum load capacity is 5 kg. Furthermore, the robot is equipped with a motor for every wheel and one wheel in the middle for stability. Turtlebot is able to rotate in its place up to 180. Its maximum driving speed is limited to 0.65 meters per second.

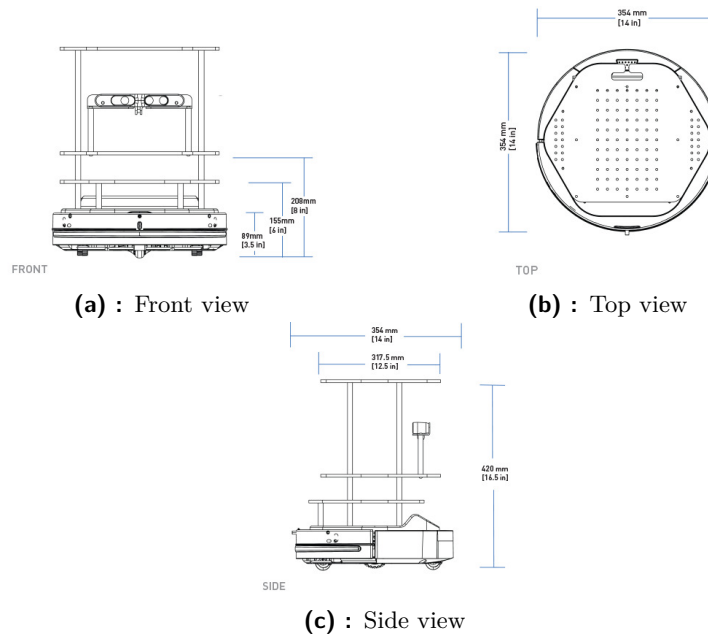


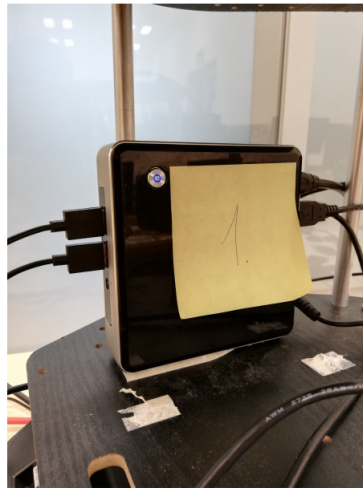
Figure 4.2: Turtlebot dimensions

4.1.2 Intel NUC computer

The central control unit of the robot is the Intel NUC5i5RYK computer, as shown in Figure 4.3. Ubuntu 14 and ROS Indigo were installed on the (Next Unit of Computing) NUC. This computers main advantage is its small size and weight, so it is very suitable for being placed on Turtlebot. The power supply to the NUC is taken from the base of the robot. The specifications of the computer are shown in Table 4.1.



(a) : NUC Intel computer



(b) : NUC attached to the Robot

Figure 4.3: NUC Intel computer attached to the robot

Processor	Intel Core i5-5250U
Graphics Card	Intel HD Graphics 6000
RAM	16 GB
Storage capacity	230 GB
Ports	4x USB 3.0, mini HDMI 1.4a, mini Display Port 1.2, 3,5mm Jack

Table 4.1: NUC specifications.

4.1.3 Camera

The camera used in the experiments is the Basler daA1280-54uc (S-Mount). It is shown in Figure 4.4. It delivers 54 frames per second at 1.2 MP resolution.

**Figure 4.4:** Basler camera lens

4.1.4 Vicon system

The Vicon system [44], shown in Figure 4.5 is a motion capture system that can be used to record and analyze motion. The system is used in robotics, film and gaming industry, and several medical fields. The goal is to obtain an accurate record of the robot's position, orientation, and movement.

The system covers is attached to the ceiling and it covers all the laboratory. It has a specific scene from several perspectives. The cameras are synchronized with each other emitting rays that react with some reflective markers to create an image of a moving object. Those markers are attached to the robot during its movement. It operates using an individual PC in the laboratory. Vicon Tracker software is used to visualize the position of the markers. The Vicon system has a relatively high accuracy, which is about a tenth of a millimeter. A study of Vicon system positioning performance is accessible on this reference [45].

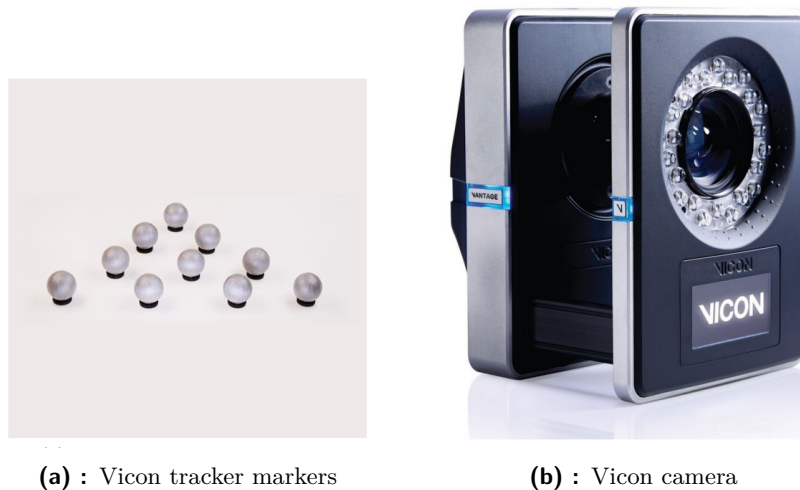


Figure 4.5: Vicon camera and tracker markers

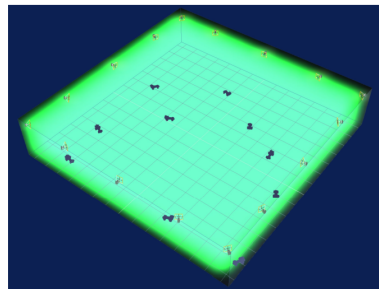


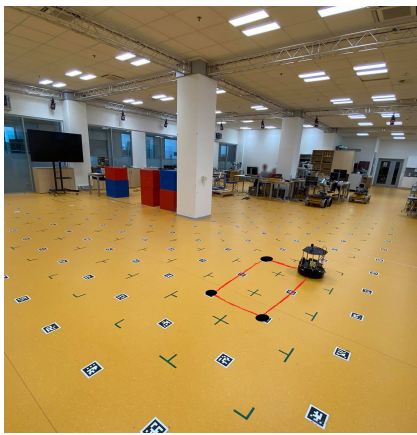
Figure 4.6: Markers attached on the robots appearing on the Vicon tracker software

4.2 Experiments

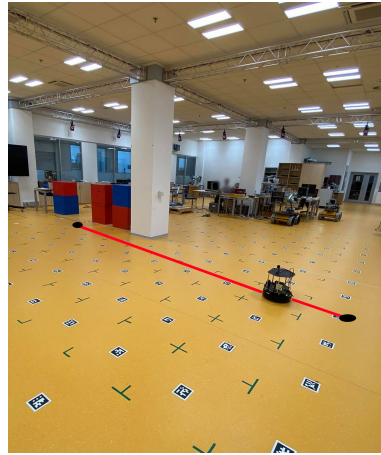
In this section, we will drive the robot in different trajectories and test the mapping and localization algorithms.

4.2.1 Experiments setup

Two different trajectories are performed. The first trajectory is a rectangular shape, and the second one is in the form of a straight line as shown in Figures (4.7a, 4.7b) respectively.



(a) : Rectangular trajectory



(b) : Straight line trajectory

Figure 4.7: Robot different trajectories

4.2.2 Mapping results

The 2D and 3D visualizations of the rectangular trajectory are shown in Figure 4.8. The results look as expected because the features are forming a rectangle. However, some features appear out of the map range, and this can happen because of some external factor that was captured via the camera during the motion of the robot, for example, random light source. These features are filtered, and the rest of the features are considered as a valid map.

The output of the straight-line trajectory is shown in Figure 4.9. The robot was moving under repetitive panels of light. At the end of the trajectory, a circular ventilator appeared in the camera frame, which can be observed in Figure 4.1. The results reflect the real movement of the robot as well.

4.2.3 Localization results

This section shows the results of the particle filter. Different sensor models will be evaluated. The particles formed clusters around the robot position

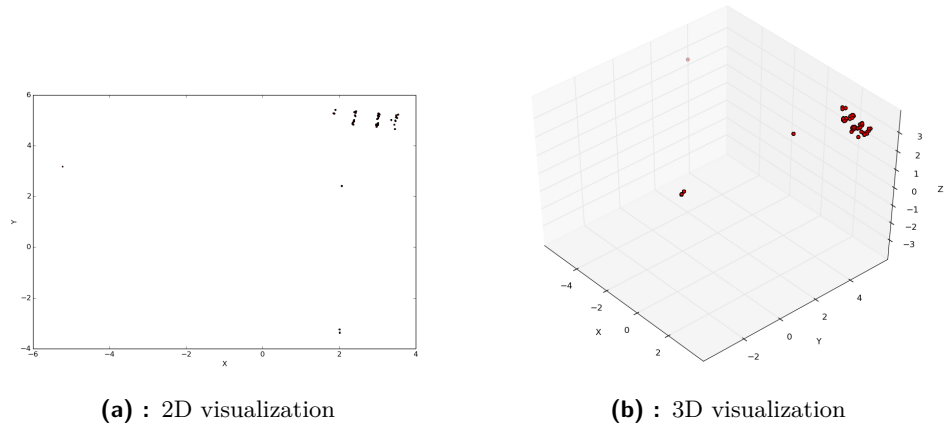


Figure 4.8: Map of features for the rectangular trajectory

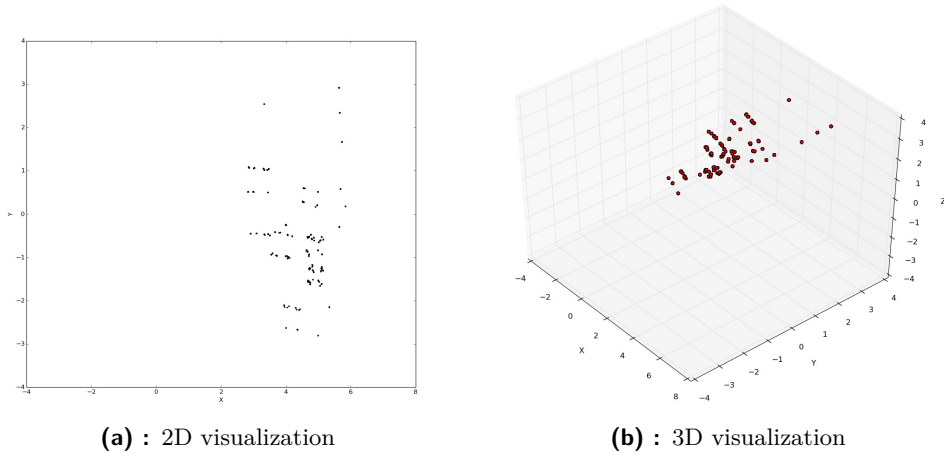


Figure 4.9: Map of features for the straight line trajectory

as shown in Figure 4.10.

However, evaluation of different sensor models is necessary to see how robust the algorithm will estimate the robot position in a long time span as the particles could deviate with time from the robot position especially if there are no matches found between the descriptors that the particles and the camera can detect. Different sensor models were tested.

$$W = 0.001 + \frac{N^2}{\sum_i d} \quad (4.1)$$

$$W = 0.001 + N^2 \quad (4.2)$$

where,
 W is the weight of the particle.
 N is the number of matches.
 d is the euclidean distance between the matches.

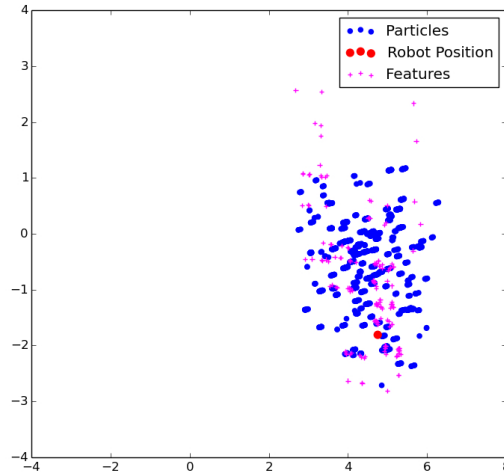


Figure 4.10: Estimated position of the robot from the particle filter

A comparison between them is shown in Table 4.2.

Sensor Model	$W = 0.001 + \frac{N^2}{\sum_i d}$	$W = 0.001 + N^2$
Usability	Thanks to this sensor model, the weights are assigned high values in the area where the robot is, which helped a lot in keeping track of the robot, the highly weighted particles are all accumulated close to the robot position as shown in Figure 4.11, this is because the W is $\propto \frac{1}{d}$, given N is known number of matches and $d \neq 0$	Unlike the other sensor model, this model proved to assign high weights even for the particles which are far from the robot as shown in Figure 4.12. This can cause a wrong estimate of the robot position with time.
Relationship between variables		

Table 4.2: Sensor model comparison

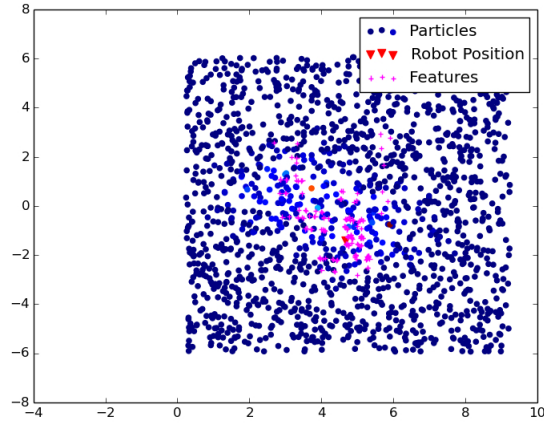


Figure 4.11: Weights assigned to the particles using $W = 0.001 + \frac{N^2}{\sum_i d}$ Sensor model

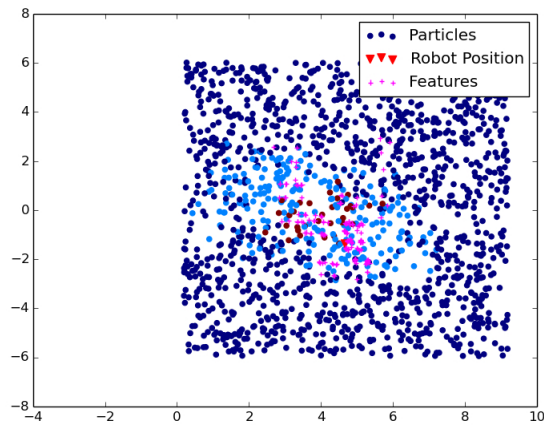


Figure 4.12: Weights assigned to the particles using $W = 0.001 + N^2$ Sensor model

Chapter 5

Conclusion and future work

The thesis has proposed a practical method for the localization of a robot using a camera looking at the ceiling. The principles and methods of mapping and localization algorithms were described. OpenCV libraries were used in camera calibration, features detecting and matching, and triangulation of points. A particle filter was used in the estimation of the robot position. A comparison between different sensor models was accomplished. The work was established in Python programming language in the ROS environment.

For the validation of the system, two experiments were accomplished, as described in Section 4.2. It was concluded that the mapping algorithm is effective even with such a repetitive ceiling as it was, in the laboratory, the visualized output reflected the landmarks which were seen by the camera, also, with prior knowledge regarding the X, Y, and Z coordinates of the arena from the Vicon system, the features appeared in the correct coordinates.

Clusters of particles were formed around the robot's actual position and keep track of the robot movement. A comparison of two sensor models was achieved, and it was proved that the sensor model, which includes the Euclidean distance between the matches as a factor, is more effective. Moreover, the evaluation reveals that the particles with higher weights are more concentrated near the robot position, which lead to better accuracy in estimating the robot coordinates.

As a Future work, Another programming language can be used as the particle filter is a demanding computational algorithm. Using another programming language that is faster in execution than Python; for example, C++ can lead to a more reliable system. Secondly, testing the algorithm in different environments will be beneficial for evaluation. Finally, a clustering algorithm can be implemented so that instead of having clusters around the robot, the algorithm can choose one particle only to track.



Bibliography

- [1] M. Arrua, “Exploring ros with a 2 wheeled robot gmapping.” www.theconstructsim.com/exploring-ros-with-a-2-wheeled-robot-13-gmapping, 2019.
- [2] Amit Patel, “Map representations.” <http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html#polygon-movement>.
- [3] R. Siegwart, EPFL, Illah Nourbakhsh, “Autonomous mobile robots.” <http://www.cs.cmu.edu/~rasc/Download/AMRobots5.pdf>.
- [4] Andrewjkramer, “Lidar odometry with icp.” <http://andrewjkramer.net/lidar-odometry-with-icp/>, 2019.
- [5] Wikipedia contributors, “Kalman filter — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Kalman_filter&oldid=967968225, 2020.
- [6] Mathwork help center contributors, “particlefilter.” <https://www.mathworks.com/help/control/ref/particlefilter.html>.
- [7] S. Frintrop, P. Jensfelt, and H. Christensen, “Attentional robot localization and mapping,” 06 2007.
- [8] W. Huang, C. Tsai, and H. Lin, “Mobile robot localization using ceiling landmarks and images captured from an rgb-d camera,” in *2012 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 855–860, 2012.
- [9] OpenCV Maintainers, “Understanding features.” https://vovkos.github.io/doxyrest-showcase/opencv/sphinx_rtd_theme/page_tutorial_py_features_meaning.html.
- [10] C. Hahne, A. Aggoun, V. Velisavljevic, S. Fiebig, and M. Pesch, “Baseline and triangulation geometry in a standard plenoptic camera,” *International Journal of Computer Vision*, pp. 1–15, 08 2017.

- [11] G. E. Karel Kosnar, Miroslav Kulich, “Intelligent and mobile robotics group, czech technical university in prague, b3m33mkr, winter semester,” 06 2019.
- [12] Wikipedia contributors, “Normal distribution — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Normal_distribution&oldid=967927766, 2020.
- [13] Sebastian Thrun, “Robotic mapping: A survey.” <http://robots.stanford.edu/papers/thrun.mapping-tr.pdf>, 2002.
- [14] A. R. Khairuddin, M. S. Talib, and H. Haron, “Review on simultaneous localization and mapping (slam),” in *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pp. 85–90, 2015.
- [15] A. Elfes, “Using occupancy grids for mobile robot perception and navigation,” *Computer*, vol. 22, no. 6, pp. 46–57, 1989.
- [16] Wikipedia contributors, “Topological map — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Topological_map&oldid=908021828, 2019.
- [17] S. Huang and G. Dissanayake, *Robot Localization: An Introduction*, pp. 1–10. American Cancer Society, 2016.
- [18] Wikipedia contributors, “Simultaneous localization and mapping — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Simultaneous_localization_and_mapping&oldid=945301892, 2020.
- [19] P. Goel, S. I. Roumeliotis, and G. S. Sukhatme, “Robot localization using relative and absolute position estimates,” 1999.
- [20] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pp. 145–152, 2001.
- [21] Q. Li, R. Li, K. Ji, and W. Dai, “Kalman filter and its application,” in *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, pp. 74–77, 2015.
- [22] H. Xiang, B. Yan, Q. Cai, and G. Zou, “An edge detection algorithm based-on sobel operator for images captured by binocular microscope,” in *2011 International Conference on Electrical and Control Engineering*, pp. 980–982, 2011.
- [23] D. Xu, L. Han, M. Tan, and Y. F. Li, “Ceiling-based visual positioning for an indoor mobile robot with monocular vision,” *IEEE Transactions on Industrial Electronics*, vol. 56, no. 5, pp. 1617–1628, 2009.

- [24] WooYeon Jeong and Kyoung Mu Lee, “Cv-slam: a new ceiling vision-based slam technique,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3195–3200, 2005.
- [25] Wikipedia contributors, “Linux — Wikipedia, the free encyclopedia.” <https://en.wikipedia.org/w/index.php?title=Linux&oldid=947638888>, 2020.
- [26] S. Cousins, “Welcome to ros topics [ros topics],” *IEEE Robotics Automation Magazine*, vol. 17, no. 1, pp. 13–14, 2010.
- [27] Tim Field, Jeremy Leibs, James Bowman, “Ros bags.” <http://wiki.ros.org/rosbag>, 2020.
- [28] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.
- [29] Open CV, “Open cv official website.” <https://opencv.org/>.
- [30] Y. M. Wang, Y. Li, and J. B. Zheng, “A camera calibration technique based on opencv,” in *The 3rd International Conference on Information Sciences and Interaction Sciences*, pp. 403–406, 2010.
- [31] Open CV contributors, “Camera calibration.” https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html.
- [32] J. Labuz, “Triangulation of surface points with cameras and projectors,” in *[1988] Proceedings. The Twentieth Southeastern Symposium on System Theory*, pp. 342–348, 1988.
- [33] F. K. Noble, “Comparison of opencv’s feature detectors and feature matchers,” in *2016 23rd International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pp. 1–6, 2016.
- [34] A. Richardson, J. Strom, and E. Olson, “AprilCal: Assisted and repeatable camera calibration,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013.
- [35] E. N. Mortensen, Hongli Deng, and L. Shapiro, “A sift descriptor with global context,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 1, pp. 184–190 vol. 1, 2005.
- [36] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011.
- [37] D. Li, Q. Xu, W. Yu, and B. Wang, “Srp-akaze: an improved accelerated kaze algorithm based on sparse random projection,” *IET Computer Vision*, vol. 14, no. 4, pp. 131–137, 2020.

- [38] P. Fan, A. Men, M. Chen, and B. Yang, “Color-surf: A surf descriptor with local kernel color histograms,” in *2009 IEEE International Conference on Network Infrastructure and Digital Content*, pp. 726–730, 2009.
- [39] S. A. K. Tareen and Z. Saleem, “A comparative analysis of sift, surf, kaze, akaze, orb, and brisk,” in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pp. 1–10, 2018.
- [40] Jií Koktan, “Localization of a mobile robot by ceiling observation lokalizace mobilního robotu sledováním stropu.” <https://dspace.cvut.cz/handle/10467/82659>.
- [41] Mathwork help center contributors, “Object detection in a cluttered scene using point feature matching.” <https://www.mathworks.com/help/vision/examples/object-detection-in-a-cluttered-scene-using-point-feature-matching.html>.
- [42] A. Jakubovi and J. Velagi, “Image feature matching and object detection using brute-force matchers,” in *2018 International Symposium ELMAR*, pp. 83–86, 2018.
- [43] “Turtlebot.” <https://www.turtlebot.com>.
- [44] “Vicon Website.” <https://www.vicon.com/>.
- [45] P. Merriaux, Y. Dupuis, R. Boutteau, P. Vasseur, and X. Savatier, “A study of vicon system positioning performance,” *Sensors*, vol. 17, p. 1591, Jul 2017.