DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

LiDAR and camera data fusion for the biologically inspired SLAM systems

Jakub Šťastný

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

LiDAR and camera data fusion for the biologically inspired SLAM systems

LiDAR und Kameradaten Fusion für die biologisch inspirierten SLAM-Systeme

Author:	Jakub Šťastný
Supervisor:	Prof. Dr.-Ing. habil. Alois C. Knoll
Advisor:	Genghang Zhuang, M.Eng.
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

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.

According to biological inspiration, localization is based on odometry, inspired by the Grid cells. Odometry is well known for its problems with cumulative errors. It does not matter how precise odometry sensors or techniques are used; even a very small error adds up over time, and the whole system will be completely out of reality after a few minutes, maximally a few hours. To reduce these errors, a loop closure technique is required. Based on the biological model, the RatSLAM implements the loop closure using a place recognition technique.

So, 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.

To achieve this primary goal, several challenges need to be solved. The first challenge is data fusion. In this part, the optimal scene representation must be designed to combine most of the information from all the sensors and reduce most of the disturbing factors typical for the input sensors. Furthermore, the synchronization and mutual calibration of the sensor must be solved.

The other challenge is to solve the place recognition problem based on the representation from the fused data. In this part, we need to think apart from accuracy to the performance and memory consumption in order to make this approach available for low-performance devices.

There will be suggested several algorithms with different expected advantages and disadvantages. All proposed techniques will be tested on accuracy and various 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

TODO

1.4. Outline

Chapter 2 presents a related work about various scene recognition approaches and biologically inspired SLAM systems
Chapter 3 describes all used frameworks and tools used in this work
Chapter 4 provides a detailed description of all the algorithms and techniques implemented in this thesis. Namely, the complete system overview is provided in the beginning, followed by the solution to the data fusion problem. Afterward, three different place recognition techniques are offered.
Chapter 5 TODO
Chapter 6 TODO

2. Related Work

2.1. Scene recognition problem

Scene recognition is a problem that received a lot of attention from many researchers and engineers in the last few years. Even if approaches for several sensors were developed, the most popular is visual scene recognition based on images from a camera.

This section presents some of the most popular approaches for visual scene recognition based on traditional methods and machine learning.

2.1.1. Local image descriptors

This technique aims to find, describe and compare significant features from the images. Each image is processed in two phases: detection and description. Since both stages can be solved independetelly, a large number of various approaches for each phase have been developed.

The detection phase aims to detect all essential features in a given image, such as edges, corners, significant points, or objects. The feature detection algorithms generate pixel coordinates of each feature, usually with an occupied area. There are many different approaches to this problem, like FAST [TODO ref], Laplacian of Gaussian [TODO ref], SUSAN [TODO ref], and many more, detecting different kinds of features.

TODO image of feature detection example

The goal of the description phase is to provide a summary of the image information around each feature. The feature is represented as its position in the image, and the output is defined as an N-dimensional vector. A good feature descriptor should fulfill three following rules: Repeatability, Distinctiveness, and Efficiency. The repeatability means that the feature descriptor is robust and invariant to the image's translation, rotation, or illumination changes. Distinctiveness represents the ability to distinguish between two close features. Finally, due to real-time processing, which is increasingly applied nowadays, efficiency also plays an important role. Among popular approaches to solve this problem can be included SURF [TODO ref], GLOH [TODO ref], BRIEF [TODO ref], and many more.

Using these two stages, the set of local image descriptors

$$\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}]^T$$

is extracted, in which $\mathbf{x}_i \in \mathbb{R}^k$. However, each image can contain hundreds of individual features, which can be impractical in real-time processing and requires a huge amount of memory. To lower the dimensions of the local descriptors vector, different techniques like bag-of-visual-words [TODO ref], VLAD [TODO ref], or Fisher kernel [TODO ref] may be applied to aggregate the vector $\mathbb{X}$ into a more compact single vector. Finally, these vectors are

compared to decide if two images represent the same place. The precise comparison technique depends on the techniques used in the description phase and for the final descriptors' vector compression.

2.1.2. Global image descriptors

This technique works similarly to the previous one, with one big difference. In this approach, the detection phase is wholly omitted, and the whole image is considered as only one feature. Therefore, the feature description algorithms must be modified to suitably describe large and diverse features. Some alternations of the algorithms provided in the last part might be WI_SURF [TODO ref] or BRIEF-GIST [TODO ref].

2.1.3. Neural networks based approaches

Convolutional neural networks achieve outstanding performance on several recognition or classification tasks, including a solution to scene recognition problems [TODO refs]. The basic idea behind this approach is similar to the presently presented technique. In the first step, the global feature descriptor is extracted from the image and afterward compared with the other extracted feature vectors. However, unlike the previously presented technique, this approach uses machine learning instead of classical techniques to perform these two tasks.

According to the studies [TODO refs], the features extracted from images using convolutional neural networks significantly outperform the features extracted by classical algorithms like SIFT. There are many different kinds of convolutional neural networks used for these purposes. From the most popular ones, we can mention VGGNet [TODO ref] or GoogleNet [TODO ref].

2.2. Biologically inspired SLAM systems

2.2.1. RatSLAM

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. 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.

3.3. 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.

3.4. 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.

3.5. 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.

3.6. 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.

3.7. PointNet

PointNet [TODO ref] is a deep neural network widely used for point cloud processing by many applications. This network can extract the feature vector from any point cloud, invariant to the order of the points and the rotation of the scene that the input cloud represents. Furthermore, the input size in terms of the number of points can vary for each input. There is available a C++ implementation [TODO ref] of the network, as well as a Python implementation in TensorFlow [TODO ref] or PyTorch [TODO ref] version.

Except for the feature extraction, the implementations of the PointNet network also include additional networks for scene segmentation and object classification based on the extracted feature vector from the PointNet network. Furthermore, all required learning and dataset preparation algorithms are also included.

3.8. Color Models

There are many different ways to represent a single color on a computer. Except for the well-known RGB representation, there are many different other representations with distinct advantages. Lab [TODO refs] representation is for the work the most interesting because it allows easy color difference computation, using CIE76 [TODO ref] and CIE2000 [TODO ref] that correspond in the best way to the difference told by humans.

Similarly, as in an RGB format, the colors are represented using three different components, L, a, and b. The L part stands for lightness and means how light or dark the color is. The a and b components represent the balance between two different colors. The a part describes the balance between green and magenta, and the b part represents the balance between blue and yellow. Even if this color scheme can describe the whole space of colors, the colors visible by human eyes are located in a sphere, displayed in Figure 3.1.

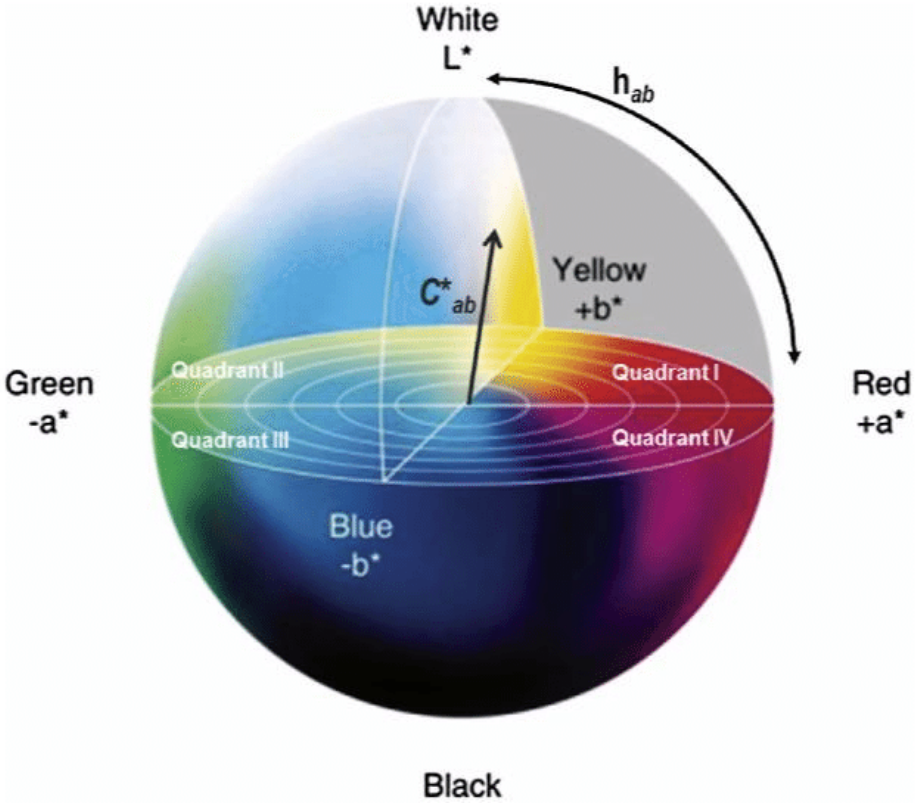


Figure 3.1.: Visualization of an Lab color representation [TODO ref]

4. Methodology

This chapter provides a detailed description of all the algorithms and techniques implemented in this work. At the beginning of this chapter, the problem was precisely defined. The following section describes a basic system overview with all used components and descriptions

In this thesis, we are mainly solving two problems: data fusion and place recognition. The approach to solving the data-fusion is described in the third section. The rest of the chapter describes three methods for solving place recognition problems.

Furthermore, the second section also presents how to connect the data fusion and place recognition algorithm to the RatSLAMRos to solve the SLAM problem.

4.1. Problem definition

1. The place recognition algorithm receives the data from the unsynchronized sensors, namely the 3D-point cloud from the LiDAR and RGB image from an HD camera. The goal is to recognize if the robot is in the current scene for the first time or if the scene has already been visited. Furthermore, if the scene was already visited, the algorithm must precisely tell which previously visited scene corresponds to the current one. The following steps must be solved:
   - Data synchronization
   - Data fusion
   - Place recognition

   - **Input:** 3D Point cloud and RGB image
   - **Output:** Id of a corresponding previously visited scene or a new id for the scenes visited for the first time
2. The place recognition algorithm is integrated with the RatSLAMRos system to solve the SLAM problem.
   - **Input:** 3D Point cloud, RGB image, odometry
   - **Output:** estimated robot path

4.2. System architecture

This section presents an overview of the architecture of the whole system. The system is decomposed into several Nodes. Together with the RatSLAMRos and simulator, the system uses three custom nodes: Data fusion, LV Builder, and LV matching. Furthermore, the system contains two additional nodes that do not influence the algorithm's run but are used for the performance analyses and other help utilities, helping with the development process. The system architecture is presented in Figure 4.1. The visualization and topics used only for visualization purposes are omitted for clarity.

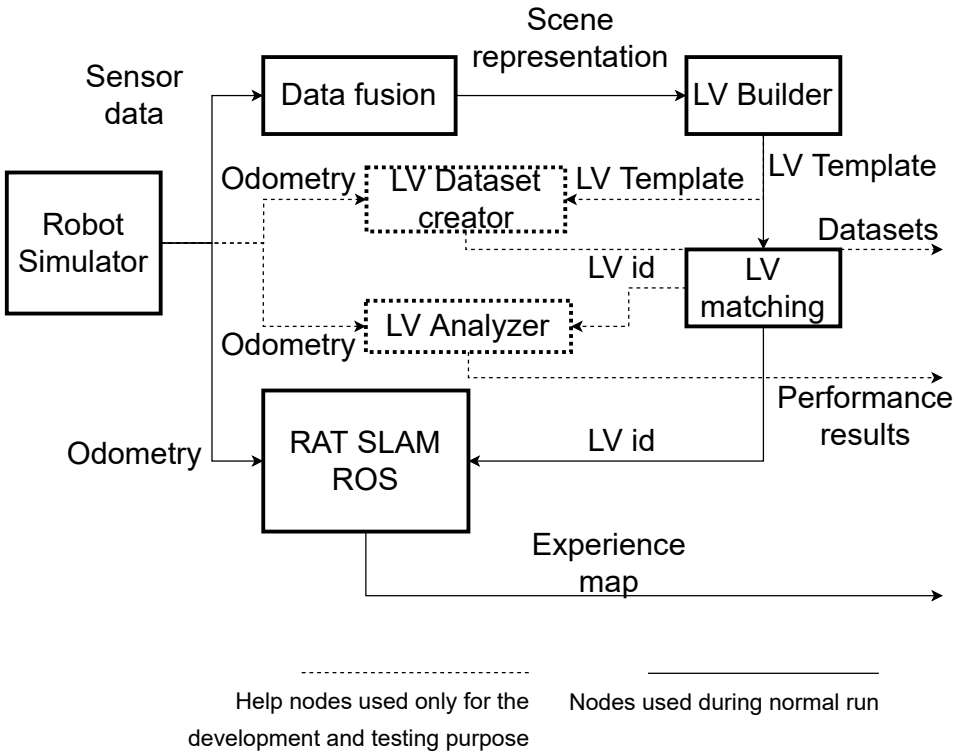


Figure 4.1.: Diagram of the ROS nodes in the whole system

4.2.1. Robot Simulator

This node represents the whole simulation and can be easily replaced with a physical robot. The robot is simulated using a Gazebo simulator, introduced in chapter [TODO chapter ref]. Besides much other information about the scene and the robot, the simulation publishes data from sensors, namely HD Camera and 3D LiDAR, and odometry data, further consumed by other nodes.

Table 4.1.: Relevant topics published by the simulator

Topic	Type	Mode
camera/rgb/camera_info	sensor_msgs::CameraInfo	publish
camera/rgb/image_raw	sensor_msgs::CompressedImage	publish
velodyne_points	sensor_msgs::PointCloud2	publish
odom	nav_msgs::Odometry	publish

4.2.2. Data fusion

The goal of this node is to take raw data from the sensors and create a single representation of the environment, combining the advantages of each sensor. This node takes care of the synchronization of the sensors, dealing with different frames of reference and the fusion itself. The data fusion node subscribes to the camera and LiDAR topics published by simulation or a real robot and publishes one topic with the final environment representation. How this node exactly works is described in the section [TODO chapter ref].

Table 4.2.: Subscribed and published topics by the Data fusion node

Topic	Type	Mode
camera/rgb/camera_info	sensor_msgs::CameraInfo	subscribe
camera/rgb/image_raw	sensor_msgs::CompressedImage	subscribe
velodyne_points	sensor_msgs::PointCloud2	subscribe
rgb_cloud	sensor_msgs::PointCloud2	publish

4.2.3. LV Builder

This node subscribes single topic published by the Data Fusion node. The main task of this node is to take the environment representation and reformat it into a suitable template that will be stored in a memory and compared with previously visited scenes. This node publishes a topic with the built template and two other topics for visualization and debugging purposes. Three different approaches to the exact implementation of this node are described in the sections [TODO chapters ref].

4.2.4. LV Matching

This node subscribes to a topic published by the LV Builder node. The goal of this node is to store all visited places, assign unique ids to all seen scenes, compare a received scene with other scenes, and decide if it was already seen. If the currently received scene template is similar enough to some of the stored templates, this node publishes the id of the matched

[1]Self defined message, see [TODO msg description and reference]

Table 4.3.: Subscribed and published topics by the LV Builder node

Topic	Type	Mode
rgb_cloud	sensor_msgs::PointCloud2	subscribe
current_scene_descripion	msgs::LVDescription[1]	publish
clustered_viz	sensor_msgs::PointCloud2	publish
convex_hull_viz	sensor_msgs::PointCloud2	publish

template. If there is no such template, this node generates and publishes a brand new id and stores the received template as a newly visited place. The only topic this node is publishing is a topic with ids of matched or new scenes.

Table 4.4.: Subscribed and published topics by the LV Matching node

Topic	Type	Mode
current_scene_descripion	msgs::LVDescription	subscribe
LocalView/Template	ratslam_ros::ViewTemplate	publish

4.2.5. Rat SLAM Ros

This diagram block represents not a single node but a whole package from several nodes. This is an original RatSLAMRos described in chapter [TODO chapter ref], run without almost any changes. The only difference compared to the original package is the number of started nodes. The lv Node is not activated because its job is done by LV Builder and LV Matching node, and the visual odometry Node is replaced by an odometry sensor from the simulator. The package subscribes to the topic with scene ids published by the LV matching node and to the topic with odometry directly from the simulator. The only relevant published topic is the experience map topic, with the final experience map, which is the final result of the whole algorithm.

Table 4.5.: Relevant subscribed and published topics by the RatSLAMRos package

Topic	Type	Mode
LocalView/Template	ratslam_ros::ViewTemplate	subscribe
odom	nav_msgs::Odometry	subscribe
ExperienceMap/Map	ratslam_ros::TopologicalMap	publish

4.2.6. LV Analyzer

This node subscribes to the same topics as the Rat SLAM Ros package. This node serves only debugging purposes and does not influence the algorithm. The goal of this node is to evaluate the performance of the algorithm. It receives all the ids of matched and new ids from the LV Matching node and can pair them with the exact position of the robot at the time the scene template was taken because of the information received from the simulator. This node remembers the precise position of each scene newly remembered by the LV Matching node and can calculate the position difference between each match and the original matched scene. Furthermore, it can calculate the distance to the nearest previously visited scene for each newly added template. According to the metrics described in the section [TODO section ref], the number of false positive and false negative matches can be calculated, leading to the approach's final accuracy, recall, and precision. These estimated numbers are the final output of this node.

Table 4.6.: Subscribed topics by the LV Analyzer node

Topic	Type	Mode
LocalView/Template	ratslam_ros::ViewTemplate	subscribe
odom	nav_msgs::Odometry	subscribe
camera/rgb/image_raw	sensor_msgs::CompressedImage	subscribe

4.2.7. LV Dataset creator

This node works in principle similar to the LV Analyzer node. The goal of this node is to pair the templates with their exact positions received from the simulator. Unlike the LV Analyzer node, this node receives and pairs the whole template from the LV Builder node instead of only the scene id from the LV Matching node. The set of pairs of templates and their exact positions is the output of this node, which helps build final datasets for automatic parameter tuning, see chapter [TODO chapter ref], or neural networks training, see chapter [TODO chapter ref].

Table 4.7.: Subscribed topics by the Dataset creator node

Topic	Type	Mode
current_scene_descripion	msgs::LVDescription	subscribe
odom	nav_msgs::Odometry	subscribe

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 4.2 and each step is described in the following subsections.

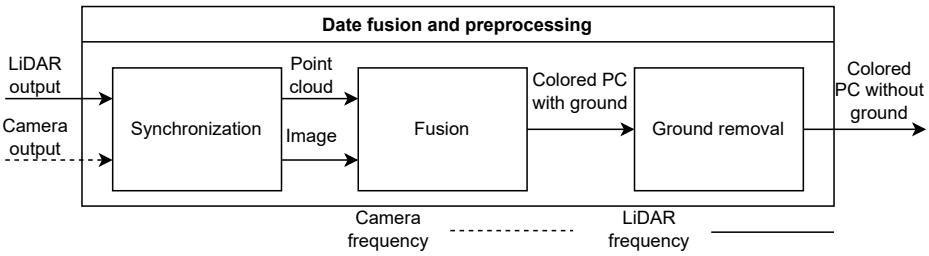


Figure 4.2.: Caption

Figure 4.3 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).

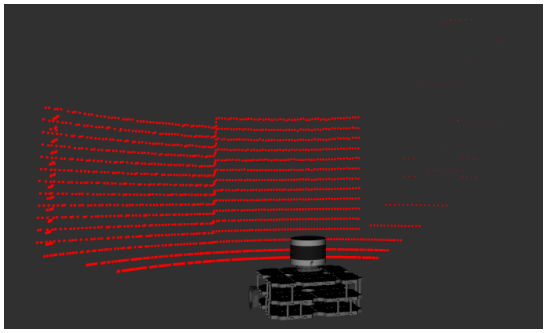
4.3.1. Sensors synchronization

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

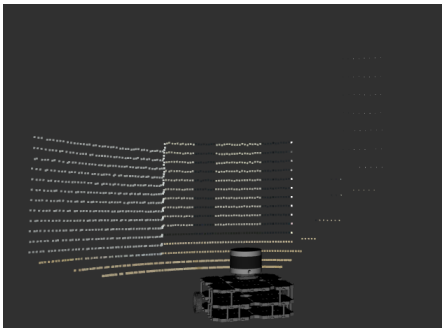
The technique used in this thesis is based on the assumption that the camera frequency is significantly larger than the frequency of the 3D-lidar[2]. Based on this fact, the system can just record all the images received from the camera. All point clouds received from the 3D-lidar are afterward paired with the last received image.

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

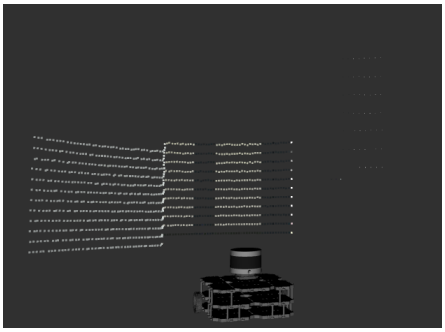
(a) Input received from the camera.



(b) Input received from the LiDAR.



(c) Colored point cloud after data fusion



(d) Final result after the ground removal

Figure 4.3.: Data fusion sensors inputs and generated outputs after each stage

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[3]:

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

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:

$$\mathbf{x} = \mathbf{PX},$$

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 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.

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

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
P ← build projection matrix
for (x, y, z) ∈ PC do                       ▷ Iterate through all points in PC
    [x_i, y_i]^T ← P · [x, y, z]^T          ▷ Project current point to the image plane
    if img(x_i, y_i) ≠ 0 then               ▷ Ignore points out of cameras field of view
        (r, g, b) ← img(x_i, y_i)           ▷ Get projected pixel color
        CPC ← CPC ∪ {(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[4], 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 filtering approach, and the results are usually not significantly different[5], so we decided to prefer this simple method despite the minor quality drawbacks.

[4]This threshold can be very easily determined experimentally.
[5]Especially in the indoor environments with static objects like furniture, which are in the main focus of this work

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) ∈ CPC do              ▷ Iterate through all points in CPC
    if y > y_th then                         ▷ Ignore points under the threshold
        CPC_2 ← CPC_2 ∪ {(x, y, z, r, g, b)} ▷ Add current point to the result
    end if
end for
return CPC_2
```

4.4. General place recognition outline

After the data are fused into a uniform scene representation, the place recognition is performed. This algorithm aims to remember all places the robot has visited and to decide if the current scene, received from the sensors, has been already visited or not, and eventually return the unique identification of the previously visited place.

In order to achieve this goal, we need to find a suitable representation (further template) of the place that will be remembered by the robot. This template must be created from the fused scene description received from the sensors, should occupy little space, and be easy to compare.

The place recognition algorithm works in three steps. In the first step, the designed template is built from the received fused scene representation. In the second step, this template is independently compared with all the stored templates, representing previously visited places. A template comparison result is a real number between 0 and 1, representing the similarity of both templates. The closer the similarity to 1, the more similar the templates are. After all the templates are compared, the most similar template is chosen. In the last step, the similarity of the most similar template is compared to the given threshold. If it is greater, the most similar template is returned as the same place. If it is smaller, the current template will be remembered, and the result will be that this place was not visited before. The whole process is visualized in Figure 4.4.

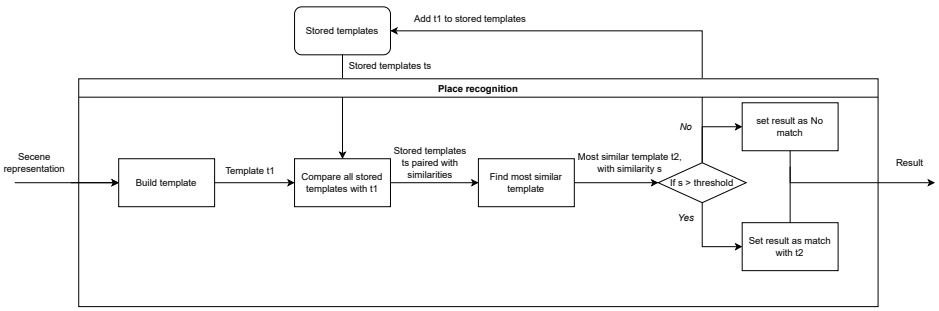


Figure 4.4.: Common place recognition workflow

The following sections suggest three different approaches to place recognition. All presented techniques differ in template structures and template building and comparison algorithms. Otherwise, they all share the same structure shown in this section.

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.

The template representation is used based on this behavior. The scene is decomposed into a set of objects based on their location, connection, and color. Each object is simplified as a list of its most significant features, namely the position of its center point, volume and area of its convex hull, and its most significant color. Together with this information, the number of points from the colored point cloud in this object is stored. Afterward, the small, insignificant objects are entirely ignored.

During the matching process, the scenes are compared according to these objects. Objects are paired by their similarities based on their stored properties' similarities. The final similarity is calculated as a weighted average of these similarities by the number of points connected with the object. In this way, the largest, usually most significant objects influence the result more than the smaller details.

TODO images of an example - scene/representation

4.5.1. Template extraction

The decomposition of the colored point cloud into the set of objects description is a two-step process. In the first step, the scene is decomposed into clusters, representing individual objects. Afterward, in the second step, all clusters are processed independently, and the objects' information is extracted.

The clustering is performed using the K-means algorithm [TODO reference] in a six-dimensional space. The first three dimensions standardly represent the points' x, y, and z positions. The additional three dimensions represent points' colors, namely the red, green, and blue components. The standard K-means algorithm assumes that all dimensions are in the same units. Because there is obviously no standard conversion between color and distance, the color dimensions must be scaled by a suitable scaling factor. The best coefficient has been found experimentally, as well as the K-means parameters.

TODO image of scene before and after the clustering

After the scene is decomposed, each cluster is processed independently of the others. Before the information about the object is extracted, its convex hull is found using the quick-hull algorithm [TODO ref]. After the convex hull is known, its center, volume, and area can be easily calculated. [TODO reference] Together with the information obtained from the convex hull, the information about the object's color is extracted. Based on the properties of the K-means algorithm, it can be expected that all points from the cluster have a similar color. Therefore, storing average color instead of all individual colors leads to negligible information loss. The average color is calculated as an arithmetic average of the red, green, and blue parts of the colors of individual points in the cluster. After calculating, the average color is converted into the chosen color format. [TODO chapter reference] The last stored information about the object is the cluster size[6].

All clusters with a size smaller than the given threshold are considered small insignificant objects and are ignored even before the object information extraction process starts.

TODO diagram of the whole process

4.5.2. Template matching

Before calculating the similarity between two whole scenes, we need to describe an approach for comparison of two objects based only on the object properties. We compare four essential properties of the objects: distance between objects, the difference between colors, sizes, and shapes of the objects. The distance of the objects is calculated as the euclidean distance of their centers. The formulas described in chapter [TODO chapter ref] are used for the color differences. The size difference is calculated as a difference between the volume of the convex hulls of the objects. For the determination and comparison of the exact shapes, we do not have enough information. However, much information about the shape of many objects is encoded in a ratio between their volume and area. There are, of course, objects with entirely different ratios between volume and area, but for most pairs with different shapes, the ratio between volume and area is also different. So the difference between shapes is represented as a difference in the ratio of volume and area of the convex hulls of the objects.

At this point, the absolute difference between objects is known for each essential property, and we have to decide if they are similar enough or not. Therefore the function is needed for each property that takes the property difference as an input and returns a number between 0 and 1, representing the similarity of the individual property. In this work, we choose

$$f_{a,x_0}(x) = 1 - \frac{1}{1 + e^{-a(x-x_0)}}$$

as the wanted function for each property, where x is the input, and a and x_0 are parameters specific for each property. This function, illustrated in Figure 4.5, has a property that for differences smaller than the threshold, it is very close to one, for differences larger than the threshold, it is close to 0, and for inputs in the neighborhood of the threshold, it is gradually decreasing. The way how to find the suitable parameters is described in the following section.

[6]Number of points in the cluster

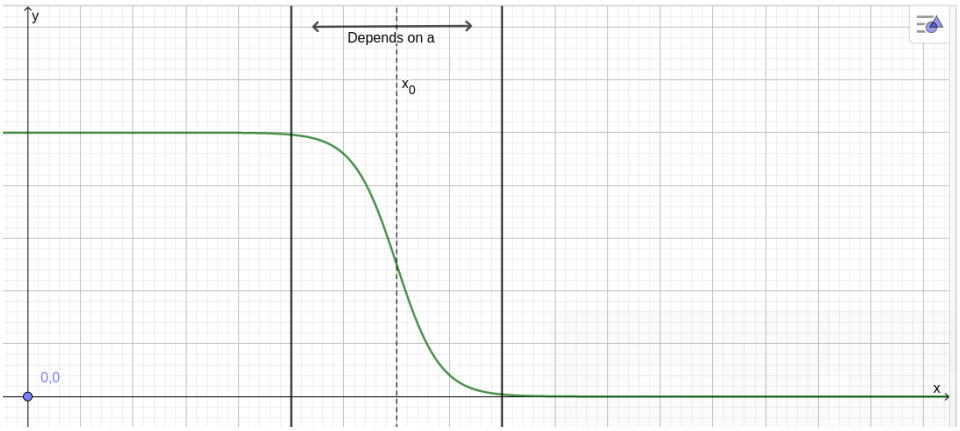


Figure 4.5.: Parametrized sigmoid function

After the similarity of each property is calculated, the final similarity of the objects is calculated as a weighted average of the similarities of the individual properties. The weights are found automatically, as described in the following section. The object matching process is summarized in Algorithm 3.

Algorithm 3 Objects comparsion

Input: Objects O_1 and O_2
Output: Similarity of the objets
$x_1 \leftarrow$ difference in colors between O_1 and O_2
$x_2 \leftarrow$ difference in positions between O_1 and O_2
$x_3 \leftarrow$ difference in volume between O_1 and O_2
$x_4 \leftarrow$ difference in volume/area ration between O_1 and O_2
$s_1 \leftarrow f_{a_1,x_{0,1}}(x_1)$
$s_2 \leftarrow f_{a_2,x_{0,2}}(x_2)$
$s_3 \leftarrow f_{a_3,x_{0,3}}(x_3)$
$s_4 \leftarrow f_{a_4,x_{0,4}}(x_4)$
$res \leftarrow \frac{w_1 s_1 + w_2 s_2 + w_3 s_3 + w_4 s_4}{w_1 + w_2 + w_3 + w_4}$
return *res*

The method for comparison of the two scenes uses the above-described approach for comparison of the individual objects. First, let's define a scene as an unordered set of objects. Then, let's define the primary scene as the current scene received from the sensors and the secondary scene as the scene from the storage. This approach iterates through all objects in the primary scene and, for each object, calculates the similarities with all objects in the secondary scene. Afterward, for each object in the primary scene, the most similar object from the secondary scene is picked[7]. Finally, the similarities with the most similar objects are used to calculate the average, weighted by the sizes of the clusters. The whole process is

[7]Two objects in the primary scene may have the same most similar object from the secondary scene

summarized in the Algorithm 4.

Algorithm 4 Scenes comparsion

Input: Primary scene S_1 and secondary scene S_2
Output: Similarity of the scenes

```
res := 0
sizesTotal := 0
for o1 ∈ S1 do
    best := 0
    for o2 ∈ S2 do
        best ← max(best, CompareObjects(o1, o2))
    end for
    res ← left + best · clusterSize(o1)
    sizesTotal ← sizesTotal + clusterSize(o1)
end for
res ← res / sizesTotal
return res
```

4.5.3. Automatic parameter tuning

To make the templates comparison algorithm work, 13 different parameters must be correctly set. Namely, a and x_0 parameters for the sigmoid functions for each of four property differences, the weight of each property, and the final threshold. There are naturally many combinations, and every change may strongly influence the results and the optimal values of the other parameters, so manual setting of these parameters wouldn't bring satisfactory results. Therefore, an automatic optimization method that minimizes the number of errors based on the choice of the correct parameters is required.

In this work, we used genetic algorithms. [TODO reference] The minimized fitness function is a count of false positive and false negative evaluations after a simulation of a robot run in a prepared environment. The simulation, environments, and evaluation metrics are the same as those used for the performance testing and are described in chapters [TODO chapters reference]. In some cases, like usage of the approach for the 2-Stage matching, see chapter [TODO chapter reference], it might be beneficial to minimize the number of false positives at the expanse of false negatives or vice versa. In this case, we can appropriately weigh the number of false positives and negatives in the final sum.

4.6. Neural Networks based view matching

The second technique offered in this work is based on neural networks. This approach is not designed as a standalone method but is presented only as a proof of concept and further used in the 2-Stage approach presented in the chapter [TODO chapter reference]. The used neural network is inspired by siamese neural networks [TODO ref].

These networks usually consist of two parts and are typically designed to compare any two entities. The first part consists of two networks with the same structure and usually with shared weights. This part is used for the automatic feature extraction from the input entities. The second part consists of a single network taking feature vectors extracted by both neural networks in the first part, performing the desired operation on the entities, and building corresponding output. In the case of comparison network is the output of the second part, and therefore of the whole network, a single number between 0 and 1, determining the similarity of the entities. The typical structure of a siamese neural network is shown in Figure 4.6.

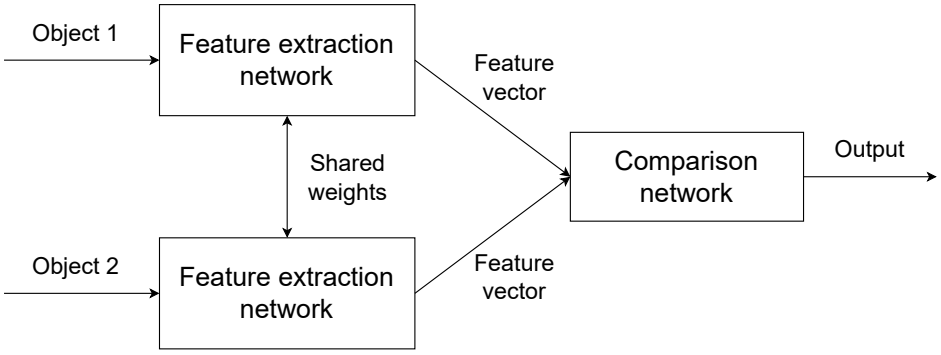


Figure 4.6.: Example of a siamese network used for object comparison

4.6.1. Networks structure

In this work, a slightly modified PointNet network, see chapter [TODO chapter ref], is used as a feature extractor. The only modification compared to the original networks is in the sizes of the layers and the output because we are typically working with significantly more sparse point clouds than the network designers[8]. As a result, the extracted feature vector contains only 256 numbers instead of 1024.

The feature vector extracted by this network is the scene template extracted from each input, processed in the future steps, and eventually stored in the storage.

The two extracted feature vectors are further compared by a single multilayer perceptron network (further MLP). This network has 512 input neurons and produces a single output. The network takes as input a concatenation of both feature vectors and produces a number between 0 and 1, describing the similarity of the scenes represented by given feature vectors. Several architectures of the networks were tried, but the best results were produced by a network with a single hidden layer containing 256 neurons and with a sigmoid as an activation function on the output neuron.

[8]The exact parameters of the network are stated in the attachments [TODO add params to the attachment]

4.6.2. Training of the networks

Both networks are trained separately. For the training of the PointNet, used for the feature extraction, we used an original algorithm presented in [TODO reference]. In order to be able to evaluate the results and compare them with the expected results in the dataset, the PointNet network was concatenated with a simple MLP network used for the object classification. Then, the whole classification network was trained on the [TODO dataset link] dataset. After the classification network was trained, the final MLP network was removed, and the first PointNet part was used as a trained model for the feature extraction.

The second part of the network, namely the MLP comparing two feature vectors, was trained using a backpropagation algorithm [TODO ref] based on the mean squared error loss function [TODO ref]. The dataset was generated from a sample simulation, see chapter [TODO chapter ref]. During the simulation, the trained feature extractor network creates a feature vector from each scene. Afterward, a set of all pairs of the feature vectors is made. A random subset was chosen from this set, and all couples from this subset were labeled using exact scene locations received from the simulator and the criteria described in the chapter [TODO chapter ref]. This labeled subset was used as a training set for the MLP network.

4.7. 2-stage view matching

This approach tries to combine both previously presented methods to achieve better results. The first approach works pretty well in many scenes but has one drawback. Since the objects' shapes are mainly ignored, taking into account only the ratio between the volume and the area of the objects, there could be many scenes with similarly placed objects with similar sizes but different shapes. This could lead to many unwanted false positive evaluations. This approach tries to use neural networks presented in section [TODO section ref] to eliminate these false negatives while maintaining most of the advantages of the first approach, described in section [TODO section ref].

4.7.1. Template building

The scene representation in this method combines the scene representations from both previously presented techniques. The scene is represented as a set of objects description, as described in chapter [TODO chapter ref], and simultaneously, the feature extraction described in the chapter [TODO chapter ref] is performed. The Final representation is a tuple of the object decomposition and features vector.

4.7.2. Template matching

As the name of the approach suggests, the matching is performed in two stages. In the first stage, the similarity of the two scenes is calculated the same as in the chapter [TODO chapter ref]. After the similarity is computed, it is compared with a given threshold so that it is predecided, if it is positive or negative. The calculated similarity is returned as a final result

if it is negative. If it is positive, the second stage is performed in order to reduce the number of false positives.

In the second stage, the feature vectors are compared in the same way as in a section [TODO section ref]. If the result is higher than another threshold, the result of the first stage is returned. If the result is lower than the threshold, 0 is returned. The whole process is summarized in Figure [TODO ref].

TODO diagram image

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

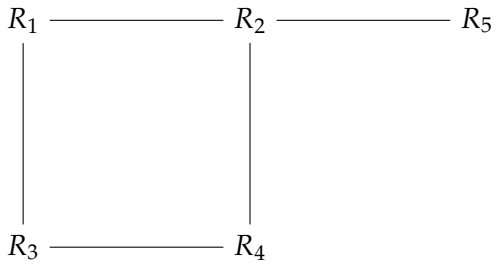


Figure 7.1.: An example for a simple drawing.

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.

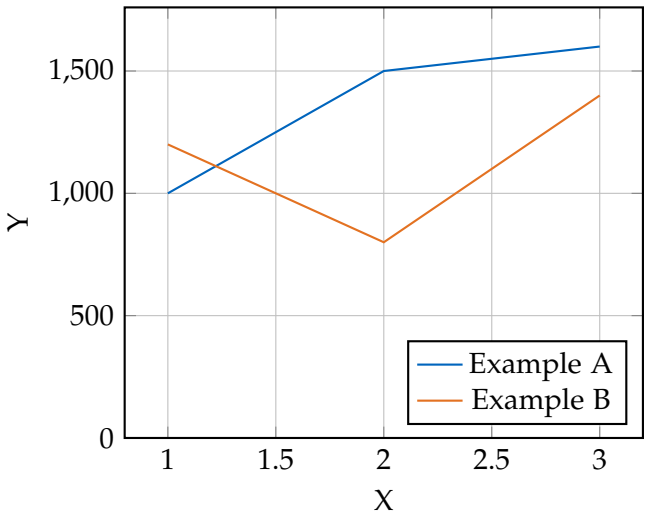


Figure 7.2.: An example for a simple plot.

```
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

List of Tables

Bibliography

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