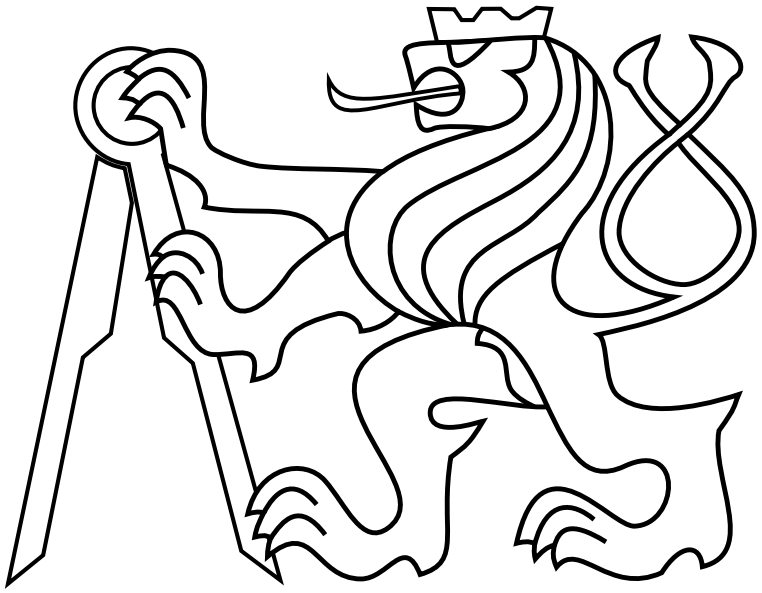
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Zdeněk Rozsypálek

Brick Detection for MBZIRC Competition

Department of Cybernetics

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

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne .....

.....
Podpis autora práce

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Rozsypálek Zdeněk** Personal ID number: **457216**

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:

Brick detection for MBZIRC competition

Master's thesis title in Czech:

Detekce cihel pro soutěž MBZIRC

Guidelines:

1) Study methods for analysis of Velodyne spatial depth data and study MBZIRC competition rules.
2) Analyze data from a Velodyne sensor placed on a ground robot and design an algorithm for detection a wall made up of blocks of predetermined sizes. The output of the algorithm should be a list of 3D cuboid positions relative to the robot position.
3) Design an algorithm that would create a map of the wall and allow to combine measurements from multiple robot positions.
4) Test the algorithm on real sensor data.

Bibliography / sources:

[1] Himmelsbach, Michael, et al. "LIDAR-based 3D object perception." Proceedings of 1st international workshop on cognition for technical systems. Vol. 1. 2008.
[2] Dou M., Guan L., Frahm JM., Fuchs H. (2013) Exploring High-Level Plane Primitives for Indoor 3D Reconstruction with a Hand-held RGB-D Camera. In: Park JI., Kim J. (eds) Computer Vision - ACCV 2012 Workshops. ACCV 2012. Lecture Notes in Computer Science, vol 7729. Springer, Berlin, Heidelberg
[3] Ma, L., Kerl, C., Stückler, J., & Cremers, D. (2016, May). CPA-SLAM: Consistent plane-model alignment for direct RGB-D SLAM. In 2016 IEEE International Conference on Robotics and Automation (ICRA) (pp. 1285-1291). IEEE.

Name and workplace of master's thesis supervisor:

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

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

Date of master's thesis assignment: **14.01.2020** Deadline for master's thesis submission: **22.05.2020**

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

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

Acknowledgements

I would like to express my appreciation to RNDr. Petr Štěpán, Ph.D. for his valuable and constructive suggestions during the planning and development of this thesis. I would also like to thank the whole ground robot team that participated in the MBZIRC 2020 contest for their cooperation. Finally, I wish to thank my family for support throughout my study.

Abstract

The MBZIRC contests are focused on using autonomous multi-robot systems for tasks that are motivated by real-world problems. One of the tasks was to exploit the group of drones and ground robots to build a wall from bricks. Because all robots have to operate autonomously, it is crucial to have a system that can reliably detect the bricks. The set of methods that were used to detect these bricks is described in the thesis. Most of the methods are based on the lidar data measured by the ground robot. The lidar scans are processed by the split and merge algorithm to find the bricks in a very sparse pointcloud. Further, the RANSAC algorithm is applied from close range to verify the spatial distribution of different types of bricks. Moreover, the maximum likelihood estimate of the brick pile model, for usage in the EM algorithm, is derived. With a proper model and symbolic map, it is possible to poll previously saved partial measurements with different levels of confidence and obtain a position of any large spatially distributed object.

Keywords: MBZIRC, object detection, lidar, pointcloud, hypothesis fitting, EM algorithm, RANSAC

Abstrakt

Soutěže MBZIRC se zaměřují na použití autonomních multi-robotických systémů pro úkoly, které jsou motivovány problémy reaálného světa. Jedním z úkolů bylo využít skupinu dronů a pozemní roboty pro postavení zdi z cihel. Jelikož všichni roboti museli pracovat plně autonomně, bylo naprosto nezbytné navrhnout systém, který může cihly spolihlivě detekovat. Diplomová práce popisuje metody, které byly použity pro detekci těchto cihel. Většina metod je založena na zpracování dat z lidaru, který nese pozemní robot. Paprsky z lidaru jsou zpracovány algoritmem split and merge, což umožňuje nalézt pozice cihel ve velmi řídkém mraku bodů. Dále je použit algoritmus RANSAC, který z malé vzdálennosti může ověřit vzájemné pozice různých druhů cihel. Kromě toho byl udělán odhad maximální věrohodnosti pro model hromady cihel, který je poté použit v EM algoritmu. Pokud je použit správný model, je takto možné nalézt jakýkoliv velký prostorově rozložený objekt, pouze na základě částečných měření, s rozdílnou úrovní jistoty, uložených do symbolické mapy.

Contents

List of Figures

Chapter 1

Introduction

Autonomous robotics experienced rapid development in the last decades. There are broad ranges of applications from heavy work in the industry to autonomous cars or space exploration. The two latter mentioned applications are very challenging, mainly because they require many abilities that are specific to the area of mobile robotics. Unlike in industry where robots usually perform repetitive tasks in an unchanging environment, the mobile robot must be aware of its surroundings. The robot is usually equipped with various sensors that provide information about the environment to enable the robot to perform complex tasks. Machine perception is a subfield of autonomous robotics which is dedicated to interpreting these sensors' output data.

The thesis is focused mainly on processing the data from a lidar (Light Detection And Ranging) sensor. The lidar is a laser-based rangefinder that uses the passive reflection of a detected object. It has a wide spectrum of usage, and it is also very popular in autonomous robotics. These applications are frequently using spinning version of the lidar, which can cover 360° around the robot. In recent years companies like Ouster or Velodyne significantly reduced the cost and improved the quality of this technology, which led to further increasing interest. The classical applications includes collision avoidance [1], SLAM [2] or detection [3].

The goal of this thesis is to present a detection algorithm for the MBZIRC 2020 (Mohammed Bin Zayed International robotic challenge). The contest aims at the development of autonomous robotics and presents challenging problems that must be resolved before the robots can be applied in the real world. All of the participants are further motivated by the prize money for the winner.

Chapter 1 presents a problem, used equipment and software. The state-of-the-art methods are also discussed. In chapter 2 are proposed general methods that can be used for solving the problem. In chapter 3 are introduced tweaks to previously presented methods, which are necessary for our use-case. Chapter 4 contains a description of the conducted

experiment and performance analysis of all methods. Last chapter 5 is devoted to the evaluation and conclusion.

1.1 State of the art

Because lidar is the most precise depth measuring sensor, it is often used for object detection in 3D space. Many proposed detectors use various machine learning methods. The SVM method was sucessfully used for classification before the boom of neural networks [3]. Nowadays, the most popular solutions are the ones powered by deep neural networks. CNNs (Convolutional Neural Networks), which operate on a 3D voxel map, are often used [4]. This Voxel network architecture provides an end-to-end solution and can detect and classify on voxel map in one run. Many other architectures, which can improve the performance of such a network, were proposed. For example, slightly different CNN architectures or loss functions [5].

These networks have excellent performance, but they are developed mainly for use in autonomous vehicles. That means that they are trained on large annotated datasets, and they are usually computationally intensive (they require GPU). Moreover, the networks are used on quite dense voxel map, which is generated by lidar with very high vertical resolution. Well known KITTI dataset was captured by lidar with more than four times higher vertical resolution than the lidar used in this thesis [6].

Very unpleasant is that the brick is very often hit by only one lidar layer. That is caused by the small size of the brick and low resolution of the lidar. We concluded that using some advanced feature extractor is unnecessarily complex to detect objects which manifest in lidar data as a simple line. Furthermore, these classifiers usually require large annotated datasets that are not available for our problem. Instead, we use more legacy approaches involving mainly line segmentation algorithms.

1.2 MBZIRC Contest

The contest took place in March in Abu Dhabi. The whole competition consisted of three challenges and the grand challenge, which connected all challenges. The first challenge was the only one that was focused solely on UAVs (unmanned aerial vehicles). The goal of the first challenge was to pop multiple big-colored balloons and catch a small ball carried by the organizer's drone. The other two challenges were designed for both UGVs (unmanned ground vehicles) and UAVs.

The second challenge was about building a wall using robots. Multiple polystyrene bricks were placed in the arena, and the robots should have moved these bricks to the

destination area and stack them on top of each other to build the wall.

The third challenge was to extinguish a fire on the surface of the building model. This task was motivated by the inability of firefighters to extinguish fire inside high-rise buildings in United Arab Emirates. UAVs and UGVs carried tanks full of water and squirted it into the fire dummy. Every team had three rehearsals before the contest, and then two competition attempts for each one of the three challenges. Only the best teams from all three challenges were nominated into the grand challenge, which was limited to just one attempt.

1.2.1 Second Challenge

This thesis is focused on the second challenge, specifically on the ground robot section of the second challenge, so that we provide a more detailed description. Each team is given thirty minutes to explore the arena (40m×60m), localize all interest areas, and build the wall. There are four types of bricks with different colors. All bricks must be very light to enable the UAVs to pick them up. The dimension and colors of the bricks can be seen in the Figure 1.1. The team obtains points for every placed brick. Bricks with different colors are rewarded by a different number of points. Placing bigger bricks means higher rewards. In addition, the UAV bricks are rewarded by twice as many points as the UGV bricks.

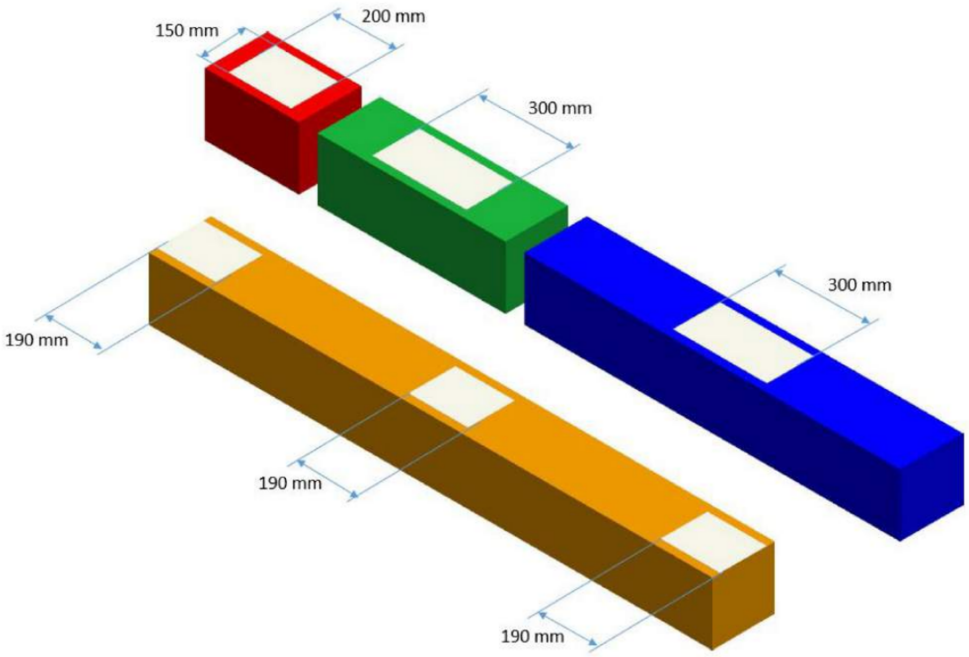


Figure 1.1: Colors and dimensions of the bricks provided by the organizer.

Each brick has a thin metal plate on top of it so that the robots are able to pick them up using electromagnets. At the beginning of the challenge, all bricks are placed in a beforehand unknown position. The initial position of bricks is unknown, but there is a predefined pattern in which the bricks are put together. There are different patterns for the UGV piles of bricks and the UAV piles. The UGV bricks are stacked into the multiple height levels, whereas the UAV bricks are stacked into the width, and all are put on the ground. Due to the low weight of the bricks, it is necessary to put UAV bricks into the rails. Otherwise, the bricks could be easily blown away by the propellers of the drones. Since the UAV bricks are all on the ground level (in the purely horizontal pattern), detecting them with the UAV bottom camera is much easier than using the lidar. That is why we are further concerned only about UGV bricks. These bricks are stacked in the positions displayed in the Figure 1.2.

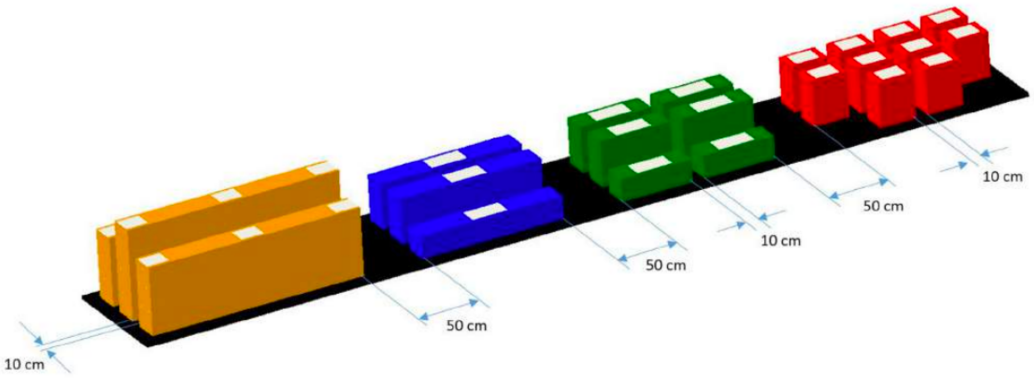
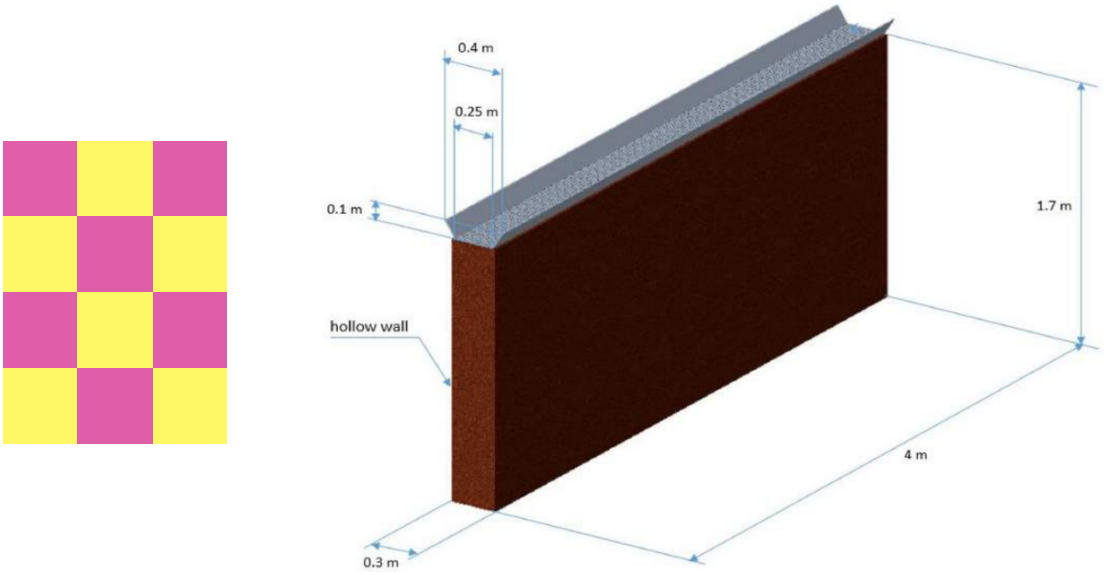


Figure 1.2: The positions of the bricks at the beginning of the second challenge.

Other objects of interest are destinations where the bricks should be placed. The robots must look for them during the exploration. UGV bricks' destination is marked by a checker pattern. Detecting the pattern was very challenging because the exact shape was not known until the second rehearsal. The final form of the pattern is shown in Figure 1.3a. Although we are not concerned about the UAV bricks, the destination of the UAV bricks is a vertical object, so it is much easier to detect it from the ground using the lidar. The UAV bricks destination is a wall, as can be seen in the Figure 1.3b. Bricks should be placed on top of this wall. The metal plate on top of each brick shifts the center of mass to the top and make the brick very susceptible to rolling. That is why are the auxiliary handles mounted on the top of the UAV destination wall. At the beginning of the challenge, each team is given the instructions which describe how the wall should look like at the end. When the built wall does not fit the instructions, the team gets a penalty and gains fewer points for inaccurately placed bricks.



(a) UGV destination pattern.

(b) UAV destination

Figure 1.3: Description of target places for UAVs and UGVs. Each square in the UGV pattern has width of 0.1m. Pattern consists of two 4×0.4m segments which are connected into the L shape. Whole UAV destination consists of four similar segments arranged into the W shape with right angles.

1.3 Equipment

For the sake of completeness, it is necessary to describe what exact equipment was available. We used **Clearpath Husky A200**, which is a wheeled robot designed for outside robotics. The robot was equipped with many additional devices. **Intel NUC** was used as a computer to run the code and control the robot. To manipulate the bricks the **Kinova robotic arm** was mounted on top of the Husky robot. Two **12V electromagnets** were attached to the end-effector to enable the arm to grip the bricks. It would be tough to grip the bricks without any feedback loop to the hand. For visual servoing and proper gripping, we placed **Intel Realsense** camera close to the end of the arm. It is also possible to obtain feedback from electromagnets thanks to hall effect sensors and decide whether the brick is gripped correctly. For the localization, collision avoidance and detection was used **Velodyne VLP-16** lidar sensor. Lastly, for moving the bricks around the arena, we created a handmade cargo area that can contain up to six bricks and attached it to the rear bumper. It was not possible to carry more bricks mainly because of restrictions on the robot's size and also due to the limited range of Kinova arm. The whole setup is captured in the Figure 1.4.

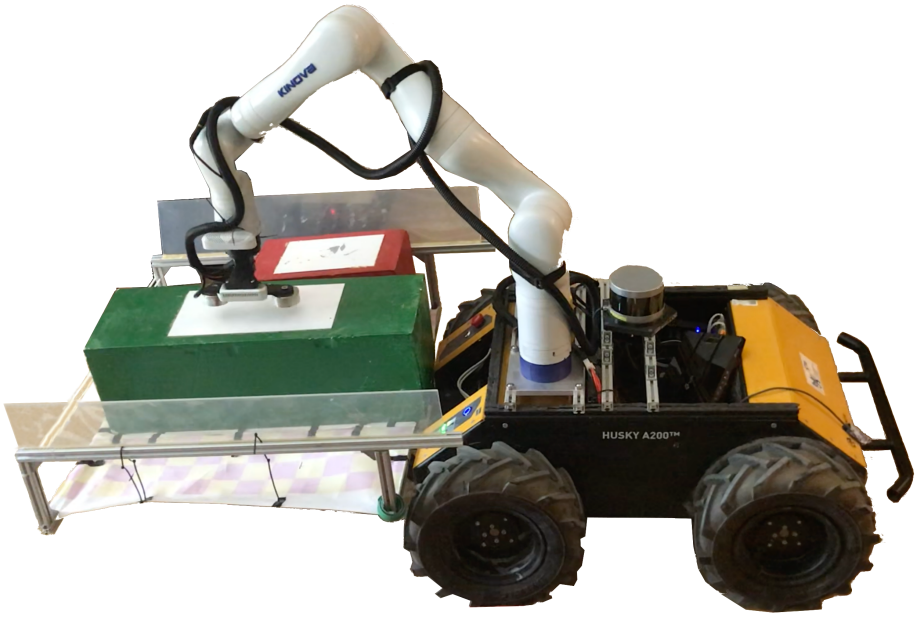


Figure 1.4: Clearpath Husky A200 adjusted for the second challenge.

1.3.1 Velodyne VLP-16

This thesis deals mainly with lidar data. Therefore, the following subsection provides a more detailed description of the lidar sensor. Inside the VLP-16 puck, there is a rotating infrared laser of class one, which measures the distance using the time of flight principle. A 12V power supply powers lidar and the data are transferred via UDP packets over the ethernet. Configuration of the Velodyne lidar used for the contest is described in the Table 1.1. The lidar can be set up with slightly better resolution, but there is a trade-off between resolution and frequency and we preferred higher frequency over the resolution..

Table 1.1: Parameters of VLP-16 lidar sensor.

Parameter	Value
Layers [-]	16
Maximal Range [m]	100
Vertical FOV [°]	±15
Vertical resolution [°]	2
Horizontal FOV [°]	360
Horizontal resolution [°]	0.4
Frequency [Hz]	20
Precision [m]	±0.03

1.4 Software

The operating system which is installed on the Intel NUC hard drive is Ubuntu 18.04. All necessary subroutines for controlling the robot are run by the Robot Operating System (ROS) [7]. That is a flexible framework for operating various robots or multi-robot systems. It offers many different tools and packages which can vastly simplify development, debugging, and deploying of any robotic platform. ROS is written in C++ but offers bindings to other modern languages such as Python or Ruby. Our work is implemented in C++ because it achieves the best performance in terms of execution time and it has the best community support. ROS precisely defines how the files must be structured within the package. ROS also offers a high encapsulation level where each executable is run as a node that can communicate with other nodes using the predefined protocols. The protocols are defined in the form of ROS messages, which are used in two different ways. One possibility is to transmit the data via the ROS topics. Topics represent the classical producer-consumer scheme, where multiple listeners can be joined to a single topic, whereas the publisher doesn't receive anything. The other possibility is a so-called service server that returns something on each request. The most important node is the ROS core, which runs the ROS master and the parameter server. ROS also provides advanced logging and package *rosbag*, which can record any specified topic and save it for replay.

It is vital to know the precise location of the robot inside the arena. For localization, the robot uses a probabilistic Monte Carlo method, which is sometimes called the particle filtering. This algorithm fuses data from range finder and odometry and creates multiple hypotheses (particles) of the robot's position and rotation. The quality of each hypothesis is evaluated by approximating posterior probability within a standard Bayesian formulation of the localization problem [8] [9]. Firstly, it is necessary to create a map when the robot is placed in a new environment. There is a ROS package called Gmapping, which can simultaneously create the map and navigate the robot using particle filter. When the map is created, it is not further necessary to run the Gmapping node because only the localization is needed. For the Monte Carlo localization on a predefined map, the ROS package called AMCL is used.

Chapter 2

Methods

In this chapter the methods that can be applied to the detection of the bricks are described. Multiple methods are presented. Each method is supposed to tackle different part of the detection process. The most of these methods are several decades old and they have proven to be very valuable in certain fields of computer science. Only a brief description is provided, and all methods are presented in a general form. First section of this chapter does not provide a description of any method, but rather shows our reasoning behind usage multiple methods with different range of detection.

2.1 Lidar detection range analysis

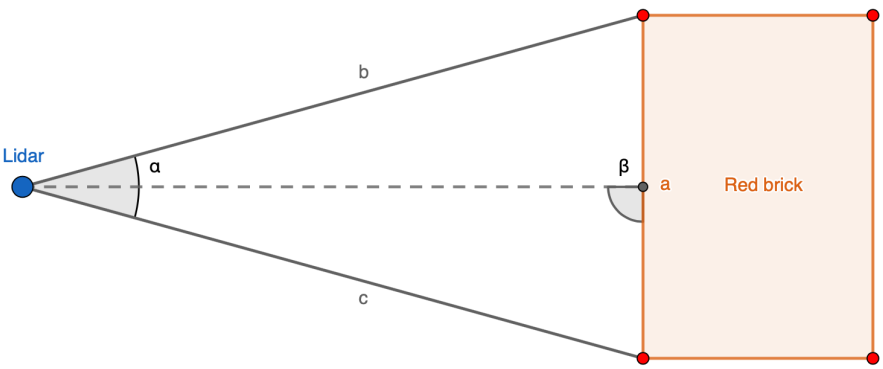


Figure 2.1: Visualization of rays hitting the red brick.

It is useful to know a possible range of detection based on the lidar sensor. This range influences how the robot explores the arena and influences the choice of used methods. Higher the detection range, the lower number of waypoints is necessary to explore the whole arena. There is a limited time for exploration because brick pickup and brick placement take much time. The speed of pickup and placement is limited mainly by the speed of the Kinova arm. We estimate the maximal range using the Figure 2.1.

Angle α is the resolution of lidar known from table 1.1. Only the maximal range is calculated, thus angle $\beta = 90°$. The cosine theorem can be used to obtain the distance between points on the brick:

$$a^2 = b^2 + c^2 - 2bc\cos\alpha. \tag{2.1}$$

Because β is a right angle we can write $b = c$ and thus:

$$a = \sqrt{2b^2\left(1 - \cos\alpha\right)}. \tag{2.2}$$

Now we want to know how many rays N would hit the brick from given distance b with lidar angular resolution α and the size of the brick a.

$$a = \sqrt{2b^2\left(1 - \cos\left(N\alpha\right)\right)}, \tag{2.3}$$

$$N = \frac{\arccos\left(1 - \frac{a^2}{2b^2}\right)}{\alpha}. \tag{2.4}$$

Finally, we can plot a function of the number of rays N with respect to distance to object b. This analysis can be done similarly for vertical and horizontal resolution.

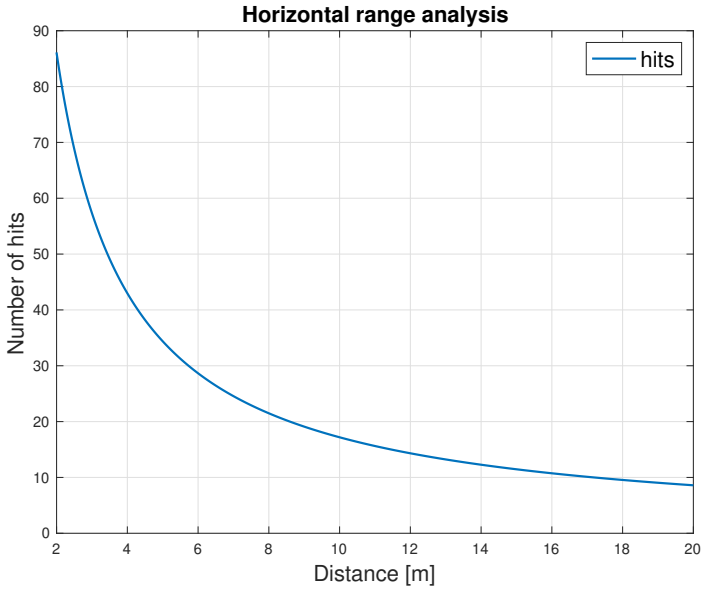


Figure 2.2: Number of rays which hits the smallest brick from given distance.

In Figure 2.2 it is visible that the horizontal resolution of the lidar is not limiting factor of the range. Even from 10m, the lidar is able to hit red brick more than 15 times. On the other hand, the Figure 2.3 shows that less than two lidar layers would hit the pile of bricks with a height of 0.4m from a distance greater than 6m. Furthermore, this is the best-case scenario analysis where β is a right angle, which rarely happens in reality, but we opted this value of beta because it simplifies the calculation and it is still reasonably precise estimate.

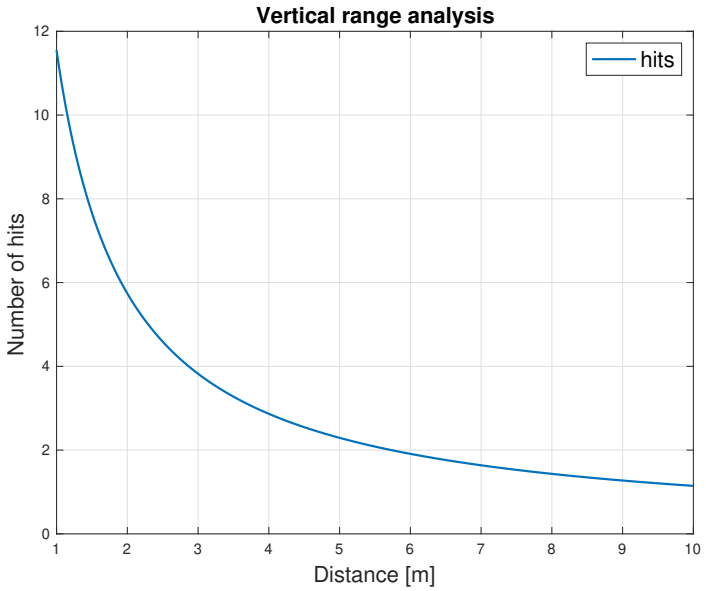


Figure 2.3: Number of layers which hits two stacked bricks from given distance.

2.2 Lidar data processing

The brick detection is based on extraction of straight lines from lidar data. Several methods can be used to achieve this goal. One of the most popular algorithms for line extraction is currently split and merge algorithm. Initially was this algorithm proposed for image segmentation by Horowitz and Pavlidis [10]. A simple version of this algorithm for point-cloud processing is described in algorithm 1, where C is clustering distance and S is splitting distance. There are many implementations of this algorithm which differ mainly in a way how they compute some particular steps of the algorithm. For example, just the method of fitting a line to the cluster can vary a lot. The method of least squares is often used, but as a simple method as connecting endpoints of the cluster could be used. When the latter method is applied, the algorithm is usually referred to as Iterative End Point Fit (IEPF) [11]. For the cluster creation, the points are iterated in each layer one by one. When the distance of the following points is too high, we split the cluster. Every cluster

is then further recursively split based on the most distant point from the fitted line. In comparison to other line extraction algorithms is the split and merge algorithm, one of the best performings in terms of precision and computational complexity [12].

Algorithm 1: Lidar data segmentation using split and merge algorithm.

```
   Data: pointcloud
   Result: line_segments
 1 initialize constants C, S;
 2 clusters = find_clusters(pointcloud, C);
 3 while clusters is not empty do
 4 |   cluster = clusters.pop();
 5 |   line = fit_line(cluster);
 6 |   point = most_distant_point(cluster, line);
 7 |   if distance(point, line) > S then
 8 |   |   c1, c2 = split_cluster(cluster, point);
 9 |   |   clusters.push_back(c1, c2);
10 |   else
11 |   |   line_segments.push_back(cluster[start], cluster[end]);
12 merge_parallel(line_segments);
```

2.3 EM algorithm

Expectation-maximization (EM) algorithm is an iterative process that can find parameters of a specified statistical model based on incomplete data. One of the most used statistical description for the EM algorithm is the Gaussian mixture model. This model is particularly useful because it emerges in many real-world situations, and it is easy to maximize. As the name of the algorithm suggests, it repeats the expectation and maximization step. Each iteration of the algorithm should improve the likelihood of the model until the terminating criterion is met. The termination criterion can simply be the number of iterations, or the algorithm can be stopped when the model is not improving anymore. Although we are discussing mainly the Gaussian distribution, the EM algorithm can also be used for other distributions from exponential family [13]. The usage of the EM algorithm for the classification of 3D lidar data is not an entirely novel approach. Gaussian mixtures were already used for terrain recognition [14].

2.3.1 Maximization

For maximization step, the maximal likelihood estimate (MLE) is used weighted by γ from expectation step. For parameters of Gaussian distribution $\mathcal{N}(\mu, \sigma)$ and number of samples N maximization looks as follows:

$$\mu = \sum_{n=1}^{N} \gamma_n x_n, \tag{2.5}$$

$$\sigma = \sum_{n=1}^{N} \gamma_n (x_n - \mu)^2. \tag{2.6}$$

A critical assumption for the convergence of the algorithm is that its likelihood with respect to the estimated parameter must be concave. This can be easily proved by computing the second derivative of the likelihood function. For example, for the mean value of distribution μ, it is easy to show that the second derivative of likelihood is always negative, which means that the function has no local optima:

$$\begin{aligned} \mathcal{L} &= \prod_{n=1}^{N} \mathcal{N}(x_n, \mu, \sigma), \\ \frac{\partial \log \mathcal{L}}{\partial \mu} &= \frac{1}{\sigma^2} \sum_{n=1}^{N} (x_n - \mu), \\ \frac{\partial^2 \log \mathcal{L}}{\partial \mu^2} &= \frac{-N}{\sigma^2}. \end{aligned} \tag{2.7}$$

2.3.2 Expectation

The expectation step is done simply by evaluating conditional probability of current parameters given the sample:

$$\gamma_n = P(\mu, \sigma | x_n), \tag{2.8}$$

which is calculated using the Bayes theorem and the law of total probability:

$$\begin{aligned} \gamma_n &= \frac{P(x_n|\mu, \sigma) P(\mu, \sigma)}{P(x_n)}, \\ \gamma_n &= \frac{P(x_n|\mu, \sigma) P(\mu, \sigma)}{\sum_{k=1}^{N} P(x_k|\mu, \sigma) P(\mu, \sigma)}. \end{aligned} \tag{2.9}$$

There is only one class so the priors can be evaluated $P(\mu, \sigma) = 1$ and the equation simplifies to final form:

$$\gamma_n = \frac{\mathcal{N}(x_n, \mu, \sigma)}{\sum_{k=1}^{N} \mathcal{N}(x_k, \mu, \sigma)}. \tag{2.10}$$

2.3.3 Algorithm

How to implement a generic version of the EM algorithm on sampled data is shown in algorithm 2, where **x** is the observed data samples. All relevant calculations for Gaussian distribution are described in previous subsections. It is not clear where to start iterating. It is possible to start both with the expectation and with the maximization step, but both parts are dependent on the result of the other one. Here we start with the maximization step, so during the initialization, we set $\alpha_n = 1$. If some prior information about the model's parameters is available, they can be set during the initialization, and the algorithm can be started with the expectation step. This informed initialization can remarkably reduce the number of iterations and sometimes even an outcome of the algorithm.

Algorithm 2: Pseudocode shows how to implement the EM algorithm.

```
Data: x
Result: parameters θ
set all α_n = 1;
while not stopping_criterion(x, θ) do
    θ = maximization(x, α);
    α = expectation(x, θ);
```

2.4 RANSAC

Random sample consensus (RANSAC) is an iterative method which can estimate parameters of the hypothesis given the data. It was first presented by Fischler [15] with application in scene and image analysis, but it can be used for fitting arbitrary hypothesis. The most significant advantage of this algorithm is its robustness to outliers. The major drawback of this method is its high time complexity when fitting hypotheses to noisy data with a large number of samples. The whole iterative process is described in the algorithm 3, where x is the observed data, η is the maximized cost and θ are the parameters of the hypothesis.

The number of drawn samples in **draw_samples** must be equal or higher than the number of degrees of freedom of the hypothesis. After drawing the samples method, **find_parameters** assigns the correspondences between sampled data and the hypothesis. The correspondences are used to obtain the parameters of the hypothesis. Then the algorithm is evaluating the quality of the hypothesis by applying the hypothesis to the whole dataset. This quality estimate can be done by an arbitrary cost function. A common practice is to define some metrics in our domain and use a threshold value to obtain the number of samples that fit the hypothesis. These samples are often referred to as inliers. Stopping

Algorithm 3: Pseudocode shows how to implement the RANSAC algorithm.

```
Data: x
Result: best parameters θ*
initialize θ, θ*, η, η*;
while not stopping_criterion(x, θ) do
    samples = draw_samples(x);
    θ = find_parameters(samples);
    η = compute_cost(x, θ);
    if η > η* then
        η* = η;
        θ* = θ;
```

criterion is usually met when the probability of sampling a better hypothesis is lower than a specified threshold. This section describes just the basic version of the algorithm. Many improvements to the RANSAC algorithm was proposed since 1981 [16].

2.4.1 Tentative Correspondences

The tentative correspondences can help us to choose better samples from the data to generate a better hypothesis. It is necessary to define some function which measures the similarity between data and hypothesis. The data which has a higher similarity to the hypothesis is then chosen with higher probability. It is also possible to completely ban correspondences with low similarity. Given the typical application in scene analysis, the similarity is usually computed by comparing keypoint descriptors.

2.5 Global model and transformations

One of the goals of this thesis is to develop a global model that can efficiently store and update the positions of interest points. This global model is in the map coordinate frame. The localization of the robot is essential for the precise global model. Every detection must be transformed from the coordinate frame of the sensor to the coordinate frame of the map. Transformation can be easily done using matrix multiplication:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \vec{t}, \tag{2.11}$$

where R is 3×3 the rotation matrix and $\vec{t}$ is a 1×3 translation vector between coordinate frames. Similarly the transformation can be done in homogenous coordinates by merging

translation and rotation into one matrix T:

$$\begin{bmatrix} \vec{x'} \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \vec{t} \\ \vec{0} & 1 \end{bmatrix} \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}. \tag{2.12}$$

Different framework for computing the transformations is using quaternions. The quaternions are currently the standard for transformations in computer graphics and robotics. The most significant advantage of quaternions is that they are more efficient and do not suffer from gymbal lock and ambiguity of rotation. Arbitrary rotation and scaling can be expressed as quadruple of numbers in the quaternion framework. The rotation between coordinate frames $B \rightarrow A$ is computed with quaternions as:

$$q_A = q_T q_B q_T^*, \tag{2.13}$$

where q_A is quaternion in coordinate frame A, q_B is quaternion in coordinate frame B, q_T is quaternion representing the transformation between these coordinate frames, and q_T^* is its conjugate. There is available a library within the ROS which can handle all these transformations in different forms [17].

2.5.1 Symbolic map

When all detections are transformed into the map frame, we can add them to a symbolic map. The symbolic map is storing the positions of all interest points and makes up the global model of the arena. Every object added to the symbolic map has a float number, which indicates the confidence of detection. When there is a new detection within a specific range from an object already stored in the symbolic map, a new object is not added, but only the confidence is increased. The range in which we just update the confidence is denoted as *cluster_size*. This approach creates clusters of interest points of different types. All interest points can be polled from the symbolic map, and the robot can make decisions based on the confidence of such an interest point. The symbolic map also checks whether the inserted object is located inside the arena. Otherwise, the object is rejected. One of the features of such a map is that it does not have to rely on measurements just from one sensor and can contain entries based on different types of measurements [18].

2.5.2 Lidar to camera registration

As can be seen in Figure 1.1 (where the bricks are defined) besides the dimensions, another important feature of the bricks is their color. Although the color manifests itself a little bit in reflectivity of the surface, which can be detected by lidar, it is not possible to reliably distinguish the colors using only the lidar sensor. Until there is a gap between the individual bricks, it is possible to detect brick using just the spatial data. Ideally, the robot

should be stacking bricks next to each other without any significant gap. Without any information about the color, it is impossible to decide whether the robot is detecting one large brick or several small bricks put together. So for this part of detection it is necessary to color the point-cloud. Coloring can be done by using the image from Intel RealSense camera and projecting a 3D lidar point-cloud to the camera plane. For this purpose, the pinhole camera model is used. To describe such a model is used intrinsic camera matrix K which consists of intrinsic camera parameters [19]

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.14}$$

where f_x, f_y are focal lengths and c_x, c_y stands for optical center of the camera.

Firstly, it is necessary to transform the whole point-cloud from the lidar coordinate frame to the camera frame. For this purpose, the so-called extrinsic camera parameters are used, which describe where a camera is placed in the lidar coordinate frame. This type of transformation is discussed at the beginning of this section. Secondly, we can use the intrinsic camera parameters to calculate the projection. Note that it is needed to work in a homogenous 2D coordinate system:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{2.15}$$

It is possible to decide which coordinates u, v on image plane corresponds to 3D point x, y, z from point-cloud, and assign the color of a specific pixel to the point, using this matrix equality. However, this works only for the simplest pinhole camera model without any distortion of the image. If the lens has non-negligible distortion, this distortion must be included in the camera model. For the description of distortion the distortion coefficients are used.

Chapter 3

Application of methods

In the following part of the thesis is described how to apply discussed methods to our problem. It is necessary to make several adjustments and also combine some of these methods to ensure reliable detections.

3.1 Detection pipeline

Detection is divided into three parts. All three parts are discussed in the next three subsections. detection pipeline is visualized in the Figure 3.1.

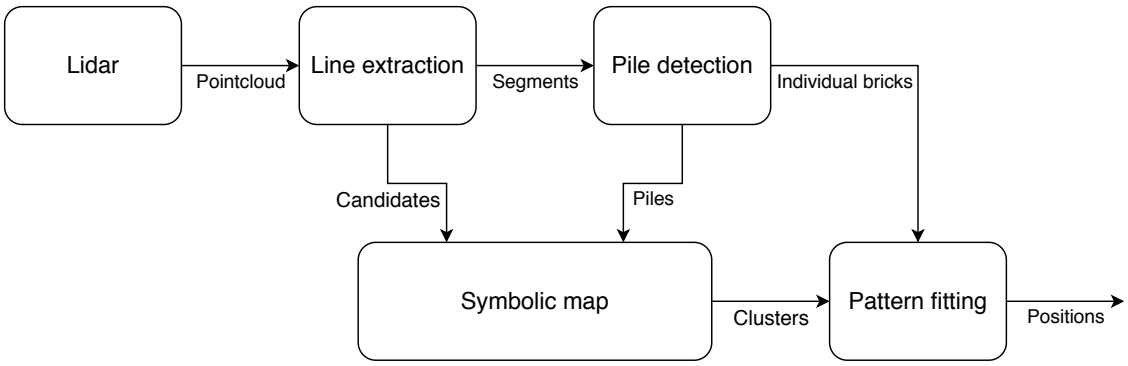


Figure 3.1: Visualization of detection pipeline.

It can be seen that the symbolic map has two types of inputs. The main advantage of this approach is that it extends the lidar detection range. As discussed in the previous section, the used lidar has a low vertical resolution of only 16 layers. Thus the bricks are

often visible in only one layer of lidar scan. If we use only one scan, there is a high probability of false-positive measurements. On the one hand, we can decrease the occurrence of false positives by adding other lidar layers into the detection process. But on the other hand, two layers are available only from a distance smaller than 6m and that decreases the detection range. Therefore we exploited both approaches. One layer line segmentation for generating the candidates with low confidence and multilayer pile detector providing high-quality estimates.

It is important to obtain positions in the map frame, but for grasping it is also necessary to introduce an algorithm which can give a real-time feedback to the robot frame to align itself with the brick pile. When the robot is too close to the pile it is impossible to get feedback from lidar because the bricks can be occluded or they can be closer than the lidar minimal range. Therefore the robot needs a camera servoing for fine alignment during the brick pickup.

3.2 Line segmentation

For line segmentation, the IEPF algorithm is used, which is very similar to the one described in algorithm 1. Only the final merging of parallel segments is omitted because it can connect two bricks into one. After retrieving the segments, filtering based on the segment size is done. It is possible to assign the color to the segment because each brick type has unique size. For this reason a new constant max_err must be defined which thresholds out detections of incorrect size. An example of lidar measurement with extracted and filtered lines is in Figure 3.2.

Algorithm performance is influenced by the correct setup of constants C and S (clustering and splitting distance). If we choose C too high, the algorithm could join two bricks into one segment. As described in Figure 1.2, the distance between two bricks of the same color is 0.1m. Therefore the clustering distance must always be less than 0.1m. If the clustering distance is too low, one brick can be unintentionally divided into many segments, and without merging at the end, these segments are useless. It is necessary to bear in mind that the lidar precision, as shown in Table 1.1 is ± 0.03m, so any clustering with a distance of similar magnitude would be highly affected by sensor noise. The found segments are transformed into the map frame and passed to the symbolic map as low confidence detections. The segments are further passed to pile detector which can filter out false-positive detections.

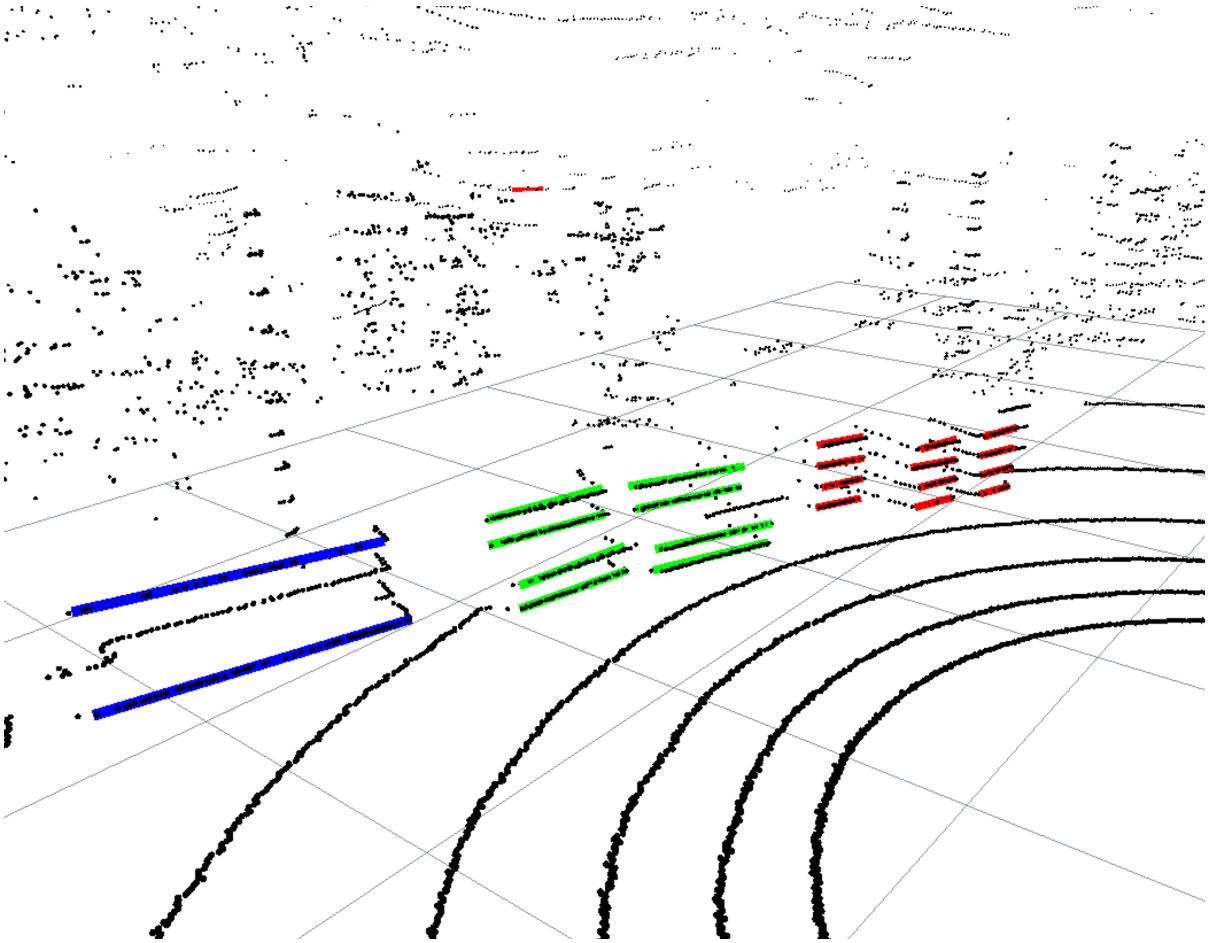


Figure 3.2: Visualization of line segmentation. In the figure it is evident that the bricks are well detected. This detection is done approximately from the distance of 3m. There is one false positive detection visible behind the brick piles on a tree.

3.3 Pile detection

The pile detector uses one of the simplest versions of the EM algorithm. As a model for the pile is applied 2D multivariate Gaussian distribution with variance fixed to one. Although this model is not an accurate description of the detection probability, it has other advantages already discussed in previous chapters. It is easy to work with, and it converges to the global optimum very well. Only the mean value is optimized, and it should converge into the center of the pile. The stopping criterion for the algorithm is solely the number of iterations. The robot is a realtime system, and there are strict demands for meeting a deadline.

There are further requirements for a hypothesis to be declared as a pile after the

optimization is done. We look around the proposed center in a one-meter radius, and we inspect all bricks found in this area. All the following conditions must be fulfilled for segments in a pile:

(a) There are at least two unique heights (z positions).

(b) There are at least two unique places ((x, y) positions).

(c) Difference between maximal and minimal height is smaller than the pile height.

(d) Center of the pile is inside the arena.

When these conditions are met, the hypothesis is declared as the pile and pushed into the symbolic map with high confidence. When at least one of these conditions is violated, then all the segments in this hypothesis are deleted, and the algorithm runs again until there are no more segments within the hypothesis radius. The condition (b) does not apply to the orange pile, because this pile is much higher than others. The whole procedure is shown in the algorithm 4.

Algorithm 4: Algorithm to obtain pile centers.

```
Data: segments
Result: pile_position
while True do
    pile_position = fit_em(segments);
    pile_segments = segments_in_pile(pile_position, segments);
    if pile_segments.size() < 2 then
        return None;
    if is_pile(pile_segments) then
        return pile_position;
    else
        segments.delete(pile_segments);
```

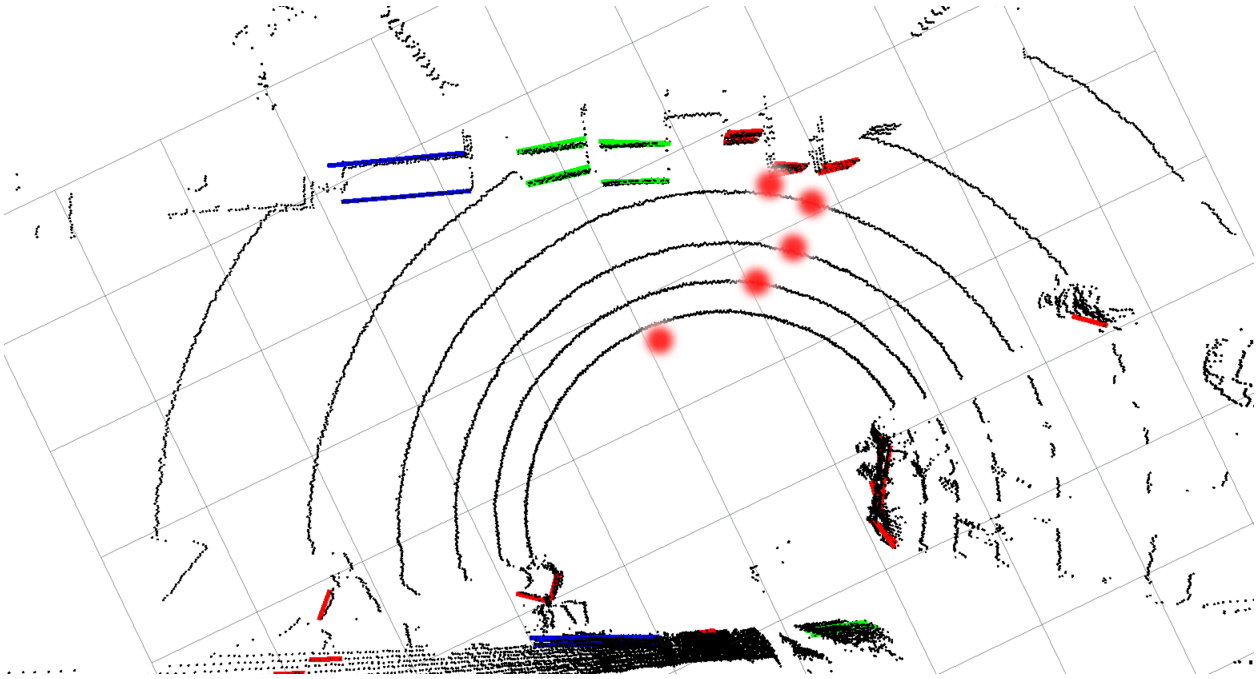


Figure 3.3: Several steps of EM algorithm for the red pile. Colored lines are the visualized segments the same as in Figure 3.2. Red dots show subsequent positions of the mean value of multivariate Gaussian. Although there are many false positives, especially at the bottom of the picture, the pile model ensures that the algorithm converges to the correct place.

3.4 Pattern fitting

As shown in Figure 3.1, the last step of the detection pipeline is pattern fitting. It is also visible that there are two types of input into this last step. Each input is used for a different type of pattern fitting. In the previous two sections, it was described how to generate candidates with different levels of confidence. This section describes how these candidates can be used for generating actual positions and how information about the spatial distribution of bricks can be exploited for the detection. The exact position of each pile and even each brick is visible in Figure 1.2. The pattern in which the bricks are stacked has three degrees of freedom - position (x, y) and rotation ϕ. The goal of the pattern fitting is to find the values of these parameters in the map frame.

3.4.1 Brick fitting

The first type of fitting uses detected 3D positions of individual bricks obtained in a pile detector. The pattern defines the position of each brick, which can be detected by the

lidar sensor. The next step is to generate a hypothesis that aligns this pattern with the measured positions of bricks. This step utilizes the RANSAC algorithm. Firstly we draw two different bricks from the detected set of bricks. Secondly, the correspondences are used to find the transformation - two correspondences are enough to generate the hypothesis (rotation matrix $\boldsymbol{R}$ and translation vector $\vec{t}$). The brick types (colors) and brick z positions are used as the correspondences. Also, we know that the distance between corresponding pairs should match. Otherwise, it would be incorrect correspondence. Lastly, we measure the cost of the hypothesis. When the bricks are stacked into the pattern with reasonable precision, it is possible to fit the pile with average brick error down to ±3cm. After such an observation, we set the inlier distance d_{inlier} to 5cm. The algorithm is stopped after a certain number of iterations. The result is passed only if all detected bricks are inliers.

Hypothesis ambiguity

We cannot properly test the hypothesis when there are not enough bricks found. In addition, we can see the pile from behind, which renders two different patterns to match. For this reason, at least five detected red bricks are required to start the brick fitting. As shown in Figure 3.4, even with four detected red bricks the hypothesis can be easily fitted incorrectly. All conditions which can start fitting the hypothesis from front view are listed below:

(a) 5 red bricks detected.

(b) 3 green bricks detected.

(c) 4 bricks of at least two colors detected.

For view from behind only the condition (c) is sufficient.

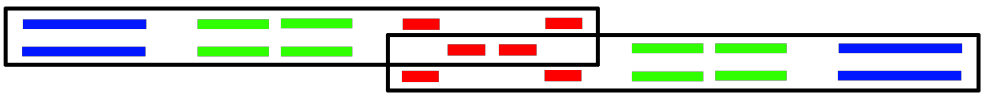


Figure 3.4: Two possible hypothesis which can be generated by four red bricks in their intersection are visualized in the black rectangles. This is top view of detection from the front, so it is not visible that all red bricks are in two layers.

3.4.2 Cluster fitting

The second type of fitting polls the clusters from the symbolic map and utilizes the confidence of clusters and their spatial distribution. For RANSAC algorithm, a hypothesis

using only four clusters would be too sparse to fit reliably. Because of this reason, the cluster fitting is done by the EM algorithm. As in pile detection, multivariate Gaussian distribution is used to represent the pile, but this time, the model takes into account relationships between positions of the piles. We define the probability model as follows:

$$P(\vec{x}_m) = \mathcal{N}_m(\vec{x}_m; \vec{\mu} + k_m\vec{v}; \boldsymbol{\Sigma_m}), \tag{3.1}$$

where m is pile (color) index and M is absolute number of colors, $\vec{x}_m$ is measurement of color m, $\vec{\mu}$ is mean value of whole model, $\boldsymbol{\Sigma_m}$ is covariance matrix, k_m is scalar multiplier unique for each pile and $\vec{v}$ is defined as:

$$\vec{v} = \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix}, \tag{3.2}$$

where ϕ is the rotation of the model. This definition of the model ensures that all piles (Gaussians) are in line with a distance defined by multiplier k. How could such a model look like is visualized in the Figure 3.5.

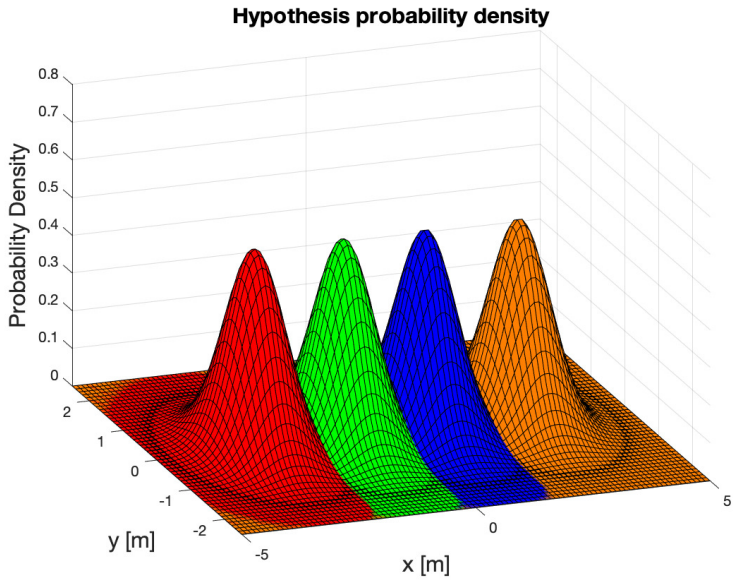


Figure 3.5: Example of probability density function which can be described as model 3.1. This particular model was created using $\vec{k}_{pile}$ from the Table 4.1, $\mu = [0, 0]$, $\phi = 0$ and $\boldsymbol{\Sigma} = 0.3\boldsymbol{E}$, where $\boldsymbol{E}$ is identity matrix of size 2.

Now we want to obtain the position and rotation of such a model based on real measurements. There is a closed-form solution for maximizing both terms, which can be found using the maximum likelihood estimate. For mean value, the derivation is very similar

to multivariate Gaussian:

$$\frac{\partial \log \mathcal{L}}{\partial \vec{\mu}_m} = \mathbf{\Sigma}_m^{-1} \sum_{n=1}^{N_m} (\vec{x}_{m_n} - \vec{\mu} - k_m \vec{v}), \tag{3.3}$$

$$\vec{\mu_m} = \sum_{n=1}^{N_m} \frac{\vec{x}_{m_n} - k_m \vec{v}}{N_m}. \tag{3.4}$$

Further, it is necessary to derive MLE for rotation ϕ which is hidden inside vector $\vec{v}$. For simplicity the matrix equation is now split into one part for each of two dimensions:

$$\frac{\partial \log \mathcal{L}_x}{\partial \phi_m} = \frac{1}{\sigma_x^2} \sum_{n=1}^{N_m} (x_{m_n,x} - \mu_x - k_{m,x} \cos\phi) \left(-k_{m,x} \sin\phi\right), \tag{3.5}$$

$$\frac{\partial \log \mathcal{L}_y}{\partial \phi_m} = \frac{1}{\sigma_y^2} \sum_{n=1}^{N_m} (x_{m_n,y} - \mu_y - k_{m,y} \sin\phi) k_{m,y} \cos\phi. \tag{3.6}$$

Likelihood derivative is now set equal to zero to find the extremes for each dimension:

$$\cos\phi_m = \frac{1}{k_{m,x} N_m} \sum_{n=1}^{N_m} \left(x_{m_n,x} - \mu_x\right), \tag{3.7}$$

$$\sin\phi_m = \frac{1}{k_{m,y} N_m} \sum_{n=1}^{N_m} \left(x_{m_n,y} - \mu_y\right). \tag{3.8}$$

Now it is possible to divide one equation by the other, use basic relationship of trigonometric functions, and derive final expression for maximizing the rotation ϕ:

$$\phi_m = \arctan \left(\frac{\frac{\sum_{n=1}^{N_m} (x_{m_n,y} - \mu_y)}{k_{m,y} N_m}}{\frac{\sum_{n=1}^{N_m} (x_{m_n,x} - \mu_x)}{k_{m,x} N_m}} \right). \tag{3.9}$$

Although the expression can be further simplified, in this form, it is easier to weight each data sample by the expectation α. Also, it is necessary to consider the contribution of each color to the pile model. This can be done in two ways. One possibility is to set an equal contribution to each data sample. In our case, this could lead to bias towards the most detected color (usually red). Therefore we set equal contributions to all colors instead of the samples. The final form used in the maximization step looks as follows:

$$\vec{\mu} = \sum_{m=1}^{M} \frac{1}{M} \sum_{n=1}^{N_m} \vec{\gamma}_{m_n} \left(\vec{x}_{m_n} - k_m \vec{v}\right), \tag{3.10}$$

$$\phi = \arctan \left(\frac{\sum_{m=1}^{M} \frac{\sum_{n=1}^{N_m} \gamma_{m_n,y} (x_{m_n,y} - \mu_y)}{k_{m,y}}}{\sum_{m=1}^{M} \frac{\sum_{n=1}^{N_m} \gamma_{m_n,x} (x_{m_n,x} - \mu_x)}{k_{m,x}}} \right). \tag{3.11}$$

Method is demonstrated in the Figure 3.6. Expectation is calculated simply by evaluating the probability of sample $P(\vec{x}_m)$ as in equation 3.1. Results can be further improved when the confidence of sample is used as the prior probability.

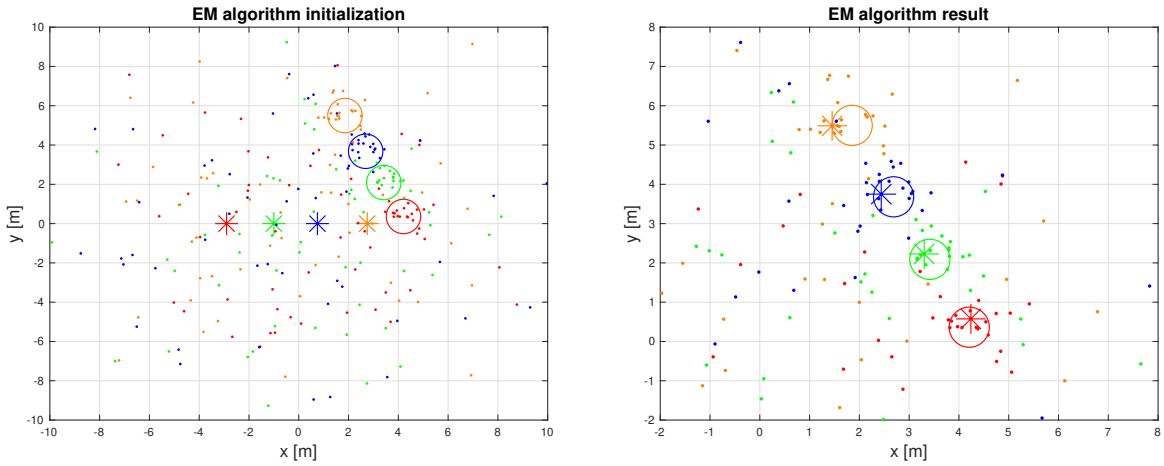


Figure 3.6: In the pictures, the pattern fitting using EM algorithm is visualized. On the left there is algorithm initialized with parameters $\vec{x} = (0, 0)$ and $\phi = 0$. Algorithm estimate is marked by stars. Points are generated by sampling multivariete Gaussian distribution. Model was sampled with parameters $\vec{x} = (3, 3)$ and $\phi = 2$ to create desired pattern of points. Positions of sampled piles are marked as circles. At the end of the procedure parameters are found correctly.

Convergence

The unimodality of the model's likelihood was disrupted by adding the rotation parameter to Gaussians. Possible local optimum is shown in the Figure 3.7. Now there are no guarantees that the algorithm converges into the global optimum. That is a common issue of the EM algorithm when dealing with more complex problems. Many solutions to this issue were proposed. Very advantageous is that local optima are usually significantly worse in terms of likelihood than the global optimum.

One way to avoid local optima is the deterministic annealing, which influences how the expectation is used in maximization step [20]. Another way how to escape local optimum is to apply a perturbations to parameters of the model. In our case, it could be, for example, rotating the model by 180°. A different approach would be a population-based em algorithm with multiple initializations or informed initialization. The latter is easily applicable in our case because the confidences of clusters could be used in the expectation step.

Because we have also the formulas for gradient, it is also possible to use an arbitrary gradient ascend method instead of an MLE to maximize the likelihood of the model. The

advantage of using the MLE for maximization step is that the algorithm converges very fast and the computation is usually much faster than the computation of any gradient method.

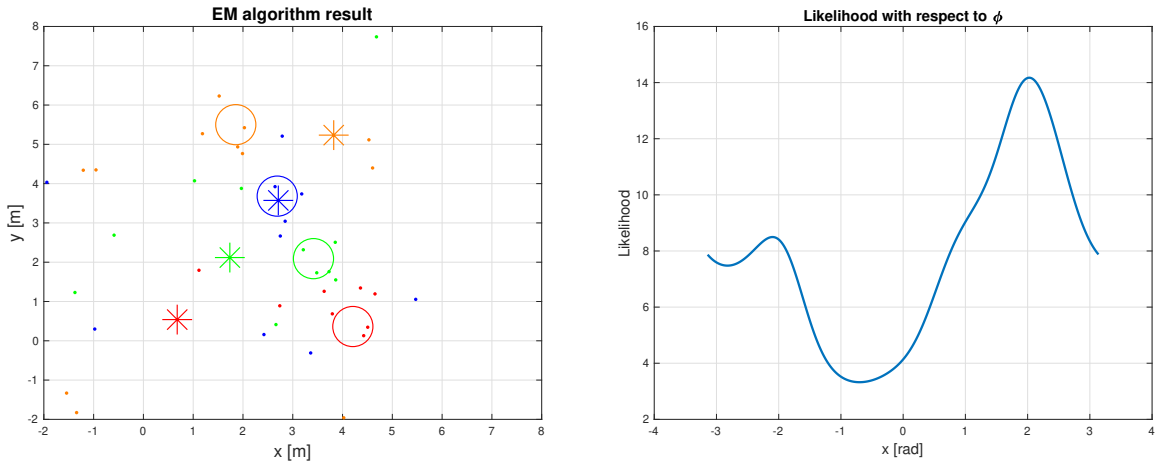


Figure 3.7: When there are not enough samples, the EM algorithm is more susceptible to getting stuck in local optima, as shown on the left. The right picture shows how different rotations of the model with fixed mean influences the likelihood, and it also shows that there is a local maximum. The likelihood is maximal when $\phi = 2$ which is optimal rotation of model.

3.5 Arena exploration

It is vital to emphasize on proper arena exploration. If all the interest points are not found, the robot can not proceed in completing the challenge. Except for the initial brick position, it is necessary to find also the destination where the bricks should be placed. This place is marked by the checker pattern in Figure 1.3a. The only sensor which can detect the destination pattern is the camera. The big advantage is that the camera is mounted onto the arm so that it can be raised into height and turn around. As can be seen in Figure 3.8, the most limiting factor of the detection range is currently the UGV destination pattern search.

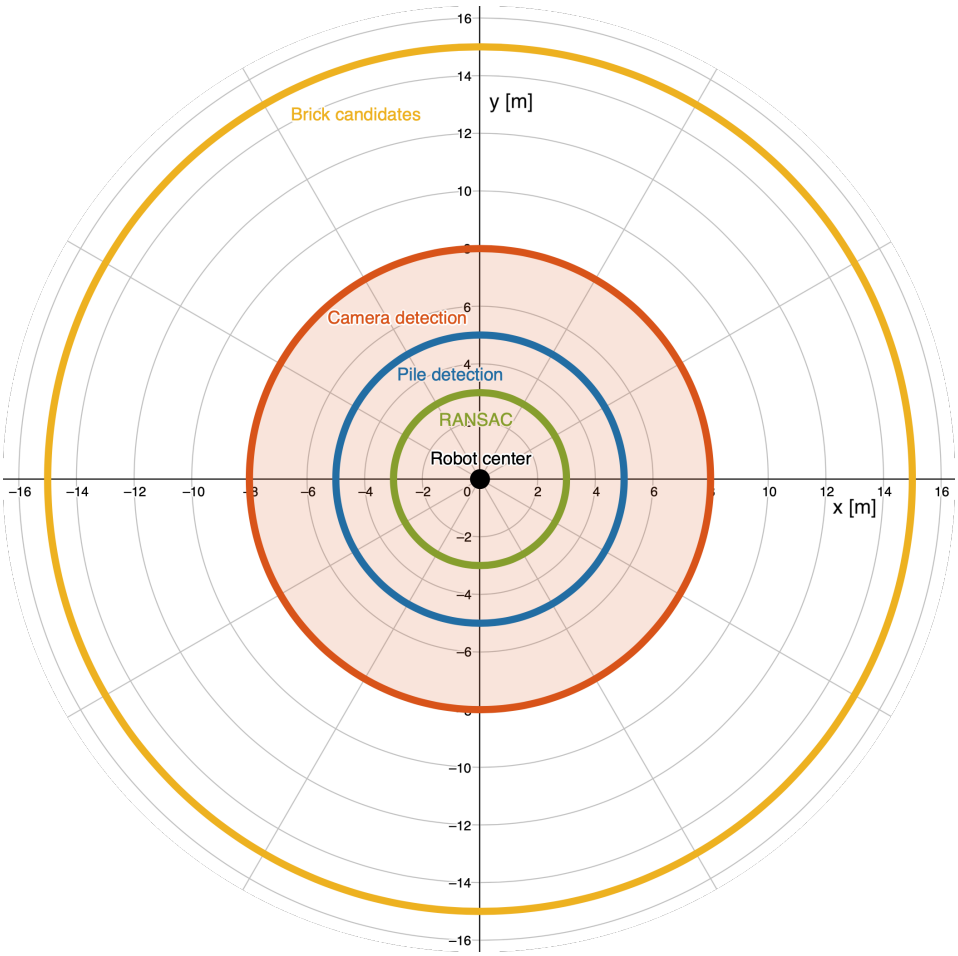


Figure 3.8: In the picture we see the range of each type of detection. Fitting the complete hypothesis with RANSAC method (green) requires high number of detected segments in piles, so the range is low - around 3m. For detection of the pile (blue) just few detected segments are required, so the detection range longer - around 5m. Camera can obtain candidates for checker pattern (red) from maximal distance of 8m, that is the limit distance for exploration. From even bigger distance candidates for bricks (yellow) using the lidar sensor can be generated. The upper boundary can be even higher but we set it manually to 15m to reduce the number of false positive detections.

The waypoints of exploration movement should be generated so that circles cover the area of the whole arena with an 8m radius. Generated waypoints in the arena are visualized in the Figure 3.9. On each waypoint, the robot must stop, raise the arm, and look around with the camera. There are several reasons why camera detection cannot be done while moving. First of all, the arm can not be raised to the highest possible position while the robot is moving. Otherwise, it could get stuck or damaged. In addition, the motion blur of camera would reduce the detection range even further. During camera detection, the robot must stay still, but the lidar detections can be done while moving. That can improve

the detection range and speed because the robot can do measurement from much more different places.

Whole map exploration is described in the algorithm 5. The waypoints are ordered prior to this algorithm. How to order them is beyond the scope of this thesis. In algorithm is visible that the method **start_lidar_detection** is non-blocking service which just turns on the detector, whereas the method **do_camera_detection** is blocking service which waits until the arm looks around with camera and finishes the detection. This method is also the most time consuming part of the loop.

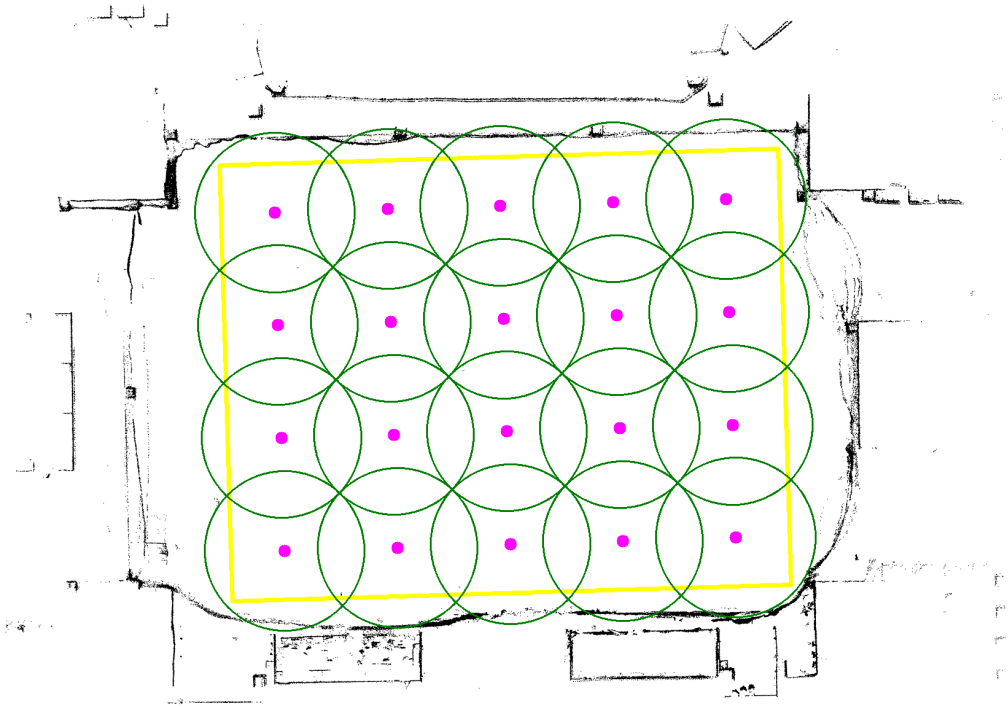


Figure 3.9: Purple waypoints are places where the robot should stop and look around. Green circles are camera ranges from the corresponding place. In this figure, the waypoints are generated so that there is no unexplored area. That is usually not necessary since the objects of interest have non-negligible size. It is thus possible to further reduce the number of waypoints.

Algorithm 5: Algorithm to explore whole map.

```
   Data: waypoints
 1 done = False;
 2 while not done do
 3 |   waypoint = waypoints.pop();
 4 |   start_lidar_detection();
 5 |   go_to(waypoint);
 6 |   stop_lidar_detection();
 7 |   raise_arm();
 8 |   do_camera_detection();
 9 |   fold_arm();
10 |   if has_all_objects() or waypoints.empty() then
11 |   |   done = True;
12 if not has_all_objects() then
13 |   go_to(get_strongest_candidate());
```

3.6 Stacked bricks detection

It is evident that when the bricks are stacked close to each other without any significant gap, it is impossible to distinguish which bricks the lidar detects. This issue was already addressed in previous chapters. It is desirable to know what color particular lidar point has. That is done by the camera to lidar registration. Since we know the relative position of the camera to lidar and the intrinsic camera matrix, it is not a problem to assign a color to each point. The colors are further utilized during the clustering. We can now split clusters not only using the spatial data, but the splitting can be based on the color difference of the subsequent points.

Modified clustering is visible in algorithm 6. Input points must be from a single layer of pointcloud and ordered by yaw. Constant C is clustering distance, T is color clustering distance and min_size is minimal cluster size - this has high influence on range of brick candidate generation. Note that the color distance is computed from the whole cluster mean to a new point. When the new point is added, the color of the cluster should be recalculated. This is important to reduce camera noise influence on the final form of clusters. When this running-mean technique is used, the clusters are split only in sharp color transitions.

How the point-cloud coloring and final detection looks like is shown in Figure 3.10. Since the robot stacked all of these bricks, their relative position should be known. Whole stacking is a predefined sequence of movements, and thus we can add each placed brick into the list of stacked bricks with its relative position to place where stacking started. These known positions can be used to generate hypothesis which can be fitted by the RANSAC algorithm in the same way as during the pattern fitting.

When we use colors for the segmentation, it is crucial to choose the right color space. During the experiments, it was much easier to distinguish between green and blue color in LAB color space than in HSV color space. The LAB is more computationally intensive, but it is preferred color space for segmentation [21]. The robot is not capable of carrying orange bricks, so it was not necessary to segment the orange color, which is problematic because it can be easily confused with red color. This part of detection is not discussed any further because it was more important to localize the initial position of UGV bricks. Furthermore placing more than one load of bricks was not achievable in the given timeframe.

Algorithm 6: Spatial and color clustering.

```
   Data: points
   Result: clusters
 1 initialize constants C, T, min_size ;
 2 cluster = [], clusters = [];
 3 foreach pt in points do
 4 |   if cluster.empty() then
 5 |   |   cluster.push_back(pt);
 6 |   else
 7 |   |   if distance(pt, cluster[end]) < C and color_distance(pt, cluster) < T then
 8 |   |   |   cluster.push_back(pt);
 9 |   |   else
10 |   |   |   if cluster.size() > min_size then
11 |   |   |   |   clusters.push_back(cluster);
12 |   |   |   cluster.clear();
```

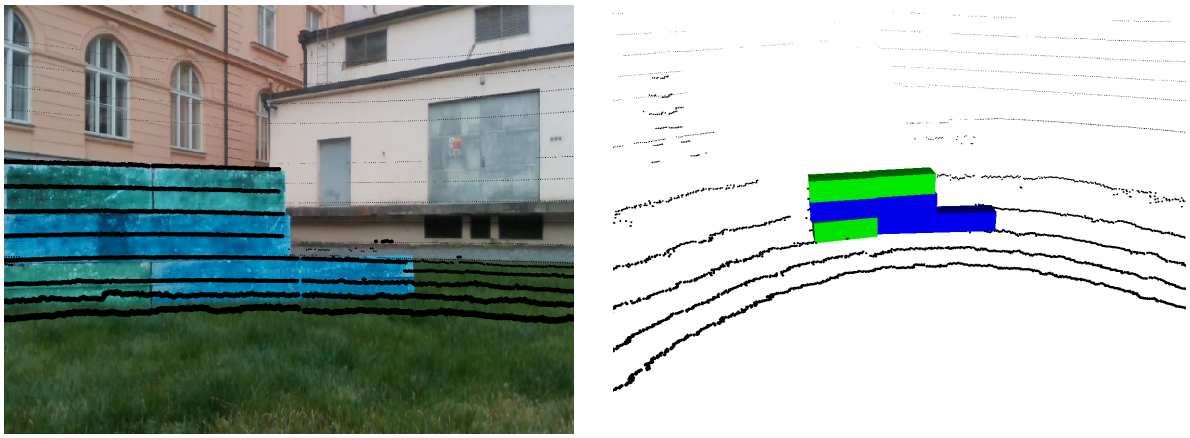


Figure 3.10: Detection of bricks using color based clustering. On the left: Image from camera pointing at built wall. Black dots are points captured by the lidar sensor. On the right: Final detection in map frame. All of the bricks are correctly detected, even though they are poorly painted and the environment is quite challenging, especially with the green grass on the ground.

Chapter 4

Experiment

The experiment was conducted on the *rosbag* taken during the first rehearsal attempt. The robot drove around the perimeter to map the whole arena using the Gmapping package. The brick pattern is placed in the left bottom quadrant and the drone delivery in the right top quadrant of the arena.

It was surprising that the organizers chose the arena with a significant slope in the middle. We did not expect perfectly flat ground, but the angle of the slope was much higher than our expectations. Besides, the middle part of the arena was made of different - very smooth surface, which negatively influenced odometry, especially in rotational movement. That made the localization in the middle much more difficult because, in this place, the robot was tilted, so the laserscan aimed into the ground or into the height, and there was also a quite high distance to the nearest significant feature inside the map, which could helped to adjust the position. This behavior is visible on the particle cloud during the experiment. When the robot arrives in the middle, the particles are spread into the width. These circumstances caused many problems to teams, which trained for the contest in laboratory grade conditions. Also, it is necessary to mention that there were many people inside the arena during our experiment, which makes the detection even more challenging because people could be easily interpreted as a red pile. Visualization of experiment is shown in the Figure 4.1.

The robot managed to localize all interest points even though the line segmentation produces a large number of false-positive detections. A significant advantage for the detection is that the clusters are merging measurements from different positions and because the robot is moving during the detections, it is improbable to accumulate multiple false positive detections into the one cluster. Thus these false-positive candidates have very low confidence, and it is not hard for the EM algorithm to find the actual position of interest points.

At the beginning of *rosbag* the robot drives around the piles in approximate distance

of 5m and then in approximate distance of 4m to UAV destination wall. At the end the robot drives back to the center of map and comes closer to the red pile. For experiment the detection pipeline as desribed in Figure 3.1 was used. Value of all constants is listed in Table 4.1. We conducted additional experiments when we tested only individual parts of the detection pipeline. In following sections the precision of such detections are measured.

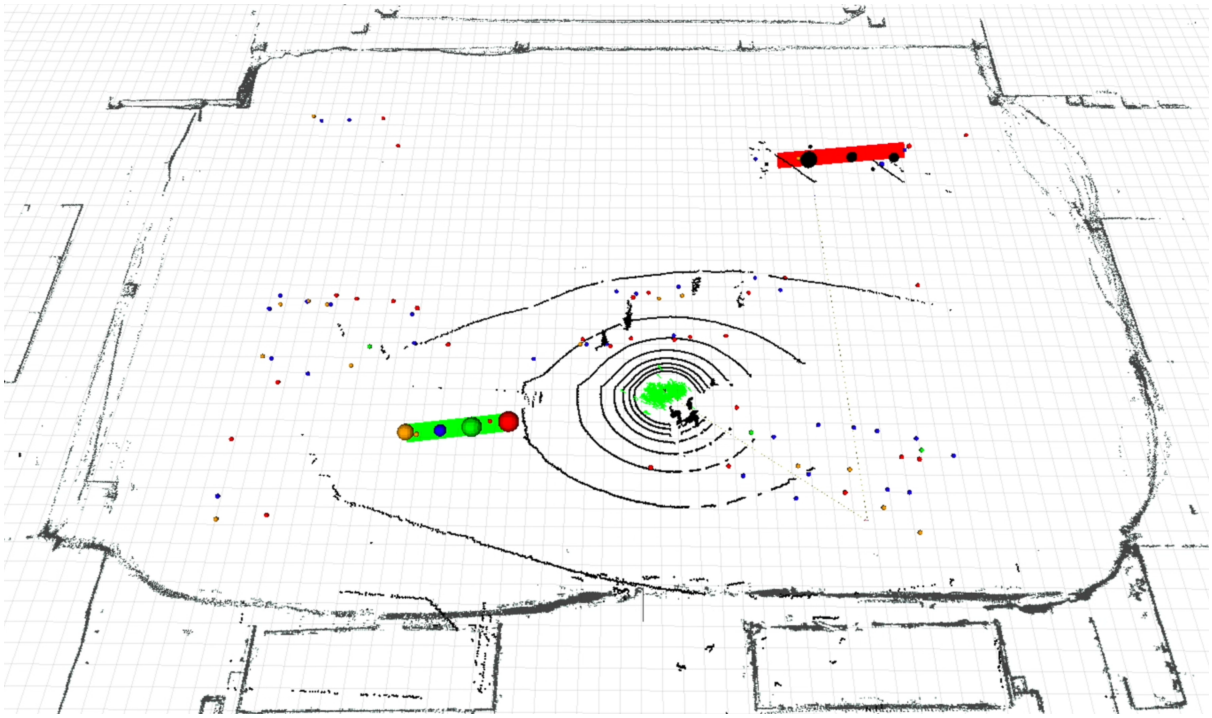


Figure 4.1: In the figure there is a screenshot from the experiment *. Small spheres on the map are polled clusters from the symbolic map. Different types of detections create these clusters. The size of the sphere represents the confidence of a cluster. The spheres' color corresponds to the color of bricks, and the spheres representing the clusters for UAV destination are colored black. Black is also the pointcloud obtained from lidar and borders of the map. All particles from the AMCL and also the line representing detection of initial brick pattern are visualized in green color. A red line marks the position and rotation of the UAV destination. Note that the RANSAC detector was disabled in this experiment to test the EM algorithm fitting properly.

*https://www.youtube.com/watch?v=KKpCGdJnv7Y

Table 4.1: List of constants used in experiment.

Constant	Value
C [m]	0.07
S [m]	0.07
σ_x [m]	0.3
σ_y [m]	0.3
d_{inlier} [m]	0.05
min_size [-]	10
max_err [m]	0.04
$cluster\_size$ [m]	0.5
$\vec{k}_{pile}$ [m]	$\begin{bmatrix}-2.9 & -1 & 0.75 & 2.75\end{bmatrix}$
$\vec{k}_{destination}$ [m]	$\begin{bmatrix}-\frac{6}{\sqrt{2}} & -\frac{2}{\sqrt{2}} & \frac{2}{\sqrt{2}} & \frac{6}{\sqrt{2}}\end{bmatrix}$

4.1 Precision of detections

Although all the objects of interest were correctly localized, it is visible in the Table 4.2 that the line segmentation produces high number of false positive candidates. That is caused mainly by incorrect filtering of ground plane. This behavior is further discussed in the next section. Very advantageous is that the EM algorithm converges well, until the correct clusters has at least slightly higher confidence than the false positives. It is no surprise that the most frequently detected bricks are the red ones, because it is the smallest of bricks and there is higher number of red bricks than bricks of any other color. Similarly the blue bricks are the rarest in the arena and blue pile is also the hardest one to detect as shown in the Table 4.3. The UAV destination wall has so low number of false positive detections mainly because the filtered segment has size of 4m, which is much larger segment than any brick. It is unlikely to obtain a large false positive segment in relatively empty arena.

The pile detector is very reliable which confirms that the conditions described in section 3.3 were chosen correctly. Both false positive detections were probably caused by pedestrians inside the arena. Finally the RANSAC detector produced zero false positives detections, that is mainly thanks to our very strict conditions to even start RANSAC fitting and also thanks to very low inlier distance. This level of precision is achieved mainly because we prefer precision over the detection range, which is the lowest of all detectors. In the experiment the robot drove much closer to the pile from behind, so in the Table 4.4 it is visible that the rear model was applied more times. It is noteworthy that the EM algorithm was able to fit the hypothesis correctly even when only the individual detection methods were used. Note that in the tables are considering only detections inside the arena.

All the detections outside the arena are filtered out by the symbolic map.

Table 4.2: Precision of line segmentation.

Type	True	False	Precision
Red	174	70	71.3%
Green	57	19	75.0%
Blue	24	63	27.5%
Orange	32	20	61.5%
Wall	280	11	96.2%

Table 4.3: Precision of pile detection.

Type	True	False	Precision
Red	39	1	97.5%
Green	36	0	100%
Blue	14	1	93.3%
Orange	17	0	100%

Table 4.4: Precision of RANSAC detector.

Type	True	False	Precision
Front model	1	0	100%
Rear model	9	0	100%

4.2 Ground Segmentation

As we already mentioned, the ground segmentation applied in this experiment was not sufficient enough. The slope inside the area was responsible for a high number of false-positive detections. The easiest possible ground segmentation method was used in the detector because we did not expect such harsh terrain. The height of lidar above the ground was 0.45m, so all points with z coordinate lower than -0.4m were filtered out from the measurement. This method can segment only wholly flat ground, which was not the case of the arena.

Ground segmentation is a popular subject of study. There are various published papers concerning this topic. Some methods are incorporating machine learning [22]. The

usage of machine learning again involves evaluating neural networks, and that can be inefficient on CPU. There are also purely algorithmic methods that perform well even in rough terrain [23]. These methods do not incorporate any knowledge about the environment, which is not entirely our case, because the arena terrain is known in advance.

We had advanced mapping device (Leica BLK 360) available, which is able to create a model of the arena using very dense pointcloud. Such a pointcloud can then be used to create an elevation map of the arena. This elevation map can help to segment out the ground in the robot measurements. Firstly, it would be necessary to implement queries to such a map. Then the robot could obtain the transformation of pointcloud using its position. When the candidates are generated, it could be verified whether the candidate is above the terrain. This method would work reasonably well until the localization of the robot works fine.

4.3 Time benchmark

The execution time of each detection algorithm is highly dependent on input size. The line segmentation time complexity grows quadratically with the number of samples [24]. It is the most time-consuming part of the whole pipeline because, unlike the pattern fitting, it must be done each iteration. The time complexity of the pile detector (EM algorithm) and the RANSAC algorithm is linear, but in our case both methods have a low number of inputs, so the final execution time is very short.

On the other hand, pattern fitting is also using the EM algorithm, but it polls the whole history of detections from the symbolic map. Hence it has a much higher number of inputs, and it takes a long time. In our implementation, the pattern fitting ran on every received pointcloud, but that is not necessary because the whole method is deterministic, and the result differs only after a significant change in the symbolic map. Relative lengths of execution is visualized in the Figure 4.2. We have compared the measured execution time to lidar frequency, and the pipeline computation time is not a limiting factor for the detection.

None of the methods is parallelizable and whole pipeline runs on one core of CPU in serial manner. Setup which ran the pipeline for benchmark is shown in table 4.5. The whole detection took on average 4.3ms.

Table 4.5: Setup for benchmark.

System	Ubuntu 18.04
CPU	Intel i7-7700k
RAM	8GB

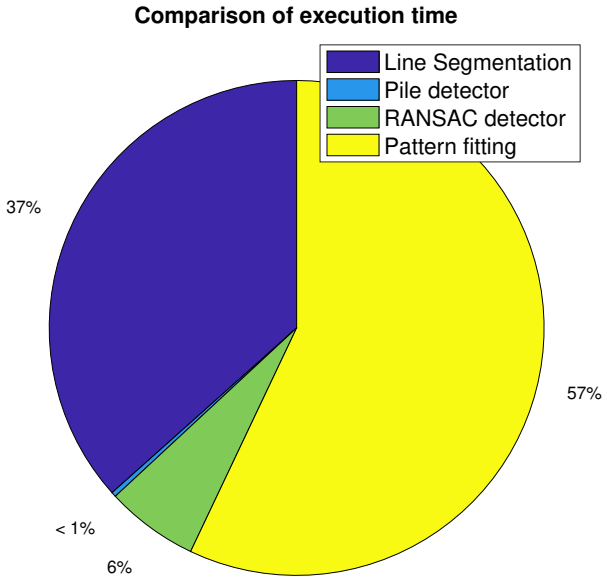


Figure 4.2: Relative comparison of mean execution time.

Chapter 5

Conclusion

The goal of the thesis was to implement a detection algorithm for MBZIRC 2020 contest. More specifically, the algorithm should detect the model of bricks for the second challenge of the contest. The Velodyne VLP-16 lidar sensor was used for the detection.

A simple line extraction algorithm was applied to find individual bricks. At the beginning of the challenge the bricks were stacked into the piles, so we applied the EM algorithm to find parameters of individual piles. Further, the piles were always placed with fixed relative positions. Since all the piles together are too large objects for detection in a single measurement, it was necessary to create a global model. In our case, the global model was the symbolic map.

When the robot's localization was used, and all detections were properly transformed and sent to the symbolic map, the initial position of all piles had to be found. We proposed a version of the EM algorithm which can exploit the spatial distribution of a model. Parameters of such a model were found based on partial detections and their confidences. The last type of detection used the RANSAC algorithm to fit individual brick positions to the hypothesis.

There could be no gap between the bricks when the robot successfully placed them. That makes it impossible to distinguish individual bricks by the lidar data. The color camera images were used to color the pointcloud. After coloring, it was possible to use custom clustering and detect individual bricks, that are very close to each other.

In the end, we experimented on the data from the contest, and we have shown that all algorithms work. Although the line extraction algorithm produces a large number of false positives, it is sufficient to detect initial positions of all piles and the UAV destination wall. Different ground segmentation methods were discussed to reduce the number of false-positive measurements. We have also shown that other types of detection have a lower range, but they produce a significantly lower number of false-positive measurements. To

sum up, all tasks of the thesis were successfully completed.

Bibliography

[1] Subramanian Ramasamy, Roberto Sabatini, Alessandro Gardi, and Jing Liu. Lidar obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology*, 55:344–358, 08 2016.

[2] Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, Karen Petersen, Uwe Klingauf, and Oskar Von Stryk. Hector open source modules for autonomous mapping and navigation with rescue robots. pages 624–631, 01 2014.

[3] M. Himmelsbach and H. Wunsche. Lidar-based 3d object perception. In *Proceedings of 1st International Workshop on Cognition for Technical Systems*, 2008.

[4] Oncel Tuzel Yin Zhou. Voxelnet: End-to-end learning for point cloud based 3d object detection. 2017.

[5] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors (Basel, Switzerland)*, 18, 2018.

[6] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[7] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[8] Dieter Fox Frank Dellaert. Monte carlo localization for mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA99)*, May 1999.

[9] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128:99–141, 05 2001.

[10] Steven L. Horowitz and Theodosios Pavlidis. Picture segmentation by a tree traversal algorithm. *J. ACM*, 23:368–388, 1976.

[11] Ali Siadat, Axel Kaske, Siegfried Klausmann, Michel Dufaut, and René Husson. An optimized segmentation method for a 2d laser-scanner applied to mobile robot navigation. *IFAC Proceedings Volumes*, 30(7):149 – 154, 1997. 3rd IFAC Symposium on Intelligent Components and Instruments For Control Applications 1997 (SICICA '97), Annecy, France, 9-11 June.

[12] V.T. Nguyen, A. Martinelli, Nicola Tomatis, and Roland Siegwart. A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics. pages 1929 – 1934, 09 2005.

[13] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[14] Jean-François Lalonde, Nicolas Vandapel, Daniel Huber, and Martial Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *J. Field Robotics*, 23:839–861, 10 2006.

[15] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[16] Ondrej Chum, Jiri Matas, and Josef Kittler. Locally optimized ransac. volume 2781, pages 236–243, 09 2003.

[17] Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6, April 2013.

[18] F. Majer, Z. Yan, G. Broughton, Y. Ruichek, and T. Krajník. Learning to see through haze: Radar-based human detection for adverse weather conditions. In *2019 European Conference on Mobile Robots (ECMR)*, pages 1–7, 2019.

[19] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2017.

[20] Naonori UEDA and Ryohei NAKANO. Deterministic annealing em algorithm. *Journal of the Society of Instrument and Control Engineers*, 38(7):444–449, 1999.

[21] X. Wang, R. Hänsch, L. Ma, and O. Hellwich. Comparison of different color spaces for image segmentation using graph-cut. In *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, volume 1, pages 301–308, 2014.

[22] M. Velas, M. Spanel, M. Hradis, and A. Herout. Cnn for very fast ground segmentation in velodyne lidar data. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 97–103, 2018.

[23] Phuong Minh Chu, Seoungjae Cho, Jisun Park, Simon Fong, and Kyungeun Cho. Enhanced ground segmentation method for lidar point clouds in human-centric autonomous robot systems. *Human-centric Computing and Information Sciences*, 9(1):17, 2019.

[24] John Hershberger and Jack Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. *5th Intl Symp on Spatial Data Handling*, 11 2000.

Appendix A

List of abbreviations

In Table A.1 are listed abbreviations used in this thesis.

Abbreviation	Meaning
MBZIRC	Mohamed Bin Zayed International Robotic Challenge
EM	Expectation maximization
RANSAC	Random sampling consensus
CNN	Convolutional neural network
ROS	Robot operation system
lidar	Light detection and ranging
MLE	Maximum likelihood estimate
UAV	Unmanned aerial vehicle
UGV	Unmanned ground vehicle
CPU	Central processing unit
GPU	Graphical processing unit
AMCL	Adaptive Monte Carlo localization
IEPF	Iterative endpoint fit

Table A.1: Lists of abbreviations