DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Thesis title

Jakub Šťastný

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Thesis title

Titel der Abschlussarbeit

Author:	Jakub Šťastný
Supervisor:	Supervisor
Advisor:	Advisor
Submission Date:	15.09.2022

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.09.2022 Jakub Šťastný

Acknowledgments

Abstract

Here goes abstract

Kurzfassung

Hier geht die Kurzfassung

Contents

Acknowledgments **iii**

Abstract **iv**

Kurzfassung **v**

1. Introduction **1**
1.1. Motivation 3
1.2. Objectives 3
1.3. Work contribution 3
1.4. Thesis Framework 3
1.5. Outline 4

2. Related Work **5**
2.1. SLAM Problem 5
2.2. Biologically inspired SLAM systems 5
2.2.1. OpenRatSLAM 5
2.2.2. Other RatSLAM variants 5

3. Technical Background **6**
3.1. Robot Operating System 6
3.2. 3D Lidar Sensors, Point clouds and Colored Point clouds 8
3.3. Gazebo 8
3.4. Turtlebot3 8
3.5. RViz 8
3.6. OpenCV 9
3.7. Pytorch 9
3.8. PointNet 9
3.9. Color Models 9

4. Methodology **10**
4.1. Problem definition 10
4.2. System architecture 10
4.3. Data fusion and preprocessing 10
4.3.1. Sensors synchronization 10
4.3.2. Data fusion 12
4.3.3. Ground removal 13

4.4. General place recognition outline 14
4.5. Hierarchical view matching based on Clustering 14
4.5.1. Information extraction 15
4.5.2. Template matching 15
4.6. Neural Networks based view matching 15
4.6.1. Siamese neural networks 15
4.6.2. Template matching 15
4.7. 2-stage view matching 15
4.7.1. Template matching 15

5. Experiments **16**
5.1. Used environments 16
5.2. Evaluation metrics 16
5.3. Experiment system setup 16
5.4. TODO outline for results and discussion 16

6. Conclusion and future work **17**

7. Template **18**
7.1. Section 18
7.1.1. Subsection 18

A. General Addenda **21**
A.1. Detailed Addition 21

B. Figures **22**
B.1. Example 1 22
B.2. Example 2 22

List of Figures **23**

List of Tables **24**

Bibliography **25**

1. Introduction

Nowadays, mobile robots are becoming more and more popular. Whether it is an autonomous car, a transport robot in a warehouse, or an automatic vacuum cleaner, these robots are finding more and more applications and becoming a part of most people's everyday lives.

One of the essential tasks every mobile robot must be able to solve is finding a valid path from its current position to the target based on the obstacles and surroundings. Knowledge of the robot's precise location and environment map is a necessary prerequisite for almost any navigation and planning algorithm.

Even if most of the world is already mapped [TODO reference] on a large scale and due to GPS, Gallileo, GNSS, etc., [TODO refs] we can locate ourselves on these maps with satisfiable accuracy, these methods can not be usually used for the indoors or small scale environments, where the maps are typically unknown, and any global positioning services can not be used.

Furthermore, a proper self-localization of the robot depends on a precise map, and a map construction depends on the accurate positions of a robot and other landmarks. Therefore the localization and mapping problems have to be usually solved simultaneously. In addition, the environment can frequently change, which makes, together with the mutual dependence of localization and mapping, the simultaneous localization and mapping (SLAM) a very challenging problem, that most mobile robots must solve.

The SLAM has been solved by many scientists since the 1980s [TODO ref]. Until now, there have been developed many different techniques for solving the SLAM, with various advantages and disadvantages. The majority of the approaches can be separated into three main categories: conventional SLAM, visual SLAM, and biologically inspired SLAM.

The conventional algorithms are based on a probabilistic model and usually work with Light Detection And Ranging [TODO ref] and odometry [TODO ref] sensors. These techniques typically work in two steps. In the first step, the position of the robot and landmarks are extracted from the raw sensor data, usually using different filtering techniques, such as Kalman filter or particle filtering. This extracted information is used in the second step to build or update the final map.

The precision and resolution of the final maps built by these approaches are usually very high. However, there is usually a high computation and storage demand that rapidly increases with the number of landmarks. Because of this fact, the conventional techniques are generally not suitable for larger or complicated environments with a lot of landmarks and can not be performed on low-performance computation devices, such as older Raspberry PI models. Furthermore, many of these techniques usually rely on accurate sensor measurements and are not robust against more significant measurement errors that can easily destroy the whole map. [TODO some references]

The visual SLAM approaches became popular mostly during the last decade, with cameras' significant cost reduction and quality improvement. As the name suggests, these techniques are based on visual input from a 2D or 3D camera and various computer vision techniques. Compared to the conventional methods, these approaches obtain more information about the environment and, therefore, can generate more precise outputs. However, most visual SLAM methods are susceptible to ambient lighting and reflections and perform differently in different light conditions, which can cause significant errors. Furthermore, these techniques work poorly in a low-texture environment, making them unsuitable for environments with many windows, mirrors, or other glass or reflective surfaces.[TODO refs and examples]

As the name suggests, the biologically inspired SLAM approaches find their inspiration in various biological systems. The ideas behind these techniques are very diverse and differ from approach to approach. In this category, we include techniques based on machine learning, models of biological structures, or methods based on the behavior of some biological species. These techniques usually can not guarantee the result's precision but use a heuristic approach to approximate the results with specified accuracies. These techniques generally have a significantly lower demand on resources than the conventional and visual approaches. Furthermore, these techniques are usually more robust against measurement errors and can also contain a mechanism to repair the previous errors based on the new data.[TODO refs]

One of the most generally known biologically inspired SLAM systems is RarSALM, which was initially developed in 2008 and improved over the years. [TODO source] The open source version of this technique, the OpenRatSLAM [TODO source] and its ROS implementation RatSLAMRos [TODO source] brings a standardized, reconfigurable, and modulized way to include this method to any program for mobile robots. Furthermore, the RatSLAM approach shows long-term stability in the indoors and outdoors scenarios.[TODO sources]

This approach is inspired by computational models of the hippocampus of rodents, which have been extensively studied concerning navigation tasks and show many of the properties of a desirable SLAM solution. During the last 50 years, four essential kinds of neurons have been discovered connected with SLAM and navigation tasks: place cells, grid cells, head direction cells, and border cells.

Place cells, discovered by John O'Keefe in 1976 [TODO ref], are connected with different places the rodent has visited and are activated every time the rat returns to a particular location. Grid cells, discovered by Edvard and May-Britt Moser in 2008 [TODO ref], react to the rodent's movement and are activated in sequence as the rat moves around in the environment. The head direction cells allow the rodent to get the spatial sense of direction based on geometry features. Finally, the border cells are activated when the rodent moves close to a wall or other obstacle.

Place recognition is one of the crucial parts of the RatSLAM solution and any intelligent system operating autonomously over a longer period of time. The main task of this problem is to tell if the robot has visited the current place before or not, despite severe changes in its appearance due to different light conditions, weather, or non-stationary objects like pedestrians or cars.

Most standard place recognition techniques are based on visual input and usually use machine learning, feature extraction and matching, or the scene decomposition approaches.

However, some other methods exist based on entirely different ideas and kinds of sensors.

1.1. Motivation

The original RatSLAM approach uses a low-resolution camera image as an input for place recognition. However, as mentioned before, this brings some drawbacks, like sensitivity to the different light conditions and reflections. Compared to a camera, a 3D LiDAR sensor can measure directly in three dimensions with a high precision [TODO ref], even over a long distance. Furthermore, the 3D-LiDAR sensor is robust against different light conditions and reflections. On the other hand, compared to the camera, the LiDAR data lose some helpful information, like colors, that can be a critical factor in place recognition of places in the environments like office buildings with different meeting rooms that differ only in the wall color and otherwise remain identical.

The proper combination of the advantages of both these sensors can significantly improve place recognition and consequently enhance the quality of the entire RatSLAM algorithm. Furthermore, the additional odometry sensor may provide more accurate speed data, improving the SLAM quality compared to the original RatSLAM, which calculates odometry information only from the visual input.

Lastly, place recognition based on visual data requires storing whole images or extracted feature vectors, which consumes a relatively large amount of memory. The proper representation of the scene based on the LiDAR and camera data may significantly improve the required space and make RatSLAM even more suitable for the low performant computational devices.

1.2. Objectives

This thesis aims to find an optimal method of combining data from several sensors and find the best solution for the place recognition problem and, as a result, improve the precision and performance of the RatSLAM algorithm. Besides, all suggested approaches find inspiration in biological systems, like the rest of the RatSLAM approach. Furthermore, all suggested techniques shell be tested on accuracy and different kinds of performance metrics and compared to each other as well as to an image-based place recognition used in an original RatSLAM.

1.3. Work contribution

1.4. Thesis Framework

TODO

1.5. Outline

TODO

2. Related Work

2.1. SLAM Problem

2.2. Biologically inspired SLAM systems

2.2.1. OpenRatSLAM

2.2.2. Other RatSLAM variants

3. Technical Background

This chapter provides an overview of most of the techniques, frameworks, and libraries related to the thesis, including the Robot Operating System ROS, used Lidar sensors, Gazebo, Turtlebo3 models, RViz, OpenCV, Pytorch, PointNet, and different models used for the representation of colors and calculating of their differences.

3.1. Robot Operating System

Robot Operating System, shortly ROS is a popular open-source framework commonly used by many researchers and robot developers. This framework allows easy launch, deployment, and communication between different modules as standalone services that create a final complex system. This modularity permitted the creation and easy application of various tools and libraries like Gazebo, RViz, or rqt (TODO references), together with custom services. Furthermore, the ROS allows an easy spread of one system among more devices, allowing an easy run of some services directly on the robot and other services on the central computer, without almost any accessive work.

Another significant advantage of the ROS's modularity is that you can exchange the robot simulator with the real robot and sensors without touching the rest of the system. Furthermore, you can record and replay the entire system's messages during its run, allowing easy reproducibility of any experiment.

The official languages of the ROS are C++ and Python, but there are also inofficial tools that allow writing services for ROS in other languages, like, for example, LISP or Swift. (TODO references)

The following subsections briefly introduce some of the essential ROS features.

Roscore

Roscore is a service that must be executed before running any Node of the ROS system. This service will automatically start all the essential services for the ROS running, like logging service, parameter server, and ROS master. Therefore, the running roscore service is necessary for ROS nodes to communicate.

ROS Node

Ros Nodes are a crucial concept of the whole system. Single Node represents a standalone service that can process data received from other ROS Nodes, sensors, or users and send them to the other Nodes or directly to the user or system actors. The entire application usually consists of many ROS Nodes that communicate using ROS topics, services, etc.

ROS message

Messages are the most common way of communication between several ROS Nodes. These messages are transmitted between the Nods via ROS topics. The ROS message, typically stored in the .msg file, describes a format that the data transmitted on a particular ROS topic must fulfill and tells the ROS Node how to represent the received data.

According to the convention, most messages start with the Header part, containing the message sequence number, timestamp, and frame id. However, this header is optional and is not present in all commonly used messages.

ROS topic

ROS topics are named buses, serving for message exchange between the Nodes. One or more Nodes can publish messages or subscribe to each ROS topic and receive all published messages. The publishing and subscribing processes work anonymously, which means that the publisher does not know if the message was read by only one or more or any Node, and receivers do not know which node published the particular message unless it's part of the sent data.

Every topic is strongly connected with a particular message type, so every message published on the same topic must have the same format.

Roslaunch

Roslaunch is a tool allowing the launch of multiple ROS Nodes with a single command based on an XML configuration. This tool also supports setting different parameters that can be set while starting the process and passed to the particular Nodes or put on the Parameter Server. Furthermore, this tool allows renaming specified topics without the need to tell any information to the publishing Nodes.

ROS package

ROS packages help to organize the ROS software. The package can contain ROS Nodes, libraries, datasets, configuration files, etc. The goal of the ROS package is to pack together connected ROS Nodes and other tools that together create some meaningful tool, program, or library. These packages allow easy publication, deployment, installation, and integration of the third-party components into custom programs or systems.

Rosbag

Rosbag is a command line tool for recording and replaying the data exchanged between Nodes during the program run. This tool can record all messages published on all or only specified ROS topics without affecting the running system. These messages can be later replayed, which allows excellent reproducibility of all program runs and further optimization, improvement, and testing under the same conditions.

Particularly researchers with this tool record all data from the sensors while running experiments and publish them on the internet for later experiment reproduction or further development.

Catkin

Catkin is a collection of CMake macros and Python scripts, commonly used for the ROS development and building of ROS Nodes and packages. Even if catkin is not the official part of ROS, it is a widely used tool among the ROS community.

3.2. 3D Lidar Sensors, Point clouds and Colored Point clouds

TODO ?

3.3. Gazebo

Gazebo is a famous open-source 3D-simulation tool supporting robot simulation in complex 3D environments, including realistic physics, like gravity, friction, collisions, light reflections, etc. These environments can also change over time, which allows adding pedestrians, moving cars, or other non-stationary objects to the simulation.

Furthermore, Gazebo supports various types of sensors, especially HD-Camera, 3D or 2D Lidar, IMU, etc., or several plugins for the robot control. This tool also provides an interface fully compatible with ROS, which makes it a perfect simulation tool for this thesis.

3.4. Turtlebot3

Turtlebot3 offers open-source mobile robots commonly used in robotics research and development. The Turtlebo3 provides two different models, Burger and Waffle-Pi, that are fully compatible with the Gazebo simulator and contain all software necessary for the robot control and simulation. These models are also easily extensible and allow adding new kinds of sensors.

3.5. RViz

RViz is a ROS graphical interface, allowing the visualization of the information published for many different available topics. Particularly, this tool is commonly used for the real-time visualization of the data received from various sensors, generated maps, landmarks, detected objects, etc.

3.6. OpenCV

OpenCV is an open-source computer vision library available for C++ and Python. This library contains many real-time image and video processing tools, from simple transformations to feature extraction and matching or object recognition.

3.7. Pytorch

Pytorch is a python open-source machine learning framework based on the Torch library. This framework is commonly used in the area of machine learning. It contains many tools for modeling Neural Networks, several training algorithms, tools for datasets preparation, advanced matrix operations tools, etc. Furthermore, because of the CUDA support, this framework allows running most of the calculations on the GPU, which can significantly speed up the training process of most models.

3.8. PointNet

TODO

3.9. Color Models

TODO

4. Methodology

4.1. Problem definition

TODO Problem definition

4.2. System architecture

TODO System architecture

4.3. Data fusion and preprocessing

This section introduces the process of processing the raw data received from the sensors and creating a final representation of the environment that is further used as an input for the other stages of the process.

In this thesis, we receive colored 2-dimensional images from an HD camera and a raw 3D point cloud from a 3D lidar sensor. The goal is to use these two inputs and generate a colored point cloud for a more accurate environment representation.

In this work, we assume that both sensors are calibrated and that the excentric parameters, namely field of view and transformation matrix between sensor's frames, are already known. If this is not the case, some of the following well-known techniques (TODO ref) can be used for their determination.

The whole process can be divided into three major parts. In the beginning, the frequencies of the sensors must be synchronized. Then, after the input data are synchronized, the data fusion is performed, and the individual scene representations are merged into one mutual model, containing information from all individual sensors. Finally, the built representation is preprocessed and simplified by removing the unnecessary data, especially the information about the ground. The whole process is visualized in Figure 4.1 and each step is described in the following subsections.

Figure 4.2 illustrates example inputs received from the sensors (a) and (b), generated output after the data fusion (c), and final generated output after the ground removal (d).

4.3.1. Sensors synchronization

This section solves the problem that the sensors deliver data at different times with different frequencies. The goal of the synchronization is to receive unsynchronized data from both

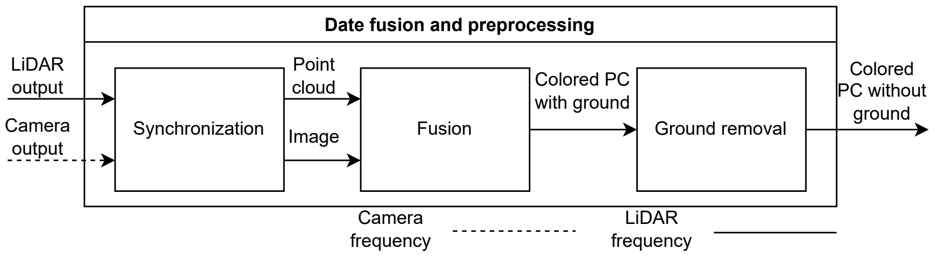
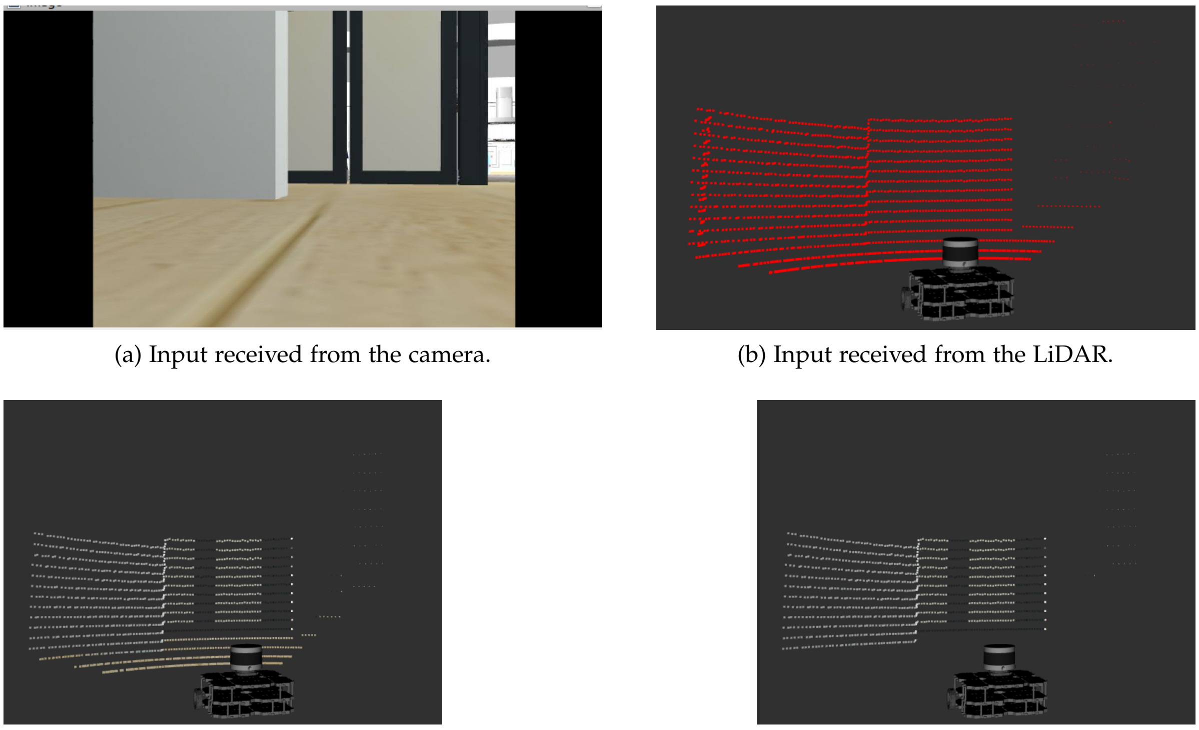


Figure 4.1.: Caption



(a) Input received from the camera.

(b) Input received from the LiDAR.

(c) Colored point cloud after data fusion

(d) Final result after the ground removal

Figure 4.2.: Data fusion sensors inputs and generated outputs after each stage

sensors and return synchronized pairs of images and point clouds that will be used for further fusion.

The technique used in this thesis is based on the assumption that the camera frequency is significantly larger than the frequency of the 3D-lidar[1]. Based on this fact, the system can just record all the images received from the camera. All point clouds received from the 3D-lidar are afterward paired with the last received image.

The time difference between the image and point cloud is so low, compared to the frequency of the lidar, that it can be neglected, and we can assume that the image and the point cloud were taken simultaneously.

4.3.2. Data fusion

At this point, we assume that all the data are synchronized and working with a pair of images and raw point clouds representing the same scene at the same time. (TODO also assumes the same frame?). The goal is to build a colored 3D point cloud.

The point cloud PC is represented as an unordered set of triples, describing a single point's x, y, z coordinates:

$$PC \subseteq \left\{(x,y,z) | (x,y,z) \in \mathbb{R}^3\right\}.$$

The image can be represented as a function $img : \mathbb{R}^2 \to \mathbb{N}_0^3 \cup \{0\}$ taking x, y pixel coordinates as input and returning RGB representation of the pixels color or 0, if the give coordinates are out of range[2]:

$$img(x,y) = \begin{cases} (r,g,b) & \text{if } (x,y) \text{ are inside of the cameras field of view} \\ 0 & \text{otherwise,} \end{cases}$$

where r, g, b corresponds to the pixel color's red, green, and blue parts.

The result CPC will be represented as an unordered set of sextuplets, describing a single point's x, y, z coordinates together with r, g, b colors:

$$CPC \subseteq \left\{(x,y,z,r,g,b) | (x,y,z,r,g,b) \in \mathbb{R}^3 \times \mathbb{N}_0^3\right\}.$$

Let $\mathbf{P}$ be the projection matrix that projects 3D world coordinates into the 2D pixel coordinates on the image plane using the following formula:

$$\mathbf{x} = \mathbf{P}\mathbf{X},$$

where $\mathbf{X} = [x_{\text{world}}, y_{\text{world}}, z_{\text{world}}]^T$ represents a 3D world coordinates vector and $\mathbf{x} = \left[x_{\text{image}}, y_{\text{image}}\right]^T$ represents a 2D image plane coordinates vector. This matrix can be found using the camera's and lidar's excentric parameters received from the calibration [TODO ref].

If we know the projection matrix $\mathbf{P}$, we can iterate through all the points in the point cloud, project them into the image plane, find the corresponding pixel and bind its color with

[1]In the experiments, we used a camera that has 15 times higher frequency than the 3D-lidar.

[2]If x or y are not the whole numbers, they are rounded to the nearest whole number.

the examined point. All points outside the camera's field of view shall be ignored and not included in the result.

The algorithm for the fusion of the data received from Lidar and camera is summarized in the Algorithm 1.

Algorithm 1 Lidar and camera data fustion

```
Input: Point cloud $PC$, camera image $img$
Output: Colored point cloud $CPC$
  $CPC := \{\}$                                   ▷ Initialize result as an empty set
  $\mathbf{P} \leftarrow$ build projection matrix
  for $(x, y, z) \in PC$ do                        ▷ Iterate through all points in $PC$
      $[x_i, y_i]^T \leftarrow \mathbf{P} \cdot [x, y, z]^T$      ▷ Project current point to the image plane
      if $img(x_i, y_i) \neq 0$ then               ▷ Ignore points out of cameras field of view
          $(r, g, b) \leftarrow img(x_i, y_i)$               ▷ Get projected pixel color
          $CPC \leftarrow CPC \cup \{(x, y, z, r, g, b)\}$    ▷ Add colored point to the result
      end if
  end for
  return $CPC$
```

4.3.3. Ground removal

Ground removal is a widely used technique, applied by many algorithms working with 3D clouds. Even if the information about the ground, particularly about the ground color, might help distinguish between two different scenes, the points representing the floor are usually in the same position for every scene and therefore carry a relatively small information value. Removing these points can significantly reduce the input size, with minimal loss of the information value, which can positively influence the algorithm's performance. Furthermore, the ground points may connect two completely separate objects and therefore have a negative influence on the scene segmentation or object detection algorithms.

Let's assume that all points from the points cloud are represented in the world's frame of reference and that y ax is orthogonal to the ground. Then, it is evident that there exists a threshold y_{th}, such that all points representing a ground have y coordinate lower or equal to the y_{th} and all other points will have y coordinate larger than y_{th}. So, if the threshold y_{th} is known[3], the ground point can be removed simply by filtering the points by their y coordinate. The ground removal process is summarized in the Algorithm 2.

Besides the ground, this method can remove some other objects' bottom parts, like the feet of pedestrians, wheels of wheelchairs, etc., which can lead to a slightly more significant information loss than pure ground removal. However, there exist other, better ground removal techniques, usually based on machine learning, which do not suffer from this issue [TODO refs]. Yet, these techniques typically require more computational resources than the simple

[3]This threshold can be very easily determined experimentally.

Algorithm 2 Ground removal

```
**Input:** Colored point cloud $CPC$, threshold $y_{th}$
**Output:** Colored point cloud $CPC_2$ without ground
  $CPC_2 := \{\}$                                   ▷ Initialize result as an empty set
  for $(x, y, z, r, g, b) \in CPC$ do               ▷ Iterate through all points in $CPC$
      if $y > y_{th}$ then                          ▷ Ignore points under the threshold
          $CPC_2 \leftarrow CPC_2 \cup \{(x, y, z, r, g, b)\}$   ▷ Add current point to the result
      end if
  end for
  return $CPC_2$
```

filtering approach, and the results are usually not significantly different[4], so we decided to prefer this simple method despite the minor quality drawbacks.

4.4. General place recognition outline

TODO

4.5. Hierarchical view matching based on Clustering

The first place recognition method presented in this work found inspiration in the human memory of places. Unlike the further proposed methods, this approach does not try to model the brain's other biological structures but tries to conceive biological inspiration more abstractly. The idea is based on the way how humans usually remember places. According to the [TODO refs], the long-term human spatial memory recall is built upon a hierarchical structure. First, people remember the general layout of the view. Afterward, they can recall basic information about the scene's largest or most significant features, followed by a few more detailed features. Finally, the small details are usually wholly forgotten or recalled at last.

This presented approach does basically the same thing. In the first step, the received scene

[4]Especially in the indoor environments with static objects like furniture, which are in the main focus of this work

4.5.1. Information extraction

4.5.2. Template matching

4.6. Neural Networks based view matching

4.6.1. Siamese neural networks

4.6.2. Template matching

4.7. 2-stage view matching

4.7.1. Template matching

5. Experiments

5.1. Used environments

5.2. Evaluation metrics

5.3. Experiment system setup

5.4. TODO outline for results and discussion

6. Conclusion and future work

7. Template

Use with pdfLaTeX and Biber.

7.1. Section

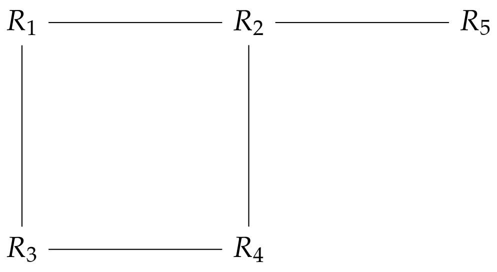
Citation test (with Biber) [1].

7.1.1. Subsection

See Table 7.1, Figure 7.1, Figure 7.2, Figure 7.3, Figure 7.4, Figure 7.5.

Table 7.1.: An example for a simple table.

A	B	C	D
1	2	1	2
2	3	2	3



$$\begin{tikzcd}
R_1 \arrow[r, dash] \arrow[d, dash] & R_2 \arrow[r, dash] \arrow[d, dash] & R_5 \\
R_3 \arrow[r, dash] & R_4 &
\end{tikzcd}$$

Figure 7.1.: An example for a simple drawing.

This is how the glossary will be used.

Donor dye, ex. Alexa 488 (D_{dye}), Förster distance, Förster distance (R_0), and k_{DEAC}. Also, the TUM has many computers, not only one Computer. Subsequent acronym usage will only print the short version of Technical University of Munich (TUM) (take care of plural, if needed!), like here with TUM, too. It can also be –> hidden[1] <–.

[(TODO: Now it is your turn to write your thesis.
This will be a few tough weeks.)]

[(DONE: NEVERTHELESS, CELEBRATE IT WHEN IT IS DONE!)]

[1]Example for a hidden TUM glossary entry.

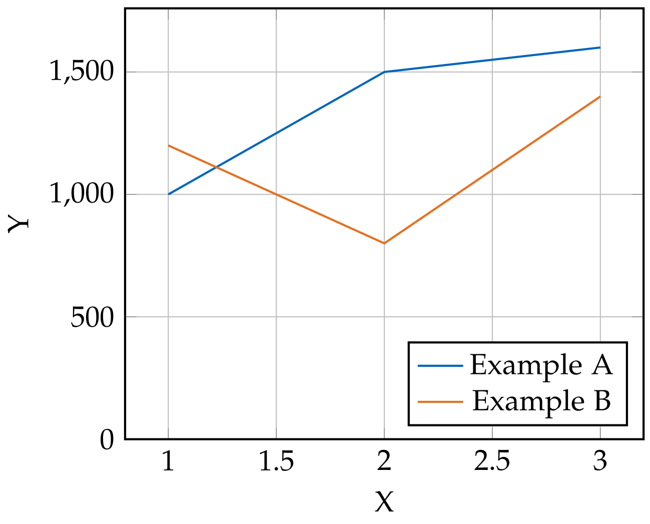


Figure 7.2.: An example for a simple plot.

```
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 7.3.: An example for a source code listing.

Figure 7.4.: Includegraphics searches for the filename without extension first in logos, then in figures.

Figure 7.5.: For pictures with the same name, the direct folder needs to be chosen.

(a) The logo.

(b) The famous slide.

Figure 7.6.: Two TUM pictures side by side.

A. General Addenda

If there are several additions you want to add, but they do not fit into the thesis itself, they belong here.

A.1. Detailed Addition

Even sections are possible, but usually only used for several elements in, e.g. tables, images, etc.

B. Figures

B.1. Example 1

✓

B.2. Example 2

✗

List of Figures

4.1. Caption 11
4.2. Data fusion sensors inputs and generated outputs after each stage 11

7.1. Example drawing 18
7.2. Example plot 19
7.3. Example listing 19
7.4. Something else can be written here for listing this, otherwise the caption will be written! 19
7.5. For pictures with the same name, the direct folder needs to be chosen. 20
7.6. Two TUM pictures side by side. 20

List of Tables

7.1. Example table 18

Bibliography

[1] L. Lamport. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Professional, 1994.