Informatics

Visual Simultaneous Localization And Mapping Evaluation on a Mobile Robot Platform

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Martin Haar, BSc
Matrikelnummer 01625753

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao. Univ. Prof. Dipl.-Ing. Dr.techn. Markus Vincze
Mitwirkung: Dr.techn. Jean-Baptiste Nicolas Weibel, MSc

Wien, 4. Mai 2023

Martin Haar	Markus Vincze

Informatics

Visual Simultaneous Localization And Mapping Evaluation on a Mobile Robot Platform

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Martin Haar, BSc
Registration Number 01625753

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ. Prof. Dipl.-Ing. Dr.techn. Markus Vincze
Assistance: Dr.techn. Jean-Baptiste Nicolas Weibel, MSc

Vienna, 4th May, 2023

Martin Haar Markus Vincze

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Martin Haar, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 4. Mai 2023

Martin Haar

Acknowledgements

I would like to express my sincere gratitude to all the people who supported me with my master's thesis. I want to specially thank ...

... professor Markus Vincze who made it possible for me to write this thesis on the *Vision for Robotics* group.

... Jean-Baptiste Nicolas Weibel for his unwavering guidance, patience and encouragement. His expertise was crucial in shaping the direction of this thesis.

... my family and friends who always believed in me and and patiently supported me during this whole time.

... all my colleagues, especially Max Geiselbrechtinger and Jan Nausner who were an essential resource for me during the whole bachelor and master studies.

... the members of the faculty and the staff of the TU Wien for providing the necessary academic environment and resources for my studies.

Kurzfassung

Roboter sind unverzichtbar in unserer modernen Welt, insbesondere im industriellen Sektor. Mit dem Fortschritt in der Hardware- und Softwareentwicklung verbessern sich auch unsere Roboter, woraufhin sich eine Vielzahl von neuen Anwendungsgebieten erschließt. Ein besonderes Gebiet, das an Bedeutung gewinnt, sind Personal Human Support Robots. Damit jedoch solch automatisierte Maschinen in einem persönlichen Kontext verwendet werden können, müssen sie in der Lage sein, sich selbst zu lokalisieren und Karten ihrer Umgebung zu erstellen. *Simultaneous Localization and Mapping* (SLAM) ist eine Algorithmusklasse, die bei diesen Problemen eine entscheidende Rolle spielt. Mit SLAM können Roboter eine Vielzahl von Sensoren nutzen, um Karten zu erstellen, sich zu positionieren und dies während der Laufzeit zu optimieren. Diese Posen können in Kombination mit RGB-D Bildern verwendet werden, um 3D-Rekonstruktionen der Umgebung des Roboters zu generieren, die später bei der Anwendung von grasping Techniken hilfreich sein können. Um die geschätzte Trajektorie einer SLAM Methode zu bewerten, muss die tatsächliche Trajektorie genau bekannt sein. Aufgrund ungenauer Aktuatoren und Rauschen werden Testräume mit externen Sensoren benötigt, welche genau kalibriert werden müssen. Solche Einrichtungen sind nicht nur komplex, sondern auch sehr teuer und limitieren die Evaluationsszenarien. In dieser Arbeit wird deshalb vorgeschlagen, 3D-Objektrekonstruktionen basierend auf SLAM Posenschätzungen zu verwenden, um die Qualität von SLAM-Trajektorien zu bewerten. Im Anschluss daran wird eine Evaluierung visual SLAM-Methoden auf der Toyota HSR Mobile Robot Platform durchgeführt. Die Qualität und Robustheit von SLAM-Trajektorien werden gemessen, indem 3D-Objekte, welche mithilfe von Positionsabschätzungen aus der visual SLAM-Methode rekonstruiert wurden, mit exakten 3D-Modellen verglichen werden.

Abstract

Robots are indispensable in our modern world and particularly in the industrial sector. As the hardware and software development progresses, so do our robots which unlock a variety of new domains. One particular field that is gaining traction is personal human support robots. However for such automated machinery to be used in a personal context it needs to be capable of self-localizing and creating maps of its surroundings. *Simultaneous localization and mapping* (SLAM) is an algorithm class that plays a crucial role in all of those problems. With SLAM, robots can use a variety of sensors to create maps as well as poses and optimize those while working. Those poses in combination with RGB-D images can be used to generate 3D reconstructions of the robot's surroundings, and let robots autonomously model objects within it to support grasping techniques. Creating such models requires very accurate poses, but evaluating the estimated trajectory of a SLAM technique depends on the availability of the real trajectory to be known exactly. Due to inaccurate actuators and noise, measuring trajectories need test rooms to be set up with external sensors, which need to be well-calibrated. Such setups are not only complicated but also expensive and limit the scenarios that can be evaluated. In this thesis, we propose to use 3D object reconstruction based on SLAM pose estimations to assess the SLAM's quality. In the wake of this, an evaluation of novel visual SLAM methods on the Toyota HSR mobile robot platform is carried out. The quality, as well as the robustness of SLAM trajectories, are measured by comparing reconstructed 3D objects generated by poses from visual SLAM methods to high-quality 3D object models.

Contents

CHAPTER 1

Introduction

The word *robot* was introduced more than 100 years ago in 1920 by Karel Čapek in his play R.U.R [Ča20]. But the idea to automate tasks by the use of machines goes back even further. The first devices which could be seen as machines in the modern sense were steam engines which go back to the late 17th century. After mankind mastered the use of electricity, the number of machines rose rapidly. According to the IFR (International Federation of Robotics) [IFR23] the number of operational industrial robots has nearly tripled in the time frame between 2010 and 2020 (see Figure 1.1).

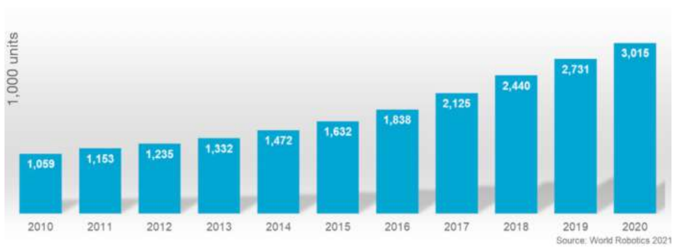


Figure 1.1: Number of operational industrial robots according to the IFR [1]

Through research and development in the fields of algorithmics, electrical engineering, automation, image processing, etc., robots also slowly transitioned from an industrial context into our homes. The annual conference on social robotics [CCF+23] deals with social robots for assisted living and healthcare, it furthermore emphasizes the increasing importance of social robotics in our daily lives. Topics from robots in educational use cases, to robotics in clinical and caring scenarios to robot acceptance and ethics are discussed there. This shows that researchers from various backgrounds are working on integrating robots into our daily live.

[1]https://ifr.org/img/office/Digitalsheet_A4_World_Robotics_2022.pdf

1.1 Motivation

Two of the key abilities a support robot must have are the capability to locate itself as well as map its surrounding environment and understand everyday objects in order to interact with them. The first of the two problems can be tackled by methods performing *Simultaneous Localization and Mapping* (SLAM). Smith et al. [SC86] introduced the first SLAM technology in 1986 and thereby created a new field of research. Since all measurements of our surroundings are littered with noise, novel SLAM algorithms employ a multitude of sensors and optimization methods to minimize the error of the estimated poses and recreated maps. The availability of cheaper and smaller cameras led to the use of cameras in all sorts of mobile robot platforms. As those sensors became ubiquitous, researchers started to incorporate the image data into the SLAM methods. So called Visuals SLAMs are now widely used, with algorithms like ORB-SLAM [CER+21], RTAB-MAP [LM19] or Kimera [RACC20] just to name a few examples.

Visual SLAMs can use monocular, stereo as well as depth cameras. While monocular cameras are the simplest and cheapest of the three, they do not give any depth information and are therefore not suited to re-create scale-accurate representations of their surroundings. For this purpose, stereo or depth cameras are needed. Such scale-accurate knowledge of a robot's trajectory can then be used to create 3D representation of its environment. Object models can be extracted from it and can then be used by algorithms to determine how to grasp and manipulate objects in that environment.

The research fields mentioned above are very active, producing ever more accurate and robust systems, but SLAM system remain quite complex. To ease the R&D in this matter Toyota has developed the Human Support Robot (HSR) [YNK+18] which is a compact mobile robot with a variety of different sensors and actuators. The HSR is used as primary research platform on the *Vision for Robotics* group, it is used for testing software and algorithms of a variety of tasks.

1.2 Challenge

As precise positioning is required for many of these tasks, SLAM is a crucial component. Hence, it is important to have the capability to evaluate SLAM methods on the HSR and to determine the limits of its positioning accuracy. To guarantee the behavior of the robot, such an evaluation should also be possible on the deployment site.

The evaluation of SLAM trajectories is a challenging task that often requires the use of external sensors. To track the trajectory of a robot, a room-level sensor array must be employed. While this approach provides a precise ground truth, it is also complex and expensive. There are datasets available which provide such ground truth such as the EuRoC dataset [BNG+16] or the TUM RGB-D dataset [SEE+12]. The methods for obtaining ground truth are described in their respective papers and highlight the complexity of the required setups. This type of evaluation cannot be done once deployed because of it. Another common method to compare the quality of a SLAM-generated trajectory is to create a loop and compare the first and last pose. However, this approach leaves significant uncertainty and should only be considered a preliminary quality estimate.

1.3 Contribution

To address these issues, this thesis proposes using the pose estimations calculated by SLAM to reconstruct 3D objects. Comparing the reconstructed 3D objects to their corresponding high-quality object models provides a more significant measure of the trajectory's quality than simply comparing the first and last pose. Fortunately, numerous high-quality 3D models of various objects are already available, offering a reliable ground truth that eliminates the need for a complicated sensor array to generate the entire robot trajectory.

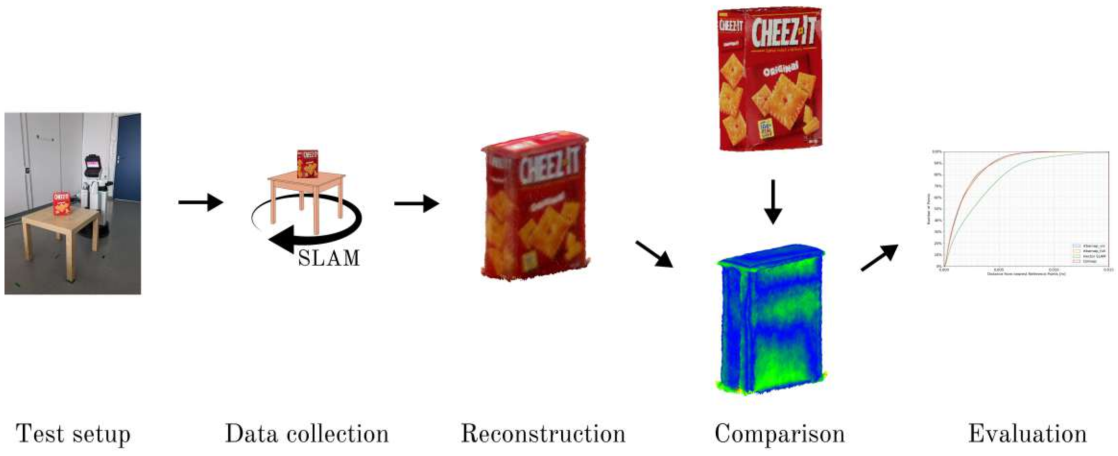


Figure 1.2: Method to evaluate the SLAM trajectory using 3D reconstruction

Figure 1.2 illustrates the methodology utilized to address the evaluation problem presented above. In the leftmost block of the figure, the HSR is shown in the test environment. It is then manually guided around the desk to collect sensor data from the RGB-D and LiDAR sensors. This sensor data is subsequently used to generate robot poses via SLAM methods. Using the poses and previously acquired image data, the 3D object reconstruction is performed. The reconstructed object is then registered with the corresponding ground truth object, and a point-to-point comparison is conducted for each test case. The accuracy and robustness of the methods are evaluated accordingly, leading to the main goal of this thesis: Visual Simultaneous Localization And Mapping Evaluation on a Mobile Robot Platform.

The actual contribution can be split into three parts:

1. Evaluation of SLAM algorithms through the evaluation of 3D object reconstruction with real data on the HSR.
2. Evaluation of SLAM algorithm's robustness when confronted with different object setups.
3. Are state-of-the-art Visual SLAM methods applicable for autonomous 3D object modeling (e.g. object grasping)?

The robustness of such algorithms is of particular importance when used in an everyday setting. Lighting changes, varying positions and more or less feature-rich backgrounds are situations that occur constantly when a robot is deployed. It is of major significance that the robot is still able to extract meaningful pose estimations in such situations. To test the robustness of given SLAM algorithms the experiments in the evaluation process should be taken with different camera positions, different objects, etc.

1.4 Outline

The theoretical background needed for this thesis can be found in Chapter 2, which covers the topics of basic image processing, SLAM, 3D object reconstruction and ROS. Chapter 3 covers the methods used to conduct the necessary experiments as well as the implementation in detail. More details about said experiments and the results are presented in Chapter 4 and the short summary of the work as well as the conclusion can be found in Chapter 5.

CHAPTER 2

Background

This chapter gives a theoretical background for this thesis and covers everything needed from technical concepts to previous work.

Section 2.1 gives an overview about image processing. Since this work deals mainly with visual SLAM methods, some theory regarding digital images, interest points, feature matching, and so on is needed. The principles of SLAM methods with a special focus on visual SLAM are explained in Section 2.2. Another key part as already stated in the thesis title is 3D object reconstruction which is covered in Section 2.3. Finally Section 2.4 gives a short insight into the robotic platform used in this thesis.

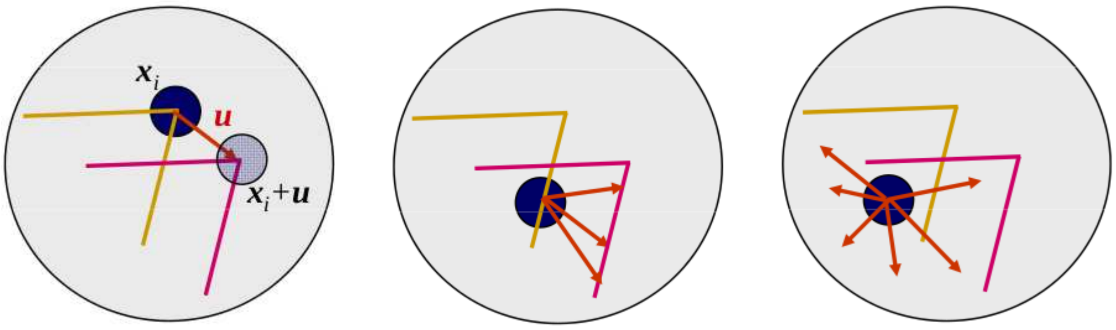
2.1 Image Processing

When working with visual SLAM or any other imaging software, image processing is needed. When humans or animals see two pictures of a scene with a temporal distance it is intuitive to them to detect similarities and recognize the temporal shift. Maybe it is even possible to make assumptions about movement, velocity, time difference and so on. Since machines and robots do not have the capabilities for such intuitive assessments, mathematical methods are necessary. Through the concept of image similarities, position and movement can be estimated.

When talking about image features, the notion of global and local features needs to be distinguished [AH16]. A global feature is represented by a single vector which gives information about the whole image, e.g. a color histogram. This can be used to distinguish whole images from each other or draw conclusions about the image content by the use of one feature. Global features, however, are not helpful when trying to find objects in images or perform odometry, such tasks require local features. Since visual odometry is at the basis of visual SLAM methods, the theory behind local feature detection, feature description, and feature matching is described in the following sections.

2.1.1 Feature Detectors

In order to align, stitch, or compare two images it is necessary to find significant points in them which can be compared. Significant areas are mostly edges, corners, or contours. Szelinsky describes the three classes of basic keypoints and how they can be found [Sze22].



(a) Corner (b) Edge (c) Textureless region

Figure 2.1: Aperture problems for different image patches from [Sze22]

Figure 2.1 shows three different scenarios of image patches. The first one in Subfigure 2.1a shows a corner, which means that the image has a strong contrast change in both x and y directions. This is also reflected in the gradient of the image, and therefore a corner can be easily located. The second scenario shown in Subfigure 2.1b shows an edge which has a gradient change only in one direction, which makes it only possible to localize the image patch along a line. The last image in Subfigure 2.1c shows a featureless plane. There are no gradient changes in the whole patch, therefore it is not possible to localize the point in another picture.

Harris Detector

A particularly famous feature detector which is also quite foundational is the Harris detector [HS+88]. Since it is such a widespread method and also captures the essence of feature detection in a good way, it is described in more detail here. A good and comprehensive derivation of the Harris detector can be found in Szeliski et al. [Sze22]

To find good corner or edge features like in Figure 2.1 the auto-correlation of the image needs to be performed:

$$E_{AC}(\Delta \boldsymbol{u}) = \sum_i w(\boldsymbol{x}_i)[I(\boldsymbol{x}_i + \Delta \boldsymbol{u}) - I(\boldsymbol{x}_i)]^2 \tag{2.1}$$

Using the Taylor expansion $I(\boldsymbol{x}_i+\Delta\boldsymbol{u}) \approx I(\boldsymbol{x}_i)+\nabla I(\boldsymbol{x}_i)\Delta\boldsymbol{u}$ where $\nabla I(\boldsymbol{x_i}) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)(\boldsymbol{x}_i)$ is the image gradient at $\boldsymbol{x}_i$. Thus the auto-correlation can be approximated by:

$$\approx \sum_i w(\boldsymbol{x}_i)[I(\boldsymbol{x}_i) + \nabla I(\boldsymbol{x}_i) \cdot \Delta\boldsymbol{u} - I(\boldsymbol{x}_i)]^2 \tag{2.2}$$

$$= \sum_i w(\boldsymbol{x}_i)[\nabla I(\boldsymbol{x}_i) \cdot \Delta\boldsymbol{u}]^2 \tag{2.3}$$

$$= \Delta\boldsymbol{u}^T A \Delta\boldsymbol{u} \tag{2.4}$$

where the auto-correlation matrix $A = w * \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$

Stable features can be found by analyzing the two Eigenvalues δ_0 and δ_1 of the Autocorrelation matrix.

SIFT Detector

Another established feature detector is the scale invariant feature transform (SIFT) [Low04]. As the name suggests, it gives scale invariance through a pyramid of scaled images, as shown in Figure 2.2. The scale space images on the left are generated by repeatedly convolving the image with a Gaussian function. Adjacent images are then subtracted from each other to get the difference-of-Gaussians (DoG), for the next octave the image is down-sampled.

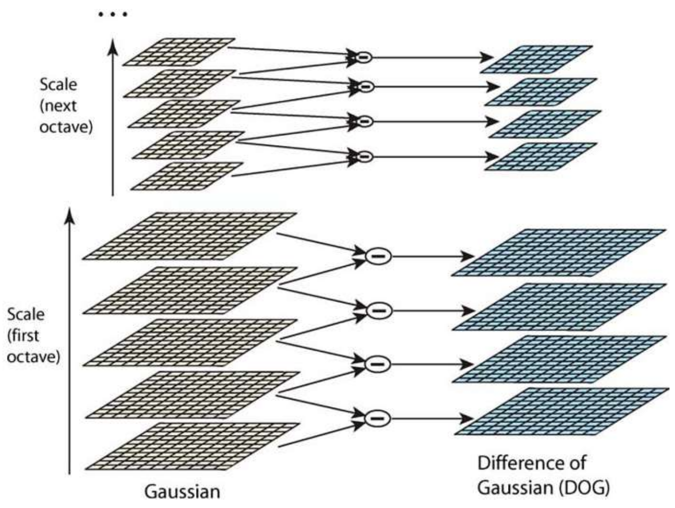


Figure 2.2: Image scale pyramid to produce DoG images [Low04]

According to Lindeberg [Lin94] the Gaussian kernel is, under reasonable assumptions, the only one which can be used for scale-space comparisons. From this follows that the scale-space of an image can be defined as a convolution of a variable scale Gaussian with the input image:

$$L(x, y, \gamma) = G(x, y, \gamma) * I(x, y) \tag{2.5}$$

where $G(x, y, \gamma)$ is the Gaussian and $I(x, y)$ the input image. With this function $L(x, y, \gamma)$ the DoG can simply be calculated by image subtraction:

$$\begin{aligned} D(x, y, \gamma) &= (G(x, y, k\gamma) - G(x, y, \gamma)) * I(x, y) && (2.6) \\ &= L(x, y, k\gamma) - L(x, y, \gamma) && (2.7) \end{aligned}$$

Lowe shows in [Low04] a derivation why the DoG method is a good approximation of the Laplacian of Gaussian as proposed by [Lin94].

To find local extrema each keypoint has to give a strong response not only in image space, but also in scale space. This ensures that if the image gets scaled, it will still be possible to find the corresponding SIFT keypoints. To detect local extrema each sample point is compared to its eight immediate neighbors and to its nine neighbors in the DoG image above and below. Should the keypoint be larger or smaller than all of the other points in the comparison, a local extrema has been found.

Some Further Detectors

There are a lot of different approaches to find stable keypoints, a few well known feature detectors are shortly outlined in following paragraphs. There is a lot of activity in this research field, and therefore a lot of different methods and improvements are presented regularly.

Another widely used feature detector is used in SURF [BTVG06]. It uses the *Fast-Hessian Detector* to find stable keypoints. Bay et al. use here a similar concept as Lowe used in SIFT, but they approximate the Gaussian with box filters, which speeds up the localization of keypoints by more than a factor of 3 when compared to DoG.

To speed-up the process of keypoint detection even more Rosten et al. introduce the FAST descriptor [RD06]. The abbreviation FAST stands here for *Features from Accelerated Segment Test*. The FAST method is to take image patches only and check, if there is a certain structure, in particular specific pixels in each patch are checked for intensity contrast. This method, however, introduces several problems, which are solved by using machine learning on top of this method.

Rublee et al. developed the ORB feature detector [RRKB11] which has become popular through the usage in ORB-SLAM [MAMT15] and its two successors ORB-SLAM2 [MAT17] and ORB-SLAM3 [CER$^+$21]. The ORB feature detector uses FAST for keypoint finding and filters the output with a Harris corner measure. The number of keypoints is set beforehand to a value N, with the use of FAST the number of N keypoints found in a frame is usually surpassed. To reduce the number of keypoints the Harris filter is applied to the detected keypoints, which are ordered accordingly and the N best are kept. Furthermore, a scale pyramid similar to SIFT is generated, and the filtered FAST features are extracted at every pyramid level to get better scale invariance. This detection method is called *oFAST: FAST Keypoint Orientation*.

2.1.2 Feature Descriptors

After a stable keypoint has been found, it must be possible to match it with the same keypoints in different images. For this reason each keypoint also comes with a feature descriptor. A feature descriptor is a combination of the pixels in the image patch around the keypoint which takes into account the orientation, the scale, or affine transformations. This reassembling of the image patches can be done in a great variety of different ways. In the following paragraphs a few of those techniques are briefly outlined.

To get rotation invariance the simplest approach would be be to create the average of gradient directions of the image patch. This approach, however, has the problem that due to its simplicity different looking patches get mapped to the same keypoint. Another method would be to establish a dominant orientation for each keypoint.

SIFT Descriptor

The SIFT feature extractor [Low04] not only has a very famous feature detector, but also a famous feature descriptor which has been used extensively in all sort of fields. To not only counteract the influence through rotation, but also the effects of affine distortion Lowe introduced in the SIFT feature detector a histogram of edge orientations.

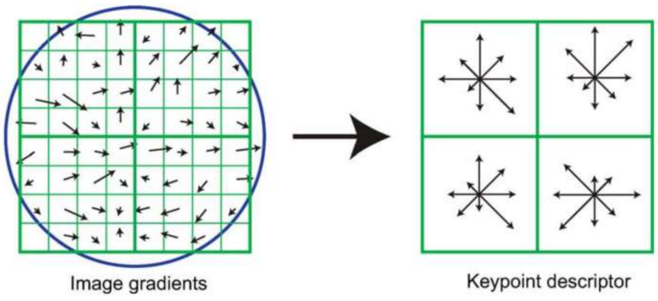


Figure 2.3: Keypoint descriptors calculated from image gradients [Low04]

Figure 2.3 shows the basic concept of the histogram approach. Affine distortions can be introduced through a variety of events, such as changes in illumination or viewpoint position. Especially position changes can introduce shifts in the gradient position, therefore a slight positional shift should still yield the same keypoint. Lowe introduces a gradient field around the keypoint which can be seen on the left in Figure 2.3. In the Figure this 8×8 gradient field is then condensed into 2×2 bins which can be seen on the right. Furthermore, all gradients are smoothed by a two-dimensional Gaussian function in order to weight them depending on their distance to the keypoint. The histogram has the great advantage that gradients which shift their position slightly still contribute to the same bin and therefore do not change the feature descriptor. If, however, a gradient spills over to a neighboring bin, the transition is smoothed by further weighting the gradients by a factor $1 - d$ where d represents the distance to the bins center. A remaining problem are non-linear illumination changes, these can change some gradients massively in their magnitude, but not their orientation. To get a hold of this problem the gradients are normalized, clipped at 0.2 and then re-normalized. This thresholding reduces the influence of gradient magnitudes and emphasizes the weight of the orientation. The value 0.2 was determined experimentally by Lowe [Low04]. Moreover, the optimal result was accomplished by using 4×4 bins consisting of 8×8 sample regions resulting in 128 gradients used for one feature descriptor.

Some Further Descriptors

The research field on feature point descriptors is still a very active field. In the following paragraphs a few interesting feature descriptors are outlined.

The feature descriptor of SURF [BTVG06] is composed of two parts. Orientation invariance is ensured through the calculation of Haar-wavelets of a circular region around the point of interest. They are then further processed to get a final orientation vector. The feature vectors are then computed from an aligned rectangle around the feature point.

BRIEF [CLSF10], which stands for *Binary Robust Independent Elementary Features*, uses pixel comparisons to find unique keypoints. To achieve this an image patch of size $S \times S$ is taken and a number of pixels inside the patch are compared to each other. If the intensity of the first pixel is larger than pixel two, the result is 1 and 0 vice versa. This comparisons are saved in a binary vector and used as feature descriptors. For this method to work, it is important to smooth the image patch before comparison with some kernel, the choice of this kernel gives some implementation margin.

The ORB feature extractor [RRKB11] uses a method called *rBRIEF: Rotation-Aware BRIEF*. This modified version of BRIEF, where Rublee et al. used the PASCAL data set [EZW$^+$06] to find the best pixel contestants for binary matches which are independent of rotations. The combination of oFAST and rBRIEF outperforms SIFT and SURF as well as BRIEF.

Gao et al. [GZ21] summed up four characteristics each feature should have:

1. **Repeatability:** It should be possible to find each feature also in different images.

2. **Distinctiveness:** The features are defined through different expressions.

3. **Efficiency:** The computational effort for each feature should be feasible and the number of pixels much larger than the number of features.

4. **Locality:** Each feature should refer to a comparatively small part of the image.

2.1.3 Feature Matching

Once keypoints are found and the respective descriptors calculated, they can be used for a variety of applications. Depending on those applications the right strategy for matching the features must be selected [Sze22].

One of the most common ways to match features is to compare the Euclidean distance of descriptors in feature space to each other. The distance can then be used for ranking the found matches. A threshold is then defined to filter possible matches. If the chosen value of this threshold is too large, no matches are found, but if it is chosen too small, too many possible matches are taken into consideration. Therefore, the selection of this threshold value is very important. To pick this value more generally a normalization or in this context *whitening* of the axis in feature space can greatly improve the results.

To measure the quality of a matching, some performance metrics are needed. The important values to calculate such metric are the rates of true positives (tp), false positives (fp), true negatives (tn) and false negatives (fn). Theses values classify how many feature descriptors are correctly matched, how many are wrongly matched and vice versa. Figure 2.4 summarizes the concept of feature matching metrics.

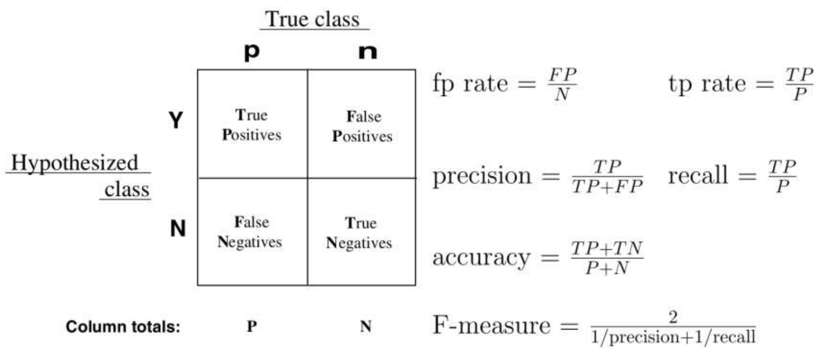


Figure 2.4: Feature matching metrics [Faw06]

Based on those metrics a receiver operating characteristic (ROC) can be drawn, it shows the *fp rate* on the X axis and the *tp rate* on the Y axis. As the threshold is varied, a curve is generated and can be displayed in the form of a ROC. Tom Fawcett explains in his paper [Faw06] how to read and analyze ROCs correctly. Figure 2.5 shows an example of such a graph.

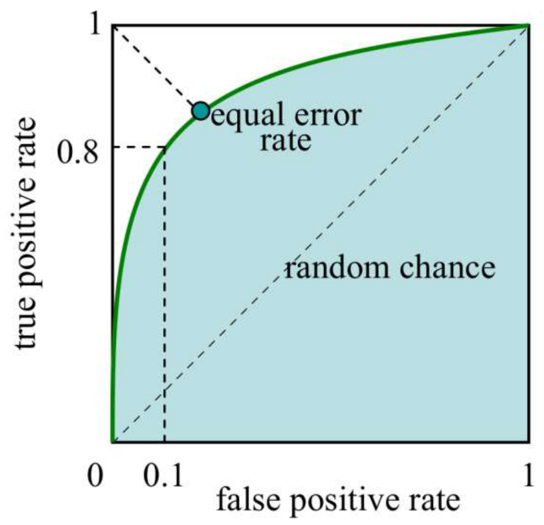


Figure 2.5: Example of a ROC [Sze22]

The performance of the threshold is measured by the area under the curve, i.e. the closer the curve is to the upper left corner of the diagram, the better the performance. The diagonal line ($x = y$) represents the case of random guessing, which colloquially said means a fifty-fifty chance to get a tp or fp.

Another important topic after describing how to measure the quality of a feature match is on how to perform it efficiently. In the most basic case each descriptor of one image is compared to each descriptor in the other image, this, however, can turn out to be quite computationally expensive. Modern GPUs can help to speed up the matching process, but in general this method is not feasible. Therefore, a lot of different strategies have been introduced which utilize indexing structures such as kd-trees [LWH10] or FLANN tree structures [ML14] to accelerate the feature searching process. Furthermore, it can be useful to only consider a subset of the descriptors when for example performing an object search. Szeliski [Sze22] describes several methods on efficient feature matching and provides a comprehensive list of examples.

2.2 Simultaneous Localization and Mapping (SLAM)

The abbreviation SLAM stands for Simultaneous Localization and Mapping and is an essential technology for all sorts of autonomous robot systems in unknown environments. Like the name suggests, it is used by robots to create a map of their surroundings and simultaneously place themselves on this map. The first SLAM technology was introduced in 1986 by Smith et al. [SC86]. They already suggested that the information gathered by sensors is inaccurate and not complete. Therefore SLAM algorithms use a multitude of sensors and filters to reduce said errors and optimize the map, the position as well as the relation of the seen objects to each other.

Since then a multitude of different SLAM algorithms have been developed. These algorithms utilize all sorts of available sensors like LiDAR, RaDAR, odometry, cameras, IMUs, and so on. The goal of this thesis is to benchmark the performance of SLAM methods for object reconstruction with visual SLAM methods being of particular interest.

A lot of the recent SLAM implementations are summarized in Barros et al. [MBMM+22], Zhang et al. [ZZT21] and Servieres et al. [SRD+21].

Both of the tasks, mapping and positioning, are done incrementally and rely on the precision of each other. Therefore, the location and map estimation are in contrast with each other, similar to the *chicken and egg* problem [LCW+12]. The convergence of these contradicting problems has been proven by Csorba [Cso97], and more novel works regarding the topic are more focused on optimizing the computational efficiency.

The basic functionality of a visual SLAM framework has been explained by Gao et al. [GZ21]. They propose a simple SLAM pipeline shown in Figure 2.6

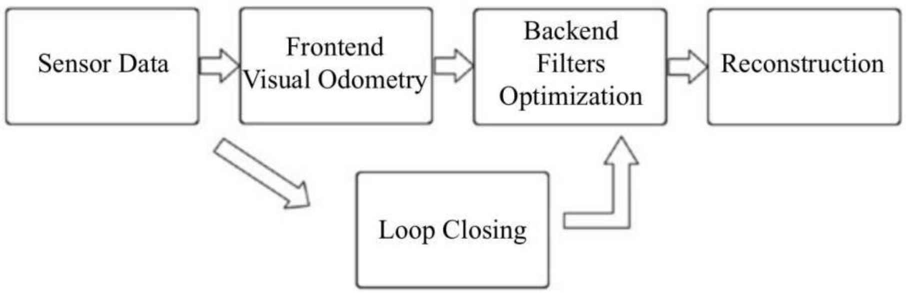


Figure 2.6: Classic visual SLAM framework from Gao et al. [GZ21]

The pipeline consists of four blocks which roughly outline the parts of the VSLAM system.

1. **Sensor Data:** The first blocks reflect the acquisition of sensor data. This mainly refers to visual data like mono, stereo and RGB-D images, but it also includes additional data like IMU, encoders, or LiDAR.
2. **Visual Odometry:** In this block the camera pose can be roughly estimated through the comparison of adjacent images, and a coarse sketch of the map can be generated.
3. **Filters/Optimization**: The third stage receives the camera pose at different time stamps and can use various optimization methods to calculate an optimized trajectory.

4. **Loop Closing:** In order to reduce drift over time some SLAM algorithms check, if a location has already been visited and hence a loop has been created. This information can then be used to further optimize the map quality and with that also the positioning.

5. **Reconstruction:** The actual creation of the map based on the before estimated and optimized camera trajectory.

The following three chapters will take a deeper look at blocks 2, 3 and 4 of the VSLAM pipeline.

2.2.1 Visual Odometry (VO)

One of the first papers which also gave the notion of VO its name was written by Nister et al. [NNB04], and since then VO is an integrative part of the R&D in the robotic sector. The name *visual odometry* is derived from *wheel odometry* due the their similar role of incrementally estimating a robots movement [SF11].

Gao et al. [GZ21] classify VO into two common methods:

1. Feature/Indirect methods

2. Direct methods

Zhang [ZZT21] also adds the class *Hybrid Methods*, but this is not dealt with specifically here.

Feature Method

Like the name already suggests when applying the feature method each image is scanned for keypoints and descriptors. How a basic keypoint with an affiliated descriptor can be calculated is explained in Section 2.1, afterwards a feature matching is applied to two successive images.

Depending on which image types are matched: 2D-2D, 2D-3D or 3D-3D, different methods are used to estimate the movement from one frame to the next [GZ21].

1. **2D-2D Epipolar Geometry:** The two cameras and the object of interest form a triangle which is called the epipolar plane. The intersection of the epipolar plane with the image plane forms the epipolar line, and the point of interest in the second image lies on this epipolar line. Along the line of interest the given point can be searched by using feature matching like already explained in Section 2.1.

2. **2D-3D Perspective-n-Point (PnP):** To find the pose of a 2D camera in contrast to a 3D scene, a set of 3D points which are non-colinear is chosen from the 3D scene.

Provided the camera intrinsics are known, those 3D point can be projected to the 2D plane and thereby get the camera extrinsics. One way to efficiently calculate the camera pose is to apply an iterative method which aims to minimize the projection error, which is the distance of the observed 2D image and the 3D $\rightarrow$ 2D projection.

3. **3D-3D Iterative Closest Point (ICP):** The 2D-2D as well as the 2D-3D method rely on feature detection and feature matching to find the camera pose. ICP on the other hand uses the whole pointcloud to find the optimal pose. A common way way to get the camera pose from one point cloud to another pointcloud is to iteratively minimize the distance between them. The first step is to find correspondences between the two pointclouds and register them accordingly. Secondly the two pointclouds can be aligned using the information gained by the registration. The transformation acquired from the alignment can then be applied to the camera pose of the first pointcloud to get the absolute pose of the second pointcloud. This method can be executed iteratively, until the pose converges.

Direct Method

As already explained in Section 2.1 the calculation of feature descriptors and the subsequent feature matching used in the feature method can be extremely resource-intensive. Furthermore, the feature method relies heavily on the gradient of the image, henceforth if there are no edges or corners, there are no good features. The direct method is intended to provide a remedy to this problem by taking advantage of a notion called *optical flow* [GZ21].

Optical flow is used to track pixels or keypoints in subsequent images. From this follows that it is still useful to calculate keypoints when using a direct method, but the calculation of feature descriptors can be left out.

The whole concept of optical flow relies on the assumption that the pixel intensities are constant for short duration. Mathematical concepts can be reviewed in Gao et al. [GZ21] or Beauchemin et al. [BB95]. As an example the derivation of the Lucas-Kanade optical flow [LK81], a representative of the sparse methods and important basis, is summarized here:

$$\boldsymbol{I}(x, y, t) = \boldsymbol{I}(x + dx, y + dy, t + dt) \tag{2.8}$$

Where $I(x, y, t)$ is the image regarded as a function of the pixel coordinates x and y at the time t, if the pixels are shifted in time and space by dx, dy and dt, the intensities stay the same. This can further be approximated in a first order tailor series:

$$\boldsymbol{I}(x + dx, y + dy, t + dt) \approx \boldsymbol{I}(x, y, t) + \frac{\partial \boldsymbol{I}}{\partial x}dx + \frac{\partial \boldsymbol{I}}{\partial y}dy + \frac{\partial \boldsymbol{I}}{\partial t}dt \tag{2.9}$$

Since it was assumed in the first place that the pixel intensities are constant and therefore a pixel with a small change in space and time does not change its intensity, the derivation part of the tailor series becomes zero.

$$\frac{\partial \boldsymbol{I}}{\partial x}dx + \frac{\partial \boldsymbol{I}}{\partial y}dy + \frac{\partial \boldsymbol{I}}{\partial t}dt = 0 \tag{2.10}$$

By some re-arranging this can be written as:

$$\frac{\partial \boldsymbol{I}}{\partial x}\frac{dx}{dt} + \frac{\partial \boldsymbol{I}}{\partial y}\frac{dy}{dt} = -\frac{\partial \boldsymbol{I}}{\partial t} \tag{2.11}$$

In this equation the $\frac{dx}{dt}$ and $\frac{dy}{dt}$ denote the pixel speed in 2D space which can be denoted as u and v. Written in matrix form this yields:

$$\begin{bmatrix} \boldsymbol{I}_x & \boldsymbol{I}_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\boldsymbol{I}_t \tag{2.12}$$

This equation contains two unknowns, namely u and v which can of course not be solved in the traditional matter. To solve this under determined linear equation it has to be assumed that all pixels in immediate vicinity move with the same speed in the same direction. Thereby several pixels in a $w \times w$ area can be used to create a solvable linear system of equations.

$$\boldsymbol{A} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{I}_x & \boldsymbol{I}_y \end{bmatrix}_1 \\ \begin{bmatrix} \boldsymbol{I}_x & \boldsymbol{I}_y \end{bmatrix}_2 \\ \vdots \\ \begin{bmatrix} \boldsymbol{I}_x & \boldsymbol{I}_y \end{bmatrix}_w \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} \boldsymbol{I}_{t1} \\ \boldsymbol{I}_{t2} \\ \vdots \\ \boldsymbol{I}_{tw} \end{bmatrix} \tag{2.13}$$

$$\boldsymbol{A} \begin{bmatrix} u \\ v \end{bmatrix} = -\boldsymbol{b} \tag{2.14}$$

This is now an over-determined linear system of equation and can be solved by applying least-squares.

$$\begin{bmatrix} u \\ v \end{bmatrix} = -(A^T A)^{-1} A^T b \tag{2.15}$$

Optical flow methods are used to track certain feature points from one image view to another which can for example be used for object tracking.

In visual odometry however the interesting part is the camera pose, and this can be achieved by minimizing the *photometric error* over the whole images. Consider two consecutive images $\boldsymbol{I}_1$ with point $\boldsymbol{p}1$ at time t and the second image $\boldsymbol{I}_2$ at $t + dt$ with point $\boldsymbol{p}_2$ which is shifted by some translation and rotation $\boldsymbol{T}$. To find the values of this transformation the photometric error needs to be minimized. Since the pixel intensity assumption from the optical flow is still force, the error can be defined by a simple subtracting:

$$e = \boldsymbol{I}_1(\boldsymbol{p}_1) - \boldsymbol{I}_2(\boldsymbol{p}_2) \tag{2.16}$$

In order to find dx and dy the this error needs to be minimized. For this an objective function J is defined which corresponds to the sum of the squared differences between the pixel intensities of the first and of the second images over T.

$$\min_{\boldsymbol{T}} J(\boldsymbol{T}) = \sum_{i=1}^{N} e_i^T e_i \quad \text{with} \quad e_i = \boldsymbol{I}_1(p_{1,i}) - \boldsymbol{I}_2(p_{2,i}) \tag{2.17}$$

When solving this optimization problem the camera pose of second image in contrast to the first one has been found. Since the whole notion of optical flow and the direct method is based on the pixel intensity assumption, large changes the intensities (e.g. sudden lighting changes) can severely worsen the pose estimation. This derivation was taken from [GZ21] where more details can also be found. There are also explanations on how to solve this minimization and how to implement and use such directed methods.

2.2.2 Backend Optimization

Every sensor and actuator used in the real world is prone to noise and exposed to a continuously changing environment. Temperature changes, lighting fluctuations, radiation from surrounding devices, and so on can affect the sensor output and influence the overall behavior of hardware, albeit very little. If ignored, this noise causes an error in the pose estimation which accumulates rapidly and can, in order of minutes, render the SLAM output completely unusable.

Through the notion of optimization this noise is modelled mathematically as a stochastic state estimation problem. The noise is treated as a random variable which certain distributions and the actual pose can then be approximated with these uncertainties in mind.

The first SLAM methods introduced over 35 years ago dealt almost exclusively with this problem. Just to name a an example, Smith et al. [SC86], who are considered the founder of SLAM, called their paper *On the Representation and Estimation of Spatial Uncertainty.*

In order to apply such state estimation methods the general problem of SLAM needs to be formulated in a mathematical form. Gao et al. [GZ21] does this in a simplistic and generalized way. To describe the SLAM problem some pose variables and map variables are needed. In this case x_k with $k = 1 \dots N$ describe the pose where k denotes the timestamp and the map is described by a number of landmarks y_j with $j = 1 \dots M$ where j denotes the landmark number. Furthermore the observations, which are also prone to noise, need to be modelled, which is reflected in $\boldsymbol{z}_{k,j}$. The whole problem can now be written in the form of two equations:

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{2.18}$$
$$\boldsymbol{z}_{k,j} = h(\boldsymbol{y}_j, \boldsymbol{x}_k) + \boldsymbol{v}_{k,j} \tag{2.19}$$

Equation 2.18 is called the *motion equation*. In this case the function $f(\cdot)$ describes a general motion of the robot, where $\boldsymbol{x}_{k-1}$ is the pose at the last time stamp and $\boldsymbol{u}_k$ is the input vector. The noise is introduced with $\boldsymbol{w}_k$ which integrates all uncertainties of the motion part in one random variable.

Equation 2.19 is called the *observation equation*. The robot sees from pose $\boldsymbol{x}_k$ a subset of landmarks $\boldsymbol{y}_j$ with the observation noise $\boldsymbol{v}_{k,j}$. The function $h(\cdot)$ represents the generalized observation of the robot.

Since the poses as well as the landmarks are now random variables some assumptions about their distribution has to be made. In this case they are both regarded as Gaussian random variables and are therefore completely determined by their mean and covariance. The mean of the Gaussian can therefore be regarded as estimated value and the variance as its uncertainty. If all the past and future values of $\boldsymbol{x}$ and $\boldsymbol{z}$ are regarded the whole estimation becomes non-linear.

To simplify further equations, some simplifications of the underlying formulas 2.19 and 2.19 are done.

$$\boldsymbol{x}_k \mathrel{\widehat{=}} \{\boldsymbol{x}_k, \boldsymbol{y}_1, \dots \boldsymbol{y}_M\} \tag{2.20}$$
$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, u_k) + \boldsymbol{w}_k \tag{2.21}$$
$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k \tag{2.22}$$

This simplification puts all the landmarks $\boldsymbol{y}_j$ into the $\boldsymbol{x}_k$ vector which makes the observation $\boldsymbol{z}_k$ only depended on $\boldsymbol{x}_k$. With the Gaussian random variables and the simplifications from Equations 2.20 to 2.22, the motion and observation equation can now be written as a likelihood:

$$P(\boldsymbol{x}_k|\boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k}) \propto \underbrace{P(\boldsymbol{z}_k|\boldsymbol{x}_k)}_{\text{likelihood}} \underbrace{P(\boldsymbol{x}_k|\boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k-1})}_{\text{prior}} \tag{2.23}$$

Most novel SLAM algorithms use non-linear estimation as optimization method, but also linear optimization is widely used as they lay the foundation for this kind of optimization.

To get from the non-linear to the linear estimation problem the Markov assumption has to be applied. From this follows that in the linear case the motion estimation and the observation are only depended on the state $k-1$, everything earlier can be omitted, since it has no longer any effect on the next state k. This simplification yields the famous Kalman filter as well as the extended Kalman filter (EKF).

$$\boldsymbol{x}_k = \boldsymbol{A}_k \boldsymbol{x}_{k-1} + \boldsymbol{u}_k + \boldsymbol{w}_k \tag{2.24}$$
$$\boldsymbol{z}_k = \boldsymbol{C}_k \boldsymbol{x}_k + \boldsymbol{v}_k \tag{2.25}$$

Although the Kalman filter is a widely used tool and gives a good foundation for optimization in the context of SLAM, there are of course many other methods.

One method which is used a lot in 2D-laser SLAMs are particle filters, in particular Rao-Blackwellised particle filters (RBPF) [Mur99]. Particle filtering introduces a mechanism of considering several possible solutions of the current position simultaneously (the particles). Each particle is sampled from the proposal distribution, and the next state is predicted for each particle. Then the weights are updated based on how good they explain the sensor measurement. The best particles are used to estimate the robot's trajectory. Rao-Blackwellization means in this context that the joint probability of the robot's trajectory and the map are factored and, only the current position needs to be estimated. The map is then updated based on the full trajectory of the robot.

Bundle adjustment is another method that is used to refine the estimates of camera poses and landmark positions obtained from image data. It is a technique which adjusts the camera poses as well as the landmark poses in order to minimize the error between the estimated 3D points and their 2D projection [GZ21]. This is done by minimizing the error of the pixel position of a landmark and the 2D projection of the 3D estimate of given landmark:

$$\boldsymbol{e} = \boldsymbol{z} - h(\boldsymbol{T}, \boldsymbol{p}) \tag{2.26}$$

In this equation $\boldsymbol{e}$ is the error to be minimized, $\boldsymbol{z}$ is the observed 2D point from the camera, $h(\cdot)$ is the 3D $\rightarrow$ 2D projection, $\boldsymbol{T}$ is the estimated camera pose and $\boldsymbol{p}$ refers to the corresponding estimated 3D landmark pose.

Pose graph optimization can be viewed as a sort of derivation of bundle adjustment. The main idea is that the landmark poses take a lot of time to optimize, although after several iterations most of them converge to a nearly constant value. This leads to a lot of computational load which doesn't really improve the overall pose and map information. With a pose graph optimization each robot pose is represented by a node in the graph and edges between those are the constraints which stem from observations. Of course the position of each landmark is initially optimized, but after this the landmark poses are no longer updated and considered constant [GZ21].

2.2.3 Loop Closure

In Section 2.2.1 and 2.2.2 the notions of initial pose estimation as well as optimization with regard to uncertainties due to noise are introduced. However, both of those methods only work on a few adjacent frames, and therefore, each error that still persists after the optimization stage is accumulated. This leads to a consistent drift of the estimated poses away from the ground truth, and there is no way to prevent this without looking further into the past.

Loop closure is used to detect previous visited positions in the robot's trajectory. This additional information can then be used to correct drift error which has accumulated since the loop beginning.

Gao et al. [GZ21] gives a good overview on why loop detection is needed and how it works. The first question is how can loops be detected in the first place. One obvious solution would be to use feature detection and feature matching like explained in Section 2.1 for every frame. This would require to extract the features from every image like for example used in the feature method of VO and then compare it to every past frame. Of course this approach has a complexity of $\mathcal{O}(n^2)$ which becomes rapidly computationally unfeasible. Another simple approach according to [GZ21] would be to compare the current frame to a fixed number of random frames. Although the computational burden stays in check the efficiency in loop detection decreases over time which is also not desirable.

Since those basic and naive approaches are impracticable, some more structured notions are needed. There are two big classes of loop closing approaches out there:

1. Odometry based
2. Appearance based

The odometry based methods rely on the pose estimation of the robot. There is already a pose estimation from the VO and the optimization part of the SLAM, therefore this positional information can be used to estimate, if a loop is forthcoming or not. Due to the error of the trajectory accumulated over time this approach contradicts itself.

The appearance-based method is more reliable in this regard and is therefore the more prominent in novel SLAM algorithms. With this method visual features in the frames are used to find potential loops. As already mentioned, simple feature matching as explained in Section 2.1.3 is too expensive. Therefore, a faster compare mechanism needs to be introduced.

Bag of Words (BoW)

Most of the loop closing methods use some form of BoW to find candidates for a potential loop. The exact details on how the word dictionary is composed and which words are used and how the comparisons are optimized vary, but the overall concept stays the same.

The BoW method is used to represent the answer to the question *What is in that picture* in a mathematical form which is fast to generate and, more importantly, fast to compare. The word *Word* is in this context some form of image feature and the word *Bag* is a synonym for *unordered list*. In the BoW sense each frame consists of a number of words which is just a synonym for high level feature. To make the concept more understandable let's use animals as high level features (in reality the features are of course more abstract). Lets assume that our dictionary consists of *cat*, *dog*, *bird* and *horse*. An image is now scanned using the notion of feature detection if there are any of the before defined words in the image. The result of this feature detection is then packed into a histogram which can be represented by a 1-dimensional vector like for example:

$$2 \times cat + 0 \times dog + 1 \times bird + 1 \times horse \mathrel{\widehat{=}} \begin{bmatrix} 2 & 0 & 1 & 1 \end{bmatrix} \tag{2.27}$$

The fictional frame in the above example contains two cats, zero dogs, one bird and one horse which is then saved in a vector. If there is now an image with the same or a similar histogram, a possible loop is detected and further steps can be taken. The big advantage of this 1-dimensional vector is that the exact position of the word does not matter and also that changes in illumination and contrast do not affect the result too much.

Precision

Since the two images which are taken into account for loop closing are never exactly the same, there is always some remaining uncertainty whether there is actually a loop or a false positive. The notion of TP, FP, TN and FN was already described in Section 2.1.3 and can be seen in Figure 2.4.

For the notion of loop closing a very high accuracy is needed, while a bad recall can be forgiven. This comes from the fact that a loop which is not detected neither changes the actual pose estimation nor the map, but a wrongly detected loop leads to an adjustment of all the past estimations in a completely wrong direction.

2.2.4 Relation to Structure from Motion (SfM)

SfM is a technique to estimate the 3D structure of a scene from a series of 2D images and is therefore closely related to traditional SLAM. To get a good comparison in this thesis a SfM algorithm was used beside the SLAM techinques. The images given as input to a SfM method are unordered and the camera intrinsics as well the extrinsics are estimated simultaneously.

The SfM used in this thesis is called Colmap [Sch23] and counts to the incremental SfM methods [SF16].

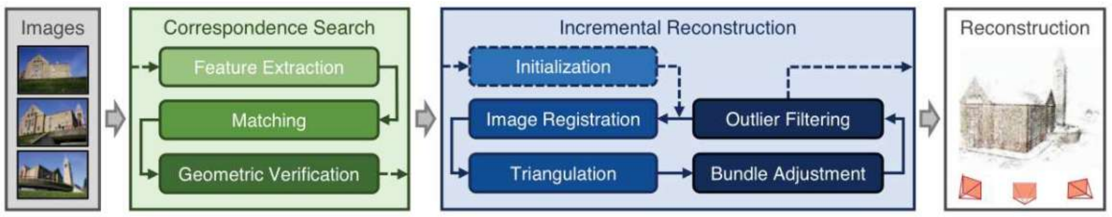


Figure 2.7: Pipeline of an incremental SfM algorithm [SF16]

Figure 2.7 shows the pipeline of an incremental SfM algorithm. The pipeline starts with feature detection and subsequent matching followed by a geometric verification. These three steps form the correspondence search which gives a so-called scene graph as output which serves as foundation for the incremental reconstruction. The reconstruction consists of an image registration, triangulation, outlier filtering and bundle adjustment.

2.3 3D Reconstruction

3D reconstruction constitutes one of the most important fields in computer vision. There is a variety of 3D reconstruction methods, a big subset of those deals with 3D object reconstruction from a series of 2D images, such as for example the SfM explained in Section 2.2.4.

Since the object data acquisition in this thesis is done with a RGB-D camera, this section focuses on the reconstruction using RGBD images, especially the reconstruction using a truncated signed distance function.

2.3.1 Truncated Signed Distance Function (TSDF)

The TSDF is based on the Signed Distance Function (SDF) introduced by Curless et al. [CL96], they proposed this method to reconstruct a 3D model from multiple depth images. Werner et al. [WAHW14] give a good explanation of SDF as well as TSDF and conduct some interesting experiments on how changes in the parameters influence the outcome of a 3D volume integration.

A d-dimensional grid is used to represent a d-dimensional environment, where each grid point is called voxel. The position of a such a voxel is defined by its centre, in the 3D case the position would be $\boldsymbol{x} = (x, y, z)$. The signed distance function would then look as follows:

$$d_i(\boldsymbol{x}) = r_i(\boldsymbol{x}) - c_z(\boldsymbol{x}) \tag{2.28}$$

Where $d_i(\boldsymbol{x})$ corresponds to the SDF for the voxel at $\boldsymbol{x}$ from the viewpoint i, $r_i(\boldsymbol{x})$ is measured depth between the camera and the nearest object surface and $c_z(\boldsymbol{x})$ represents the distance between the camera and the voxel centre along the optical axis. The SDF becomes the TSDF, if it is truncated at $\pm t$ which reduces the memory usage and is does not influence the surface reconstruction. This results to the equation:

$$td_i(\boldsymbol{x}) = \max\left(-1, \min\left(1, \frac{d_i(\boldsymbol{x})}{t}\right)\right) \tag{2.29}$$

where $td_i(\boldsymbol{x})$ now designates the TSDF for the voxel at $\boldsymbol{x}$ from the viewpoint i. Furthermore, each voxel has a weight $w_i(\boldsymbol{x})$ which describes the uncertainty of each voxel according to the SDF $d_i(\boldsymbol{x})$. In order to do a full object reconstruction from several viewing angels the variables $TD_i(\boldsymbol{x})$ and $W_i(\boldsymbol{x})$ need to be introduced. Those two variables represent the iteratively updated TSDF volume with according weights. Both $TD_i(\boldsymbol{x})$ and $W_i(\boldsymbol{x})$ are initialised with zero and the update step looks as follows:

$$TD_i(\boldsymbol{x}) = \frac{W_{i-1}(\boldsymbol{x})TD_{i-1}(\boldsymbol{x}) + w_i(\boldsymbol{x})td_i(\boldsymbol{x})}{W_{i-1}(\boldsymbol{x}) + w_i(\boldsymbol{x})} \tag{2.30}$$

$$W_i(\boldsymbol{x}) = W_{i-1}(\boldsymbol{x}) + w_i(\boldsymbol{x}) \tag{2.31}$$

Two very important parameters of the TSDF are the *voxel size* and the *truncation distance* t. The voxel size directly impacts the memory usage as well as the resolution of the resulting reconstructed 3D scene. If for a fixed 3D voxel grid the voxel size is halved, the number of voxels increase by the factor of eight and vice versa. Furthermore, with decreasing voxel size the computation time for a TSDF reconstruction increases rapidly. Henceforth it is important to find a good balance between a good memory footprint, computation time and scene resolution.

The second value is the truncation distance which influences the quantisation error of the TSDF. Werner et al. [WAHW14] suggest a two byte integer representation per voxel which would result in a quantisation of $\frac{t}{32767}$. This would suggest to choose the truncation distance as small as possible, but on the other hand it should be larger than the voxel diagonal and the noise level.

2.4 Robotic Platform

Since modern hardware grows increasingly complex, some sort of underlying software environment is needed to facilitate the work of the developer. Even simple systems like for example a single RGB-D camera needs extensive driver development and sophisticated input- output software. If the system grows larger and forms a whole robot as for example the HSR, said complexity increases even further. To develop a monolithic driver with comprehensive IO management for such versatile hardware becomes completely unfeasible.

ROS is a widely used open-source operation system for not only robots, but all sorts of embedded hardware [QCG+09]. It reduces the development complexity by giving the tools to create reusable components with precisely defined boundaries. Moreover, a well-designed communication system further eases the implementation expenditure of the developer.

2.4.1 Human Support Robot (HSR)

The specific mobile robot platform used in this thesis is the HSR from Toyota [YNK+18]. The HSR is designed to accelerate research and development (R&D) for support robots which can be used for nursing and geriatric care. Especially the demand for care for the elderly and assistance for people with disabilities has increased rapidly in the past due to people living longer and declining birth rates. Yamamoto et al. [YNK+18] hope that the compact and practical design of the HSR and the R&D coming with it will contribute to solving this problem and improve quality of life across the board.

Hardware Platform

The most important sensors and actuators used in this thesis are listed in table 2.1.

Drive system	Omnidirectional moving mechanism (max. 0.8km/h)
IMU	6DOF
Laser measuring range sensor	UST-20LX
RGB-D sensor	Xtion PRO LIVE

Table 2.1: Relevant sensors and actuators on the HSR

There is also a stereo camera as well as a wide-angle camera installed on the HSR, but these are not used in the course of this thesis, the reason for that will be explained in Section 3.2.1.

The HSR is running Ubuntu 20.04 with ROS Noetic. All the drivers are already implemented and installed.

2.4.2 Robot Operating System (ROS)

As already stated in the introduction of Section 2.4, a software framework is needed to help abstract the complexity of the robot system. ROS offers exactly this functionality, is used on the HSR and therefore also for this project. Quigley et al. [QCG$^+$09] describe ROS to have the following philosophical goals:

1. Peer-to-peer
2. Tools-based
3. Multi-lingual
4. Thin
5. Free and Open-Source

The first point refers to the communication system on which ROS is based and which also reflects the heart of ROS. Through a peer-to-peer system it is possible to run a great number of ROS nodes not only on one, but on multiple machines with a heterogeneous network. Figure 2.8 shows an example of such a distributed ROS system.

There are nodes running directly on the robotic platform which can be seen on the right, and there are also nodes running off-board on a computing cluster. A ROS master is running on one of the system to establish peer-to-peer connections, all of these nodes can then seamlessly communicate with one another without the need to send messages over a central data server. The ROS communication is based on a publish-subscriber model using messages, topics and services. Every node can subscribe to and publish on

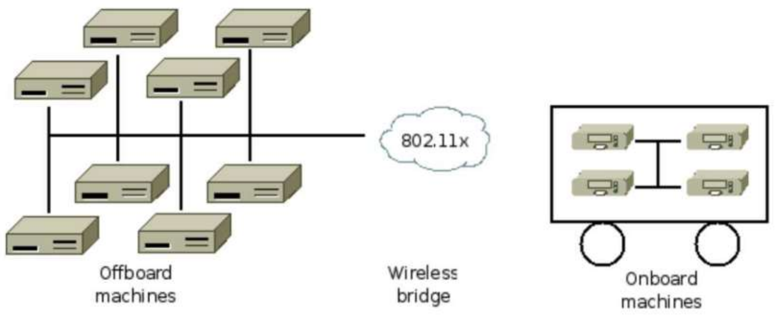


Figure 2.8: Example of a ROS network topology [QCG+09]

an arbitrary number of topics. Each of the topics has a message type which is defined with an language-neutral interface definition language (IDL). The nodes themselves are not aware, if there are any other modules subscribing or publishing on their topics. If the broadcasting routing scheme is not appropriate for a given tasks and synchronous communication between two nodes is needed, services can be used. A service can be advertised only by one single node, it needs a request as well a response message type and can be used by an arbitrary number of subscribers.

In this paragraph items two to five from the above enumerate are described only briefly, for more details on those points the paper by Quigley et al. [QCG+09] gives an in-depth look.

ROS is multi-lingual, tool-based and thin, this gives a lot of advantages when working with robot systems and projects of various sizes. Furthermore, the fact that ROS is multilingual opens up more application possibilities. Just to name two examples: Should the user work very close to the hardware and wants to implement efficient code down to the instruction level, C++ can be used. If the user on the other hand works more with complex mathematical concepts or vector arithmetics, python or matlab are possibilities. For the implementation of the underlying project of this thesis python was used to utilize libraries such as numpy [HMvdW+20], open3d [ZPK18] or matplotlib [Hun07].

The ROS implementation uses a modular design which pushes practically every function into its own module. Even essential functionalities e.g. global clock or loggers are implemented in separate modules instead of inside the master module. This highly modular design yields a very stable system with excellent complexity management with the drawback of efficiency loss.

A thin implementation means that the functional code used in ROS is completely outsourced to libraries. Modular builds by the use of CMake create small executables which are then exposed to ROS. Henceforth all the implementation complexity is fully independent from any ROS hooks and can therefore easily be reused or external code can easily be integrated respectively.

Transformation Library (/tf)

A very challenging part of robotics development are different coordinate frames. When working with a system consisting of several modules, sensors, and actuators, they will with a high probability not use the same coordinate frame. From this follows that the developer has to know the coordinate frame of the current information and transform it correctly into the desired one. This particular problem was also apparent to the developers of the tf library. Through the implementation of tf the user needs to know the names of the source and target coordinate frame, and the transformation is executed through a library call [Foo13].

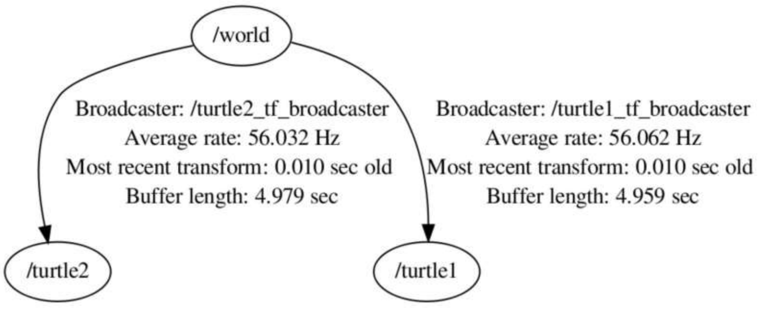

/world
Broadcaster: /turtle2_tf_broadcaster
Average rate: 56.032 Hz
Most recent transform: 0.010 sec old
Buffer length: 4.979 sec
Broadcaster: /turtle1_tf_broadcaster
Average rate: 56.062 Hz
Most recent transform: 0.010 sec old
Buffer length: 4.959 sec
/turtle2
/turtle1

Figure 2.9: Example of a tf tree taken from the ROS turtle tutorial [Foo13]

Figure 2.9 shows an example of the tf tree used in the ROS Turtle tutorial. The vertices in such a tree represent the different coordinate frames and the edges their dependencies to predecessor frames. Furthermore, information of the most recent update as well as the aveage update rate for each coordinate frames is given for each edge. In this particular example there are three different coordinate frames: *world*, *turtle1* and *turtle2*. The tf tree of HSR is with 53 vertices much larger and more complex than this illustrative example.

The tf implementation is composed of two parts, the *listener* and the *broadcaster*. All the data is collected through the *listener* and stored to comply incoming queries. Through the *broadcaster* the tf information is distributed through the system.

As already explained in Section 2.4.2 most ROS systems tend to be very distributed, therefore there are a variety of broadcasters and listeners. Each node which integrates the tf library has its own buffer which is filled by the listener part of the implementation. If new broadcasters arise in the system, they are automatically added to each listener and extend the tf structure.

Since a lot of tf broadcasters are bound to hardware components like sensors or actuators, the publishing rate of the tf information is not consistent. Due to these inconsistencies the tf library has to cope with missing and delayed data. It is imminent that in the case of missing data a valuable error message instead of invalid data is returned.

In order to ensure high quality transformation results the tf data between available time stamps can be interpolated using a spherical line interpolation (SLERP) [Kre08]. This interpolation also ensures precise transformation for tf broadcasts with low frequency, although a higher frequency increases accuracy.

Repeatability through ROS Bags

When using a robot system to evaluate algorithms or testing new software it can be become cumbersome or expensive to always make use of the hardware. One solution to this problem are simulations, through a virtual environment tests become quick, cheap, reproducible, and easy to execute. Unfortunately this method also introduces drawbacks. First of all such a simulation environment must be implemented, and it is not trivial to design the simulation framework in a way that it reflects the real world well enough for the developer's needs. Secondly, the simulation framework is not necessarily fed real world data which further increases the sim-to-real gap. Finally, simulations can be very resource hungry, because in the robot case the simulation environment has to reflect some parts of the physical world which is certainly not trivial.

Since ROS fully relies on its message oriented publish-subscriber model described in Section 2.4.2, it can simply save those messages and replay them. This can be done with a ROS feature called rosbags [FLB+23]. When using rosbags the developer can specify a number of topics which should be saved and collect them when working with the hardware. Later on the rosbag can be replayed and from ROS point of view there is no different between live hardware data and rosbag data. This can then be used to test and evaluate software without needing the hardware for each run. Henceforth the exact same movements of a robot can be replayed several times, and the results are therefore perfectly comparable.

CHAPTER 3

SLAM Evaluation through 3D Object Reconstruction

The goal of the thesis is to evaluate the quality of estimated trajectories from SLAM algorithms on a specific mobile robot platform by performing a 3D object reconstruction. To do this the camera poses provided by a SLAM algorithm are used in combination with a depth sensor to perform a volume integration of objects captured from several angles.

The main motivation to use reconstructed 3D data for the evaluation of SLAM trajectories is the complexity of ground truth trajectory acquisition. The process needed to obtain the ground truth for a corresponding robot trajectory requires a well calibrated room-level sensor array, constraining the situation where they can be deployed. However, there already exists a multitude of high quality 3D models of a variety of objects. Such models can serve as ground truth when using 3D reconstructed objects for a quality assessment.

Section 3.1 of this chapter contains the general methodology used to answer the questions asked of this work including an explanation of the chosen positioning methods. The overall software pipeline implemented to solve the given task as well as details regarding the different implementation parts is explained in Section 3.2.

3.1 SLAM Evaluation

Several SLAM methods are set up and executed on the robot. As already stated in the introduction of this chapter the acquisition of the ground truth for a given robot trajectory is a hard problem.objects Therefore, a method relying on a 3D object reconstruction is introduced in this thesis and tested in the corresponding experiments.

3.1.1 Evaluation through 3D Reconstruction

Ground truth refers to a collection of accurate and reliable measurements or reference data, which is necessary for conducting evaluations and benchmarks. Without ground truth, evaluating certain methods becomes difficult, if not impossible. The difficulty of acquiring ground truth varies for each task, but generating ground truth for a robot's trajectory is non-trivial.

To obtain precise trajectory information of a robot, external measurements are required, typically through the use of a room-level sensor array, which can be both complex and expensive. Two very prominent data sets are the EuRoC dataset [BNG+16] as well as the TUM RGB-D dataset [SEE+12], both of which include sensor data as well as a ground truth. The methods for obtaining ground truth are described in their respective papers. In the case of EuRoC, two datasets were recorded using a *Leica Multistation laser tracker* and a *Vicon motion capture system*, respectively, to provide highly accurate 3D poses. The TUM RGB-D dataset uses a *Raptor-E Digital RealTime System*, which is a room-spanning motion capture system.

Those examples showcase the difficulty of acquiring ground truth for robot trajectories. From this follows that an easier method is needed to obtain ground truth data.

High quality object models are easily available. One particularly prominent object data set is the Yale-CMU-Berkeley (YCB) dataset [CSB+17]. It contains 77 objects where the objects can be downloaded from their website, and the real objects themselves are cheap and easy to purchase. It is therefore straightforward to obtain a ground truth when reducing the trajectory problem to an object problem. The method used in this thesis is based on comparing reconstructed object models to such high quality 3D models.

For the data collection the objects will be placed on a flat elevated surface. The robot circles the object setup and takes RGB-D images and collects other sensor data. In order to get repeatable and comparable results the sensor data will be saved and later on applied to the given algorithm. This ensures that every SLAM configuration is confronted with the exact same sensor data and therefore, the results can be compared directly.

The acquired camera poses from the estimated SLAM trajectory of the robot are used to perform a volume integration using a truncated signed distance function. From each reconstructed scene the objects are extracted, registered with a corresponding high quality object model, and compared using point-to-point distances. The object extraction is done by removing the supporting plane and combining the multiple depth images of the object. The reconstructed 3D models obtained have then to be clustered to get the single objects before comparing to their high-quality counterparts. Finally the quality of the estimated SLAM trajectory can be concluded from the error magnitude of the point-to-point comparison. To get significant results several different experiments are set up. These setups take into account different camera angles, object count, or feature richness.

This comparison will certainly not yield the same accuracy as a direct comparison to the trajectories ground truth using external sensors, but it is way more flexible, simpler as well as cheaper due to availability of high quality object models.

3.1.2 Method Selection and Evaluation

Even though the visual odometry component is the central tasks to be measured, other sensors can be used to increase the robustness of the SLAM algorithm. If a robot is used for tasks like object grasping or tight maneuvering, some sensors might acquire more accurate data than others, so the robot should use all available ones to increase the precision. Therefore, we include the planar LiDAR data to the input of some of the methods to evaluate more diverse sensor setups, suitable for a variety of tasks.

One necessity for the chosen algorithms is the availability of an open source implementation for ROS. The implementation of every algorithm from scratch for the HSR would go beyond the scope of this thesis. There is already a comprehensible list of open source SLAM implementation on Github [Int21].

The following SLAM configuration were used to extract pose information

1. Hector SLAM [KMvSK11]
2. RTAB-Map [LM19] visual only
3. RTAB-Map [LM19] visual+laser scan
4. Colmap [SF16] [SZPF16]

Hector SLAM is a purely LiDAR based SLAM and available as standard ROS node for ROS Noetic and Melodic. It is used as baseline for the comparisons, since it only relies on the laser scanner and is therefore independent of camera position, number of objects, and object type. Also, changes in lighting are hardly influencing a purely LiDAR based method.

A second SLAM is RTAB-Map which is a representative of feature based visual SLAMs and has a great compatibility with ROS. One advantage of RTAB-Map is the possibility to combine a variety of sensors. This feature is also used in this thesis, because RTAB-Map is used with a RGB-D only configuration as well as in a RGB-D plus LiDAR configuration. This second configuration should make this feature based visual SLAM more robust to viewing angles, object types, and lighting.

The last algorithm used in the comparison is Colmap which is strictly speaking not a SLAM method, but a SfM as explained in Section 2.2.4. Henceforth it needs all of the input images at once and can therefore not work with live data. Nevertheless the general functionality is the same and as SfM algorithm it gives a good contrast to RTAB-Map which is a feature based SLAM. One difference of Colmap is that it does not take the image depth as input. Therefore, the pose estimation relies only on the RGB images of the RGB-D camera which introduces a scaling problem. More details on this problem are elaborated in the following section.

Pose Generation from Colmap

As already explained in Section 2.2.4 a SfM method takes all the RGB images from all the viewing angles at once and runs them through the SfM pipeline shown in Figure 2.7. The output of this pipeline is a sparse 3D model of the given scene. This sparse model is exported as a text document which consists of the camera poses, the 3D keypoints, as well as the camera intrinsics. Since the camera intrinsics of the RGB-D camera used on the HSR are already known, they are fixed beforehand and are not estimated by Colmap.

Due to the missing depth info the scale of the camera poses are only correct relative to each other, but do not have a correct absolute value. This poses a problem when employing a 3D reconstruction using the RGB-D images from the rosbags. Therefore the scale factor for each reconstruction must be estimated and applied to the camera poses before attempting a reconstruction. This is done by taking two RGB-D images n steps away from each other, calculate the corresponding PCDs and register those. With this the transformation from the first to the second PCD can be calculated. In a second step the transformation between the two corresponding 2D images is calculated by using the camera poses extracted from Colmap. The ratio of the Frobenius norm of the translation estimated from the PCDs and the translation from Colmap gives the desired scaling factor:

$$s = \frac{||t_{pcd}||}{||t_{col}||} \tag{3.1}$$

Where s is the scale factor, t_{pcd} the translation between two sequential PCDs, and t_{col} the translation between two Colmap poses. This procedure is done over all images and the median is then used as end result. Unfortunately, this scaling factor does not give a result of the desired quality and, therefore a correction factor is applied to the depth values from the RGB-D images, and some of the scale values are adjusted manually to improve the reconstruction result.

3.2 Object Reconstruction

The details of the modules used to reconstruct objects from the SLAM poses are presented here. This section explains how these modules are interfacing with each other and how the pipeline looks like.

Figure 3.1 shows a block diagram of all the modules used in this thesis and their interfaces to each other.

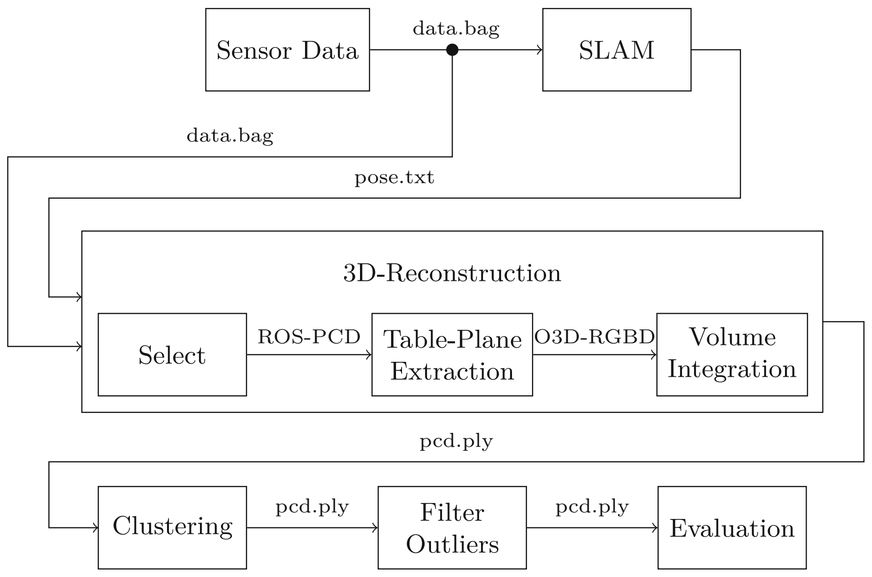


Figure 3.1: Object reconstruction pipeline from data collection to result evaluation

3.2.1 Data Collection

Since the HSR is running ROS all the modules are implemented in the form of ROS modules. The task requires no real-time capability, therefore relaxing the performance constraints of the system. Both the platform and the HSR use the same OS, namely Ubuntu 20.04 with ROS Noetic.

There are several parts which are needed to get from the raw sensor recordings to a 3D object. More details to these modules can be found in Sections 3.2.2-3.2.4.

The pipeline starts with the data collection on the HSR which is saved in ROS bags. For each test case one bag with seven topics was recorded, their details can be observed in table 3.1.

Topic	**ROS Type**
/hsrb/base_scan	sensor_msgs/LaserScan
/hsrb/head_rgbd_sensor/depth_registered/camera_info	sensor_msgs/CameraInfo
/hsrb/head_rgbd_sensor/depth_registered/image_rect_raw	sensor_msgs/Image
/hsrb/head_rgbd_sensor/rgb/camera_info	sensor_msgs/CameraInfo
/hsrb/head_rgbd_sensor/rgb/image_rect_color	sensor_msgs/Image
/tf	tf2_msgs/TFMessage
/tf_satic	tf2_msgs/TFMessage

Table 3.1: Collected ROS topics with corresponding types

The RGB image in *image_rect_color* has three color channels with a depth of 8-bit each and a resolution of 640 × 480. The depth information in *image_rect_raw* comes also in the form of an image, but with only one 16-bit grayscale channel which encodes the depth in meters. Both of those image topics are accompanied by *camera_info* topics. Finally the proprioceptive information of the HSR is saved with the topics *tf* and *tf_static* which is particularly useful to recover the relative pose of the cameras and LiDAR during the trajectory execution.

Image data collection has been carried out using only an RGB-D camera. This has been done for two reasons. An RGB-D camera achieves generally a better precision than a stereo camera when measuring distances, especially when the distance to the objects gets larger [Rod21]. The second reason is a limitation of the used robot platform. Because of the robots design, the arm with gripper is always in the field of view of the stereo camera. There is the possibility to lower the arm, but then the distance to the objects of interest increases significantly.

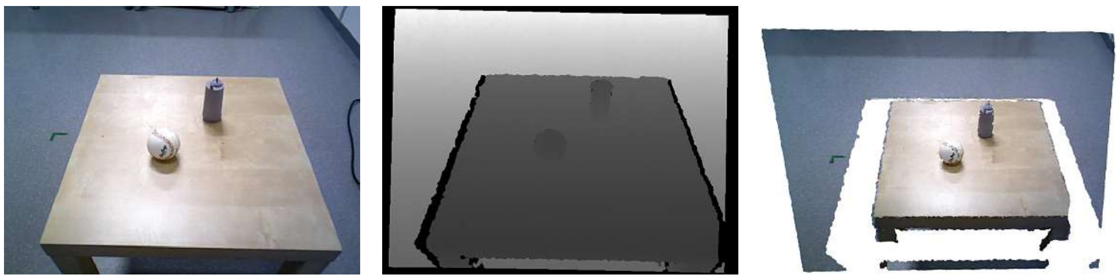

(a) RGB image (b) Depth image (c) Pointcloud

Figure 3.2: RGB-D sensor data

Subfigure 3.2a and 3.2b show the actual sensor data of the RGB-D camera and Subfigure 3.2c shows the resulting PCD. So Figure 3.2 shows the output of one view of one test case.

The bag file is used as input for the SLAM algorithms, and they are run with different configurations. Each run of a SLAM yields some sort of database output from which the poses can be extracted. These poses are then saved in csv format as text file and can be used as input for the 3D reconstruction.

The 3D reconstruction is one short pipeline in itself which consists of three functions. Firstly, the right images from the ROS bag files must be extracted by the use of the pose data from the SLAM algorithm. This is done by using ROS time stamps which have a resolution up until the nano second range.

When a RGB-D image with valid time stamp and pose is reached, the rgb- and depth part of the image are converted into a ROS PCD. This PCD is then given as argument to the Table-Plane-Extraction service which returns ROS PCDs for the objects.

Those PCDs are converted to open3d RGB-D objects which can be used in the TSDF integration and receive the reconstructed 3D object. The reconstructed objects are saved as standard PCD in .ply format for further processing.

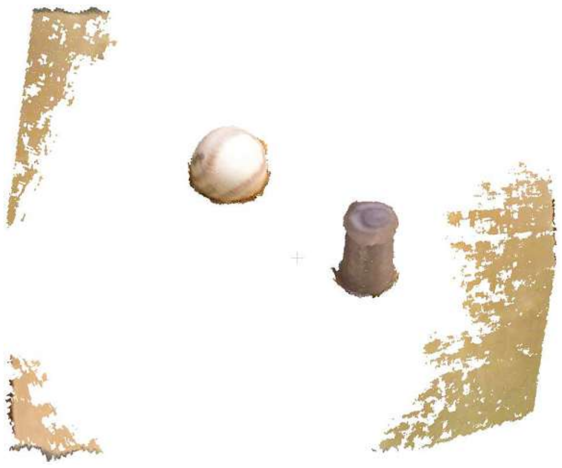


Figure 3.3: Finished reconstruction of the *baseball+canister* test case with RTAB-Map visual only and camera lift down

An example of a finished reconstruction is presented in Figure 3.3. All the camera views have been integrated using their respective poses and the table plane has been extracted for every scene pointcloud. In this finished reconstruction both objects are in the same PCD, and there are still artefacts from the table plane. The final step required before object comparison and result evaluation is to cluster and filter out the outliers from the reconstructed PCD. Two scripts are employed to accomplish this, both of which output a PCD in .ply format.

(a) Baseball (b) Canister

Figure 3.4: Clustered and filtered objects

The final result of the object reconstruction can be observed in Figure 3.4. Through clustering and filtering the artefacts were removed and the PCDs separated in two individual pointclouds. Both objects can now be evaluated in order to judge the quality of the SLAM pose estimation. The details about the result evaluation and the particular experiment setups are explained in Chapter 4.

3.2.2 Table Plane Extraction

In order to separate the objects on the desk from the surroundings and from the other objects respectively a plane extractor is needed. Since the table is completely flat, Random Sample Consensus (RANSAC) [FB81] can be used to find the dominant plane.

The institute already provides a ROS node for this task [vt22]. This module uses already several details known about the test environment. First, all points below the table surface are removed since the geometry of the table is fixed and known to the node. Moreover, the position of the camera relative to the floor is obtained through the /tf tree. Therefore, results obtained from RANSAC that are not parallel to the table surface are discarded. Once all the points which are not above the table plane are removed, the remaining points are clustered with the use of DBSCAN. Finally, the point cloud of each object on the table is returned.

3.2.3 Object Reconstruction

For the object reconstruction the Truncated Signed Distance Function (TSDF) from Open3d [ZPK18] is used. A TSDF volume is created for each object, and all the views are integrated to said volume using the poses extracted from SLAM. Furthermore, the volume generation requires two tunable parameters for the reconstruction:

1. **Voxel Length:** Specifies the length of a voxel. The smaller the voxel length, the higher the resolution of the resulting 3D image. One big drawback of a small voxel length is an increase in computational overhead as well as memory usage as explained in section 2.3.1. Furthermore, with a shorter voxel length the result is more prone to noise induced by the depth sensor. For the 3D models reconstructed in this project, a voxel length of $0.0005m$ has been chosen.

2. **SDF Truncation:** Is the truncation value for the SDF. A small truncation value reduces the error when performing the volumetric integration, due to a finer quantization. But as described in Section 2.3.1, the SDF truncation value should also be chosen according to the noise level. If the value is set too small, the SDF algorithm has less freedom when matching the PCDs which can drastically worsen the result. Signs of a too small truncation value are double edges or double surfaces in the reconstructed value. If the value is too large, the algorithm has more freedom when integrating. Henceforth the integrated surfaces can become distorted and contain bulges. The truncation value for this project was chosen as $0.0175m$.

A good combination for those parameters was found through trial-and-error. Several 3D models with different parameters were reconstructed and the results inspected visually. Once a good trade-off between resolution, noise, and surface quality was found, the values were fixed for all further experiments.

3.2.4 Clustering

Following a successful reconstruction step, the resulting pointclouds contain several objects as well as artefacts of the table surface (see Figure 3.3). To get rid of these artefacts and split off each object into its own point cloud, a clustering is necessary. In this project DBSCAN [EKS+96] from the open3d library is used which is a density-based clustering algorithm. Schubert et al. [SSE+17] argue in their article why and how DBSCAN is still a good and important algorithm.

There are two reasons why this algorithm in particular was chosen for this project. Firstly, the simplicity of the density based clustering only has two tunable parameters which are very intuitive to set:

1. **eps:** The ϵ value denotes the distance each point in a cluster must have to a certain number of neighbours in order to still belong to said cluster. The smaller this value is chosen, the finer the clusters will get. If it is too small, a lot of clusters will be split up. If the value is chosen too large, different clusters are grouped together. Furthermore, a too large eps value results in a lot of memory usage when executing the clustering. When the pointclouds are as dense as in this project, the calculation of the eps-neighbourhood can easily use up more than 16GB of memory when the eps value is not chosen carefully. For the clustering in this project eps was chosen as 0.0085.

2. **min_points:** Is the number of minimum points required to form a cluster. The value for min_points was chosen as 10.

The definition of the two parameters is not completely equal in the open3d version used in this project and the original paper [EKS+96].

The second reason this clustering was chosen is the open3d implementation. It can be used directly with the point clouds, and there is no compatibility or additional configuration necessary.

After the point clouds are clustered, the statistical outliers are removed from the image. The Open3d function for this takes two parameters the number of neighbours which are taken into accounts for the average distance calculation and the maximal standard deviation from the average distance. The parameters for this filter are chosen to be non-aggressive, because the purpose of this function is not to change the 3D object, but to remove the remaining artefacts which would influence the point-to-point distance measurement.

CHAPTER 4

Experiments and Results

The general method and the implementation details of the project accompanying this thesis have already been introduced in Chapter 3. Section 4.1 of this chapter gives an in-depth look into the experiment setup and Section 4.2 depicts the final results of this project. A discussion of the results, accomplishments, and limitations of the proposed method can be found in Section 4.3.

4.1 Experiment Setup

As already explained in Section 3.1.1 the objects for each test case were arranged on an elevated surface around which the HSR could move. A simple beige side table which is $45cm$ high and has a $55 \times 55cm$ square surface was used.

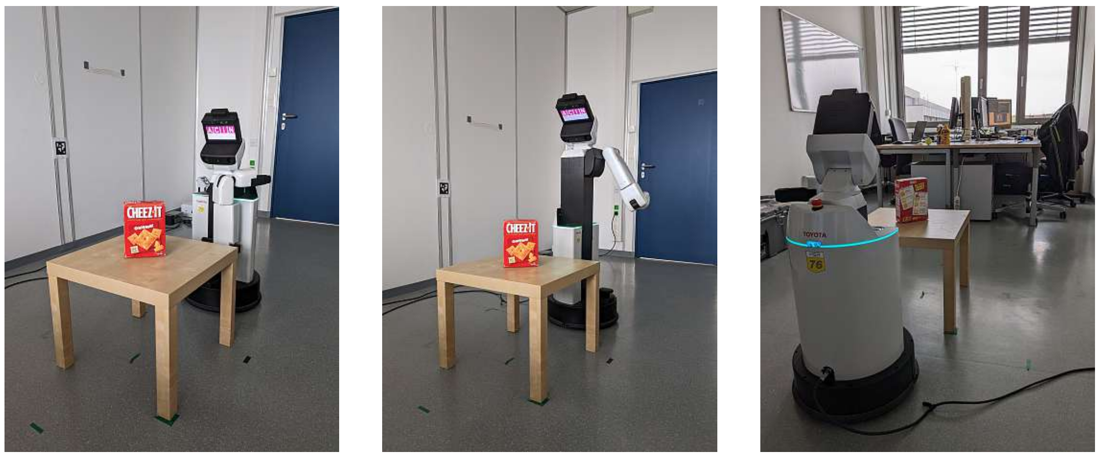


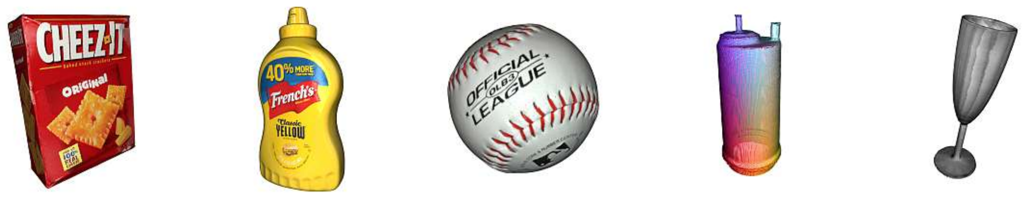
Figure 4.1: Experiment setup with HSR, elevator up and down as well as picture from behind

Figure 4.1 shows the room in which the experiments were conducted with the cracker box on the desk. The leftmost and the middle image show the HSR looking down on the object of interest once in the elevator down and once in the elevator up position. The right image shows a picture in the other direction with the big window for natural light as well the desks which also influence the SLAM results due to additional features.

4.1.1 Objects

The object selection is a crucial part in designing the experiment. Therefore, it is essential to choose objects with varying shapes, sizes, colors, and surface textures. Moreover, the objects should be common enough to ensure that the outcomes are comparable to those of other researchers. The YCB [CSB$^+$17] dataset provides a broad range of objects that meet most of these requirements. Additionally, several objects with high-quality 3D object models are available at the institute. Ultimately, the following six test subjects were chosen for the experiments:

1. Cracker box (YCB)
2. Mustard (YCB)
3. Baseball (YCB)
4. Canister (Institute)
5. Champagne glass white (Institute)
6. Champagne glass transparent (Institute)



(a) Cracker box (b) Mustard (c) Baseball (d) Canister (e) Glass

Figure 4.2: Reference objects

Figure 4.2 displays images of the reference objects used in this study. The cracker box and mustard bottle are both common objects that possess large size, rich color, and surface information, and thus are expected to yield good pose data. Conversely, the baseball has relatively few features due to its small size, smooth surface, and lack of significant color information, making 3D reconstruction more challenging. Similarly, the canister has very few discernible features, aside from small tubes at the top, and is entirely gray in color. Of particular interest is the glass, as two versions were used, a matte white version that is not translucent and a transparent version. Compared to the other objects, the glass has the worst color and surface information, and is expected to result in relatively poor 3D reconstruction accuracy.

Before commencing experiments, it was hypothesized that the objects could be ranked in terms of 3D reconstruction difficulty as shown in Figure 4.2, with the cracker box (Figure 4.2a) being the easiest object to reconstruct and the transparent glass (Figure 4.2e) being the most challenging.

4.1.2 Number of Objects

As just explained it is expected that the quality of the estimated trajectory and thereby the quality of the 3D reconstruction is proportional to the feature richness of the object of interest. The number of features can of course be increased by increasing the number of objects on the table.

The idea is to increase the 3D reconstruction quality of a small featureless object like for example the baseball by adding additional objects like the cracker box or the mustard and thereby improving the pose estimation.

Of course more objects mean more occlusion which should also be avoided, because this would then create holes in the reconstructed objects which would aggravate the object comparison.

4.1.3 Robot Position

The object reconstruction quality is not only dependent on the feature richness of the objects and the algorithm used, but also on a variety of external parameters. Two of the most prominent parameters are the robot position and the lighting source. The impact of the lighting source is discussed in Section 4.1.4.

The HSR has an elevator where the head position can be adjusted vertically by $32cm$ which can change the pose estimation as well as the picture quality drastically. For the experiments two different lift positions were chosen, once the bottom position which places the camera at $97cm$ of the ground and once at the fully up position which places the camera at a height of $127cm$. When the elevator is at the up position, the head of the HSR must be tilted further to look directly on the desk. Therefore, when measuring the camera position the lift height of $32cm$ gets reduced to a difference of $30cm$.

The camera position in vertical direction changes two aspects of the 3D reconstruction. Firstly, the viewing angle on the objects is changed, which also changes the occlusions. When the lift is at the bottom position, the occlusions for single objects are generally less, if the number of objects is increased, however, the occlusions can occur more frequently. With a raised lift, the occlusions for single objects (e.g. Baseball) can intensify, but due to the steeper viewing angle the occlusion induced through multiple objects decreases.

The second difference emerges in the pose estimation. Through the shallow viewing angle of the camera in the bottom position a lot of the surrounding office is also in the frame, which can enhance the SLAM performance. Additional feature points detected in the background can make a big difference, when the test object has poor features. This number of background features decreases when using a steeper viewing angle through a raised lift.

4.1.4 Lighting

Lighting is one of the most important factors when working with image capturing or video recording. Since the pose estimation depends heavily on the image quality, the quality of the light source is very interesting. In the ideal case there is a large areal diffuse light source which would illuminate the objects from all sides equally and would reduce shadows nearly completely.

The lab where all the experiments for this project were conducted is unfortunately very limited in terms of different lighting methods. There are two fluorescent tubes on the ceiling and a large window. For this reason only two lighting scenarios are sensible. Firstly, an experiment with controlled light, which means the window shutter for the large Window closed and both fluorescent tubes turned on. The second set of experiments could be taken with natural light, which means turned off room light and fully opened window shutters.

Since the results of experiments with natural light fluctuate heavily due to the dependence on day time and weather condition, it was decided to omit this test case. Furthermore, the image quality does not change noticeably during daytime because of the window size, bright room, the automatic white balancing, and exposure time selection.

4.1.5 Experiment Runs

According to the above mentioned criteria a number of test cases were defined. As already discussed in Section 4.1.4, all of the data was collected using controlled light. This leads to the following test cases:

1. **Single objects & lift down:** Each of the objects was placed in the middle of the table, and the robot was moved around the desk. For this set of test the lift and thereby the camera was at the bottom position.
2. **Single objects & lift up:** The same was repeated with all objects, but the camera lift up.
3. **Multiple objects:** Three test cases were created with two or three objects on the table, namely:
   a) baseball & canister
   b) cracker box & mustard

c) baseball & canister & cracker box

Each of those tests was conducted with the camera lift down.

The trajectory of the HSR was created by hand using a remote control to guide it around the table and retrieve a video sequence of $\approx 1 - 1.5min$ per test case.

4.2 Results

Figures 4.3 to 4.5 show three examples of how the reconstructed objects and the comparisons look like. The pictures of the RGB objects as well as the Cloud-to-Cloud (C2C) scalar field objects were taken with a software called CloudCompare. This software was used for quick manual comparisons and debugging. The actual comparisons that are presented below were generated with a python pipeline which is purely based on *open3d*, *numpy* and *matplotlib*.

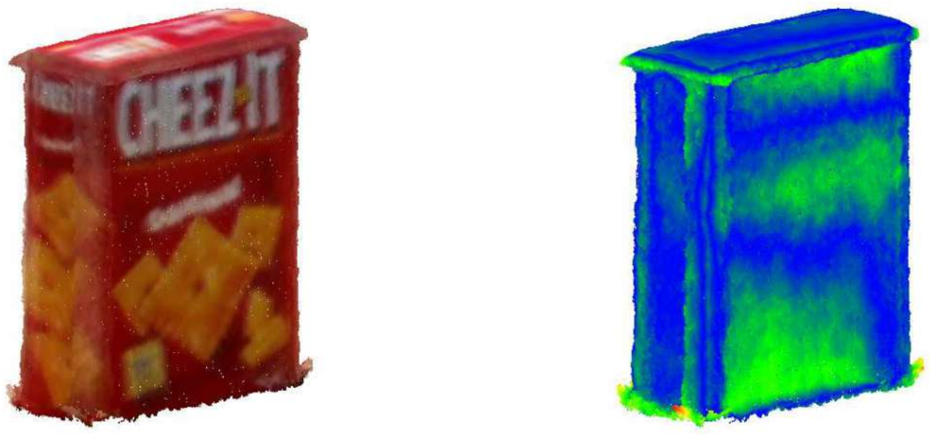

(a) RGB object (b) C2C scalar field

Figure 4.3: Results for 3D reconstruction on the cracker box

The comparison was taken from reconstructed image to optimal image which means that, if there are points missing in the reconstructed object, like in Figure 4.3, the cracker box bottom, it does not influence the comparison. Each object is registered with its corresponding optimal object (see Figure 4.2), and a point to point (P2P) comparison is conducted. The result of this P2P comparison is given in meters, which corresponds to the distance of any given point to the optimal one. In the example figures the results can be seen in the C2C scalar field where those distances are encoded with colors.

At the bottom of each reconstructed object a kind of ragged edge can be observed, especially the baseball in Figure 4.5 shows this behavior very prominently. Due to the

(a) RGB object (b) C2C scalar field

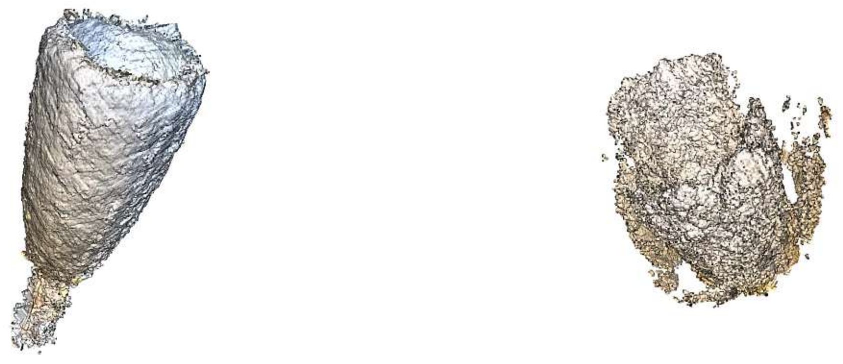
Figure 4.4: Results for 3D reconstruction on the mustard

(a) RGB object (b) C2C scalar field

Figure 4.5: Results for 3D reconstruction on the baseball

limited depth accuracy of the used RGB-D camera and image noise, the objects and the table plane fuse together and the table plane extractor cannot completely remove all the points which belong to the table. Since the baseball is rather small and completely round, this problem intensifies.

Unfortunately, the object reconstruction did not work for the transparent version of the glass (Figure 4.2e). Due to limitations of the used RGB-D camera only a few points per view are captured which is not enough for the volume integration. Therefore, the transparent glass object is excluded from the following evaluations. The matte version of the glass is detected to a large part, but only with the camera in the down position. With a raised lift the reconstruction result is again not usable for further comparison.



(a) Elevator down (b) Elevator up

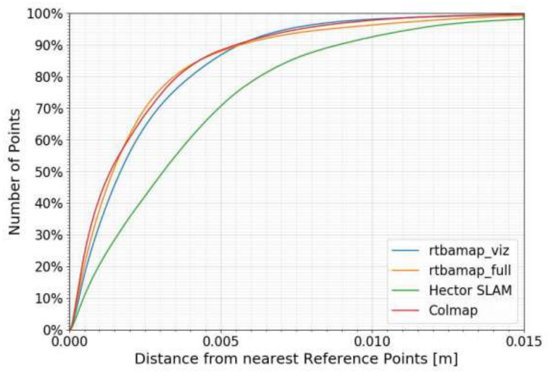
Figure 4.6: Results of 3D object reconstruction on the matte glass with RTAB-Map visual only

Figure 4.6 illustrates the result of an object reconstruction using only the matte version of the glass from both camera positions. The reconstruction result with the camera down in Figure 4.6a is good enough for further comparison. The missing glass bottom does not influence the evaluation, since the object comparison is done from reconstructed object to optimal object. Issuing the reconstruction with an elevated camera the results become completely unusable, as can be observed in Figure 4.6b.

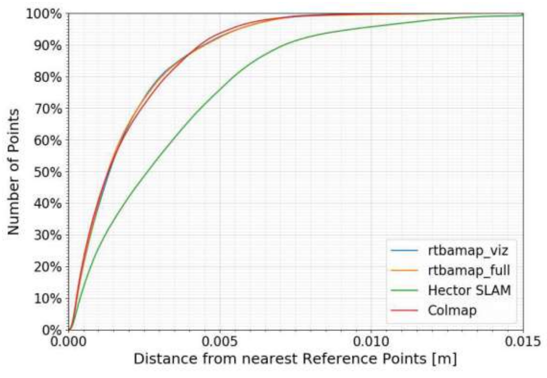
The diagrams in Figures 4.7 to 4.11 illustrate the results of this thesis. On the x-axis of each diagram, the distance of each point to the optimal one is depicted. The range of the x-axis ends at $15mm$, as the error of most points is below this threshold. On the other hand, the y-axis displays the number of points at each distance. Since the number of points per object varies due to their size, the y-axis is normalized and presented in percentage. Each curve corresponds to a cumulative histogram, which starts at 0% and increases up to 100%. Once the distance reaches $15mm$, each curve jumps to 100%, and the height of this jump is proportional to the number of points that have an error larger than $15mm$. The distance resolution for each curve is $0.0293mm$, considering that the histograms were generated using 512 bins. The speed at which a curve reaches 100% reflects the quality of the corresponding reconstructed object. A faster approach to reaching the maximum value indicates a better reconstruction quality and thereby a more accurate SLAM trajectory.

4.2.1 Single Object Comparison

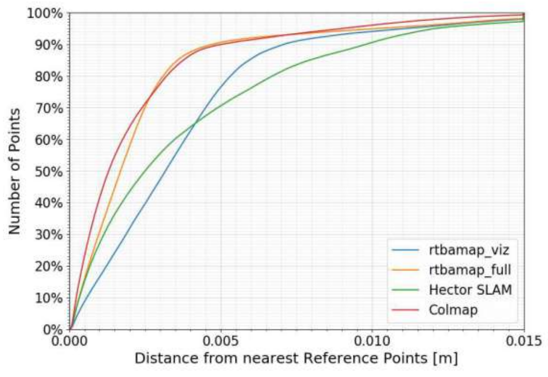
The Figures in 4.7 show the results when placing a single object on the table. The experiment has been conducted with all five objects (see Figure 4.2), all four pose estimation configurations, and the elevator in the down position.



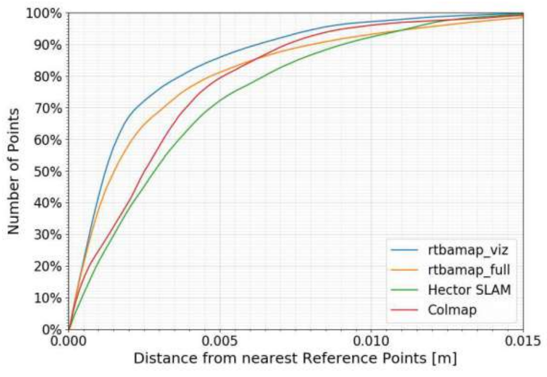
(a) Average



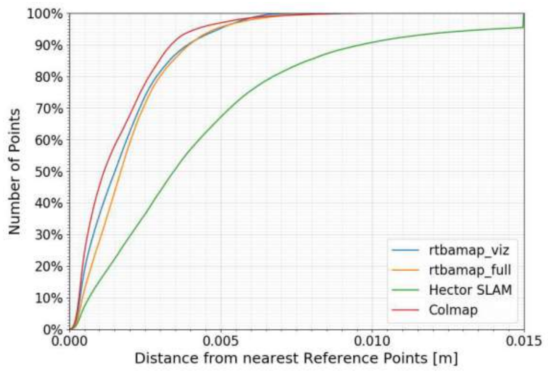
(b) Cracker box



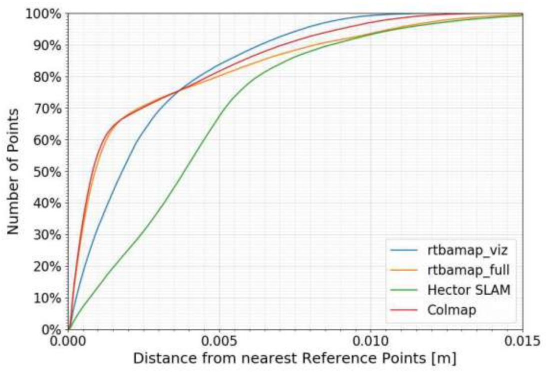
(c) Mustard



(d) Baseball



(e) Canister



(f) Glass

Figure 4.7: Single object comparison

As can be seen, the results of all the reconstruction configurations are relatively close together. The errors of all the diagrams are to a large part below $15mm$, the worst case being Hector SLAM with the canister object in Figure 4.7c where less than 96% of the points have an error below this threshold. This worst case, however, is not caused by the canister object itself, since Hector SLAM is a LiDAR only method. The quality of this result is most probably caused by some unfortunate robot trajectory which hampers the localization.

	hector	**rtabmap_viz**	**rtabmap_full**	**colmap**	**Avg**
Number of points with an error below $5mm$ in percent					
Cracker box	75.84	92.58	92.33	93.61	88.59
Mustard	70.58	76.54	90.59	89.92	81.91
Baseball	72.21	85.87	81.12	79.31	79.63
Canister	67.18	95.20	95.55	96.92	88.71
Glass	67.40	83.71	80.10	81.64	78.21
Avg	70.64	86.78	87.94	88.28	83.41
Number of points with error below $7mm$ in percent					
Cracker box	89.38	98.36	98.27	98.46	96.12
Mustard	81.39	89.66	92.81	92.70	89.14
Baseball	82.58	92.00	87.63	89.06	87.82
Canister	81.23	99.80	99.11	99.20	94.84
Glass	83.85	92.46	86.86	89.62	88.20
Avg	83.69	94.46	92.94	93.81	91.22
Number of points with error below $12mm$ in percent					
Cracker box	97.82	99.96	99.78	99.95	99.38
Mustard	94.86	95.70	96.09	97.79	96.11
Baseball	96.73	98.55	95.55	97.41	97.06
Canister	93.49	100.00	100.00	99.96	98.36
Glass	96.57	99.92	97.17	99.33	98.25
Avg	95.89	98.83	97.72	98.89	97.83

Table 4.1: Values for thresholds $5mm$, $7mm$ and $12mm$ from the diagrams in Figure 4.7

The number of points with an error below the thresholds $5mm$, $7mm$ and $12mm$ has been summarized in Table 4.1. This table contains the same results as Figure 4.7 in a numerical form. It can be seen that apart from two exceptions, with 76.54% and 79.31%, more than 80% of the points have an error below $5mm$ when employing visual SLAM methods or SfM. This drops to around 70% when using no visual data but only LiDAR with Hector SLAM. When looking at a threshold of $7mm$ every algorithm with every objects performs above 80% with RTAB-Map with LiDAR data on the canister even reaching a maximum of 99.80%. After the threshold is further increased to $12mm$, all the VSLAM methods perform above 95%, only Hector SLAM has two objects with a lower performance. Furthermore, the Table 4.1 shows the average performance for each object which cannot be read directly from the diagrams.

Overall, the ranking of the pose estimation methods is plausible, but not quite as expected. The RTAB-Map with additional LiDAR measurements outperforms the visual only RTAB-Map, but only by a small margin. It was expected that the additional LiDAR measurements would influence the quality of the estimated trajectory and thereby the 3D object quality more. Furthermore, it was anticipated that Colmap would outperform the SLAM methods, since it works on all the available images at once which gives it a considerable advantage over SLAM which works on live data. However, this is not the case, the quality of the trajectory estimated by SfM is approximately equal to RTAB-Map with LiDAR information if not worse. This could stem from the fact that only one RGB image per second was used for the camera pose estimate in Colmap. Furthermore, the scale factors which are necessary for the reconstruction were tuned manually which is certainly not optimal. Another explanation is that the pose estimation with RTAB-Map as well as with Colmap have reached the limit of the RGB-D camera in terms of depth accuracy, and the errors are induced only by sensor noise. The significantly worse performance of Hector SLAM fully met the expectations, since it only uses the LiDAR without any visual data.

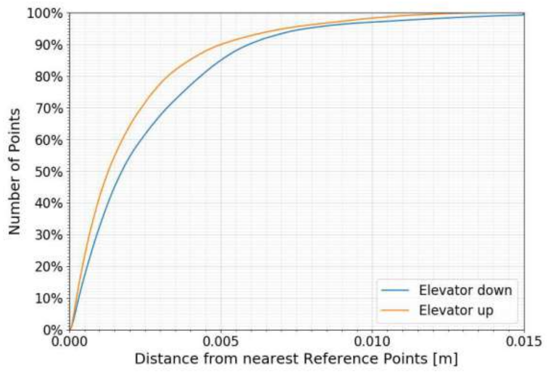
The predictions made in Section 4.1.1 were fulfilled, and the best reconstruction quality was achieved by using the cracker box in Figure 4.7b and got worse for every following object except for the canister. It was expected that the canister due to its smooth surface lack of features would perform in the vicinity of the glass or the baseball but it performs as good as the cracker box. But overall, it seems that SLAM as well as SfM could make significantly better pose estimations due to the feature richness of the bigger objects.

One especially noteworthy object in the single object comparison is the mustard bottle in Figure 4.7c. In this reconstruction Hector SLAM outperforms the visual RTAB-Map for the first 65% of the points and furthermore, the visual only SLAM scores considerably worse than the SLAMs with LiDAR data or SfM. After reviewing the video sequence saved in the rosbag, it was discovered that the robot motion is very jerky. Furthermore, a person is walking through the robot's field of view during the recording which is a dynamical object to SLAM. It seems that the additional LiDAR data or the SfM approach respectively eradicates this poor sensor data, but the visual only approach suffers.

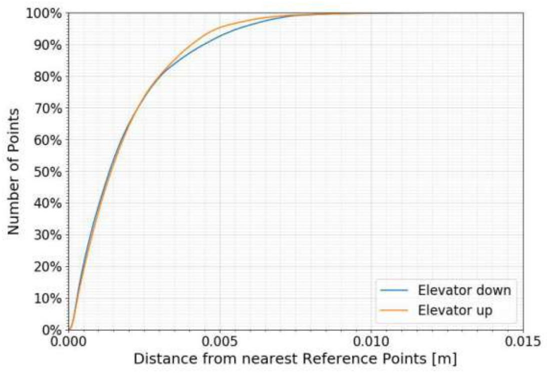
4.2.2 Impact of Elevator

The results in Figure 4.8 show the impact of the camera position on the quality of the estimated trajectory. In Section 4.1.3 it was predicted that the overall pose estimation would be better with the camera lift position down.

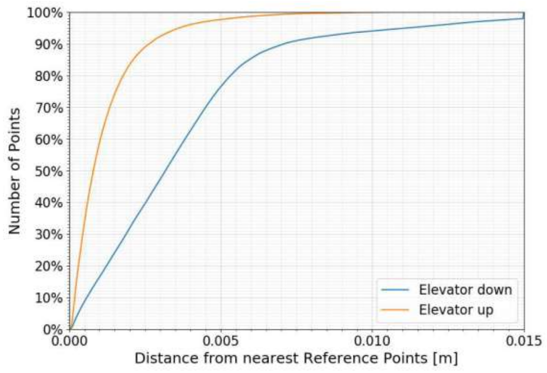
The results presented here suggest otherwise. The cracker box in Figure 4.8b and the baseball in Figure 4.8d confirm the expectations. With the cracker box being a rather big object and having a lot of features the camera height does not have a noteworthy impact on the resulting poses. The baseball on the other hand is very small and round with very little features, therefore, the down position works considerably better. Due to the features introduced by the background visible caused by the steep viewing angle the SLAM performance is improved. The trouble case is the reconstruction of the mustard shown in Figure 4.8c. In this test case the quality of the estimated trajectory with the elevator up position outperforms the down position substantially. As already stated in



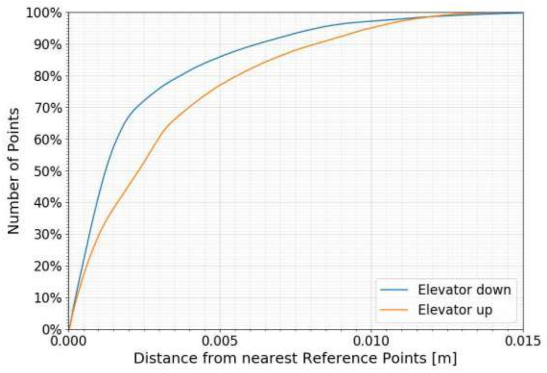
(a) Average



(b) Cracker box



(c) Mustard



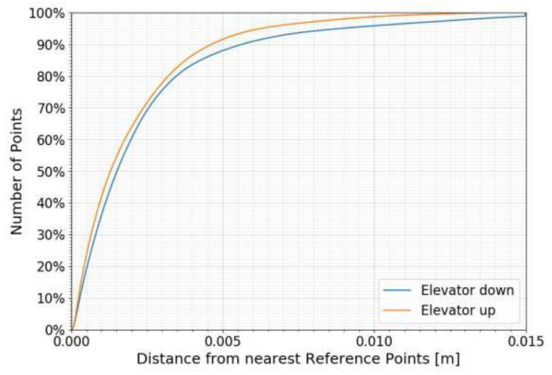
(d) Baseball

Figure 4.8: Impact of Elevator with rtabmap_viz

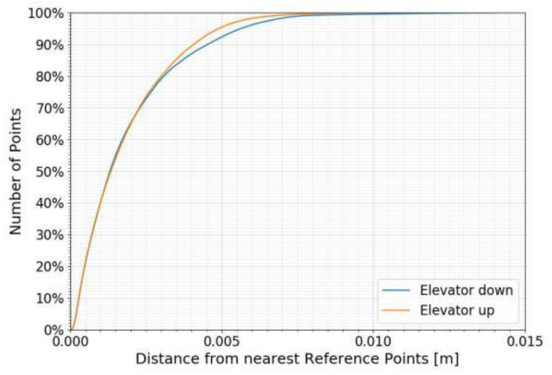
Section 4.2.1 this probably stems from the jerky video sequence as well as the person walking through the field of view.

Figure 4.9 shows the result with RTAB-Map and additional LiDAR measurements. When comparing 4.8 and 4.9 the differences are only marginal. In the average case shown in Figure 4.9a both cases perform slightly better and are closer together. Furthermore, the performance of the mustard bottle test case is significantly better than without LiDAR data. Overall, the elevated camera position still outperforms the test cases with the camera down position.

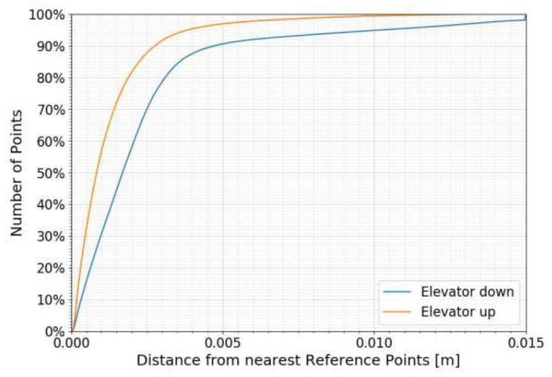
It is interesting to see that even with this difficult condition the SLAM algorithm is still able to estimate the trajectory within an reasonable framework.



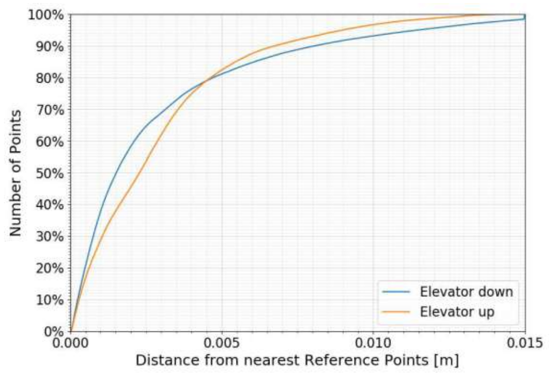
(a) Average



(b) Cracker box



(c) Mustard



(d) Baseball

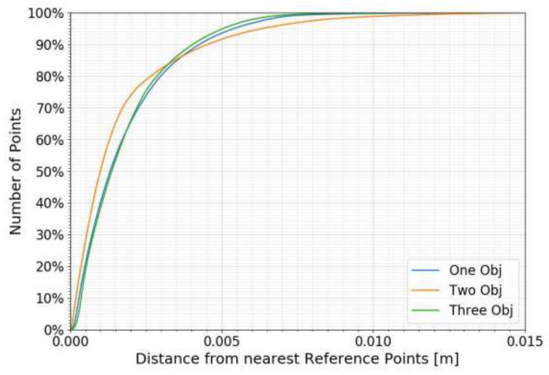
Figure 4.9: Impact of Elevator with rtabmap_full

4.2.3 Impact of Number of Objects

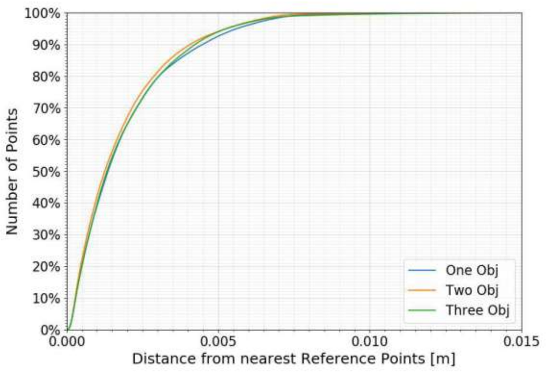
The last set of test cases presented in Figure 4.10 shows the impact of different numbers of objects on the estimated trajectory. As explained in Section 4.1.2 the quality of the estimated trajectory should increase with the number of objects.

Again the predictions are not completely true for this test case. The test case was taken once with every single object, which means these are the same results as in Figure 4.7, once with two objects (*baseball+canister* and *cracker box+mustard*) and once with all three objects (*baseball+canister+cracker box*).

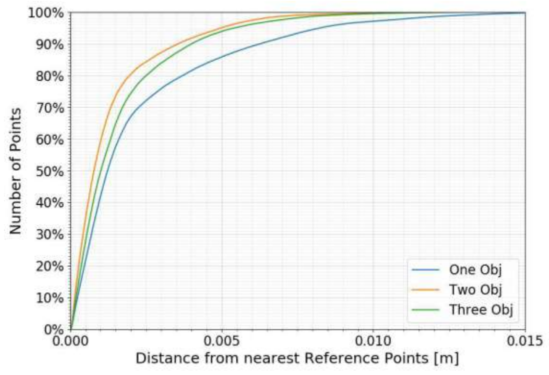
When inspecting the average diagram in Figure 4.10a the reconstructions with different number of objects do not really have an influence on the resulting quality. With the cracker box in Figure 4.10b the influence of the number of objects is negligible, which is not surprising, since the cracker box already has a lot of features. For the canister in Figure 4.10d the influence is also only marginal, which would contradict the predictions, since the canister is a smooth gray featureless object. Only the baseball in Figure 4.10c shows a difference with the number objects, which makes sense considering the poor features on the baseball. On the other hand it was not expected that the two-object test



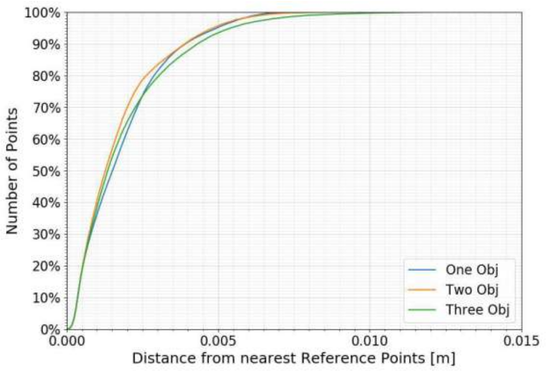
(a) Average



(b) Cracker box



(c) Baseball



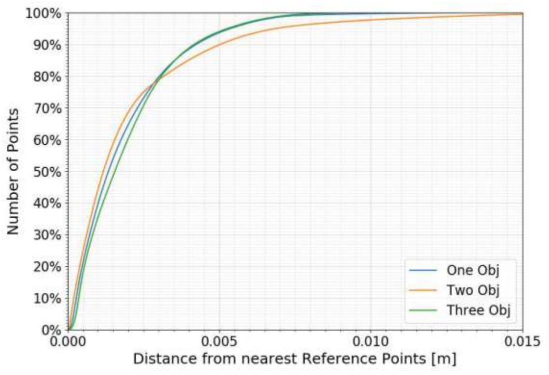
(d) Canister

Figure 4.10: Impact of multiple objects with rtabmap_viz

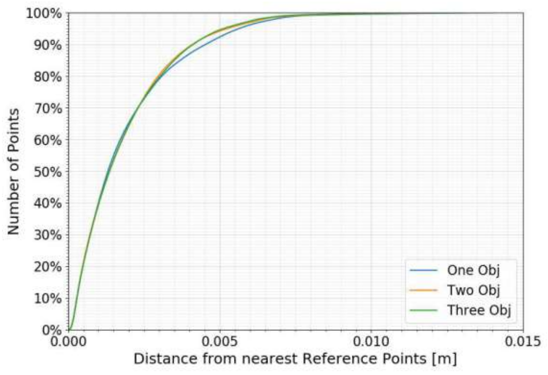
case outperforms the three-object test case. This could also stem from the data quality in the rosbag.

Figure 4.11 shows again RTAB-Map with additional LiDAR data. Similar as with the elevator test cases the result between RTAB-Map visual only and full RTAB-Map are practically the same. For the baseball test case in Figure 4.11c the two and three objects curves are in the expected order in contrast to Figure 4.11c where the two figure test case outperforms the one with three figures. Nevertheless, the difference with a peak at 3% is so minimal that it is hardly noteworthy.

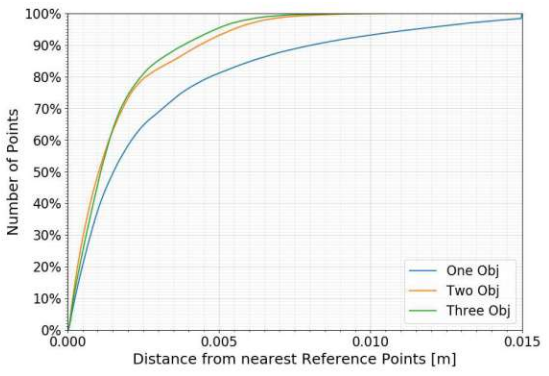
Overall, it can be said based on the results that the features of the background are more important for the trajectory estimation than the number of objects. Moreover, for the test cases with varying camera position as well as for the multiple object test cases it can be said, that the additional LiDAR data hardly makes a difference at all.



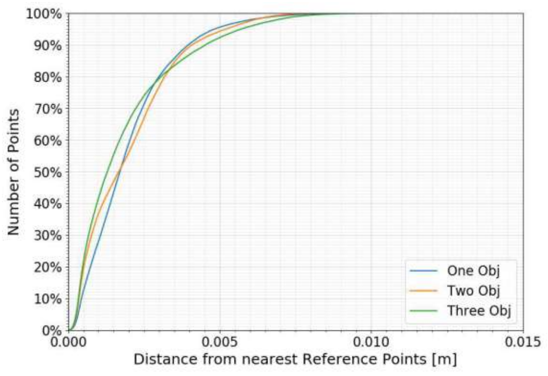
(a) Average



(b) Cracker box



(c) Baseball



(d) Canister

Figure 4.11: Impact of multiple objects with rtabmap_full

4.3 Evaluation of Accomplishments and Limitations

In the final section of this chapter, the results of the conducted experiments are summarized. Answers regarding the accomplishments as well as about the limitations of the proposed method are discussed and further outlined.

4.3.1 Object Grasping Evaluation

Since one particularly important task of human support robots is the manipulation of objects in their operational environment, the applicability in this area is very interesting. The question is, if the quality of the reconstructed object models suffices for novel grasping techniques. For this reason the methods proposed by ten Pas et al. [TPGSP17] were used to test the possible grasping solutions for some of the reconstructed objects.

The objects used for grasp pose detection where the cracker box, the mustard, and the baseball. The pose estimation was done with the RTAB-Map visual only SLAM and the camera in the down position. Figures 4.12 and 4.13 show possible grasping solutions generated by an open source implementation of the proposed paper [mar22].

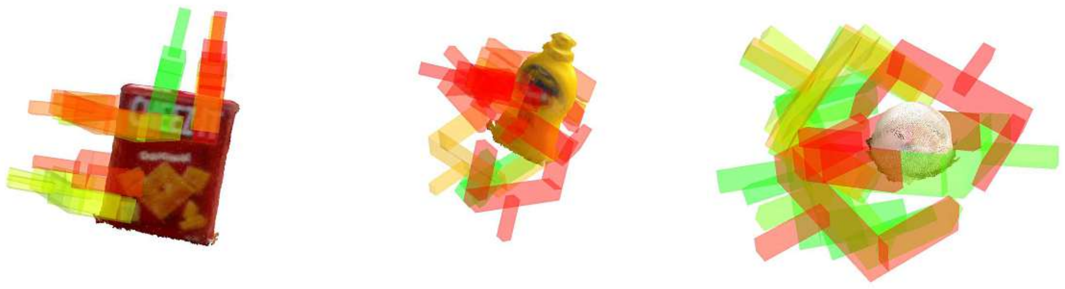


Figure 4.12: Grasping solutions from the front

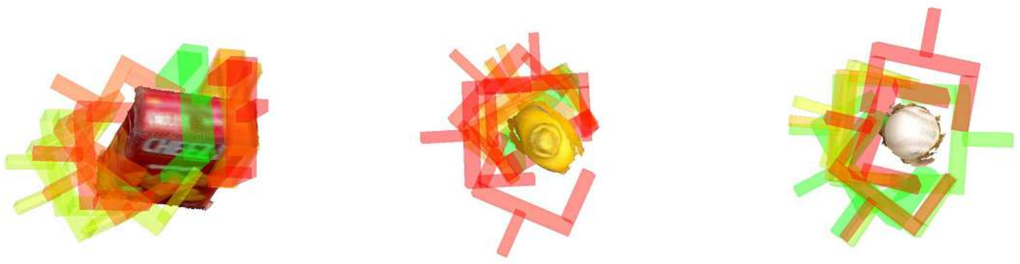


Figure 4.13: Grasping solutions from the top

The gripper arm is shown in the position prior to closing. The solutions produced for the cracker box are mostly usable. There are some attempts to grasp the box diagonally, but in general the solutions are reasonable. The same applies for the baseball although it is smaller in size. In the case of the mustard there are some attempts which would obviously fail, but nevertheless, the grasp pose detector finds some good solutions overall.

In general it can be said that the errors observed in the evaluation suggest that the reconstructed object models are sufficient for grasp pose detection. Furthermore, the results presented in Figures 4.12 and 4.13 strengthen this assessment.

4.3.2 Sensor Limitations

In Chapters 3 and 4 the depth noise of the used RGB-D camera is mentioned several times. Nguyen et al. [NIL12] wrote a comprehensive paper where the axial as well as the lateral noise of the Kinect sensor is modeled empirically.

Figure 4.14 shows a diagram from said paper where both the axial and lateral noise components are displayed dependent on z-distance and viewing angle. It can be observed that the lateral noise increases linearly with increasing z-distance of the object, while the axial noise changes in a quadratic manner. In the range $0.5m$ to $1.5m$ the lateral noise

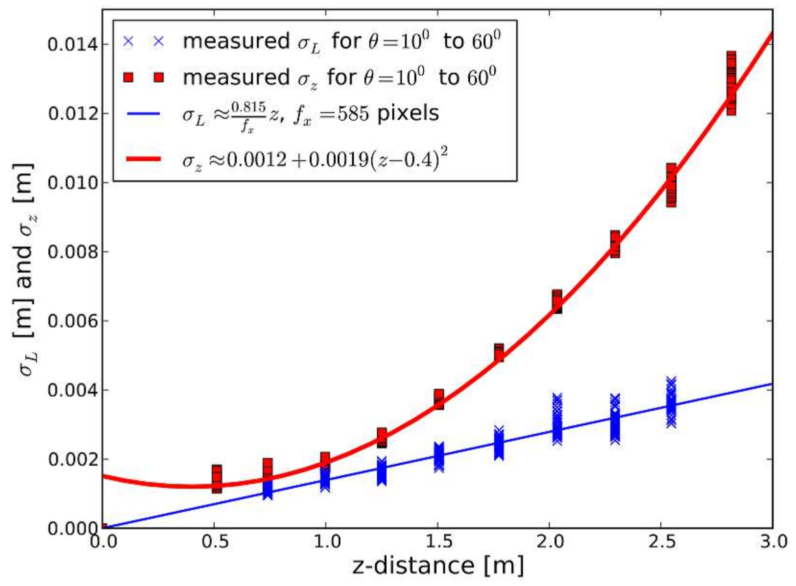


Figure 4.14: Lateral and axial noise model of the Kinect sensor from [NIL12]

component has a maximum of around $4mm$ and the axial noise component of around $2mm$. This corresponds well with the findings from Section 4.2. Around 80% of the points of each reconstruction employing the depth sensor have an error below $5mm$. This leads to the conclusion that a large part of the inaccuracies in the SLAM poses, and therefore in the reconstruction quality, do indeed stem from the RGB-D sensor. Furthermore, it can be assumed that a reduction in sensor noise, through the use of a different depth sensor, would improve the trajectory estimation, while still using the same methods as employed in this thesis.

A limitation which can also be traced back to the Kinect sensor is the detection of transparent surfaces. As explained in 4.2 the champagne glass used in the evaluation had to be coated with white paint in order to get meaningful depth data.

4.3.3 Limitations of the Performed Experiments

For all the experiments conducted, the HSR is driving a full circle around each object setup and collects detailed sensor information from all viewing angles. This is, however, not always possible when deploying a robot in real live scenarios. By always completing the full cycle the object is not only viewed from all possible sides, but the loop closing of the employed SLAM can improve the result.

Nevertheless, nothing of the data produced in this project suggests that the proposed method, to evaluate the a trajectory by reconstructing a 3D model, should not work when the circle around the object setup is not completed. If only a few viewing angles of the object of interest are collected, a reconstruction can still be executed. Since the evaluation used in the experiments compares the reconstructed object against the reference, the missing 3D points of the PCD should not hamper the evaluation. However, there is the possibility that through those missing points the object registration becomes more difficult. Furthermore, if the reconstructed models should be used for object grasping, the number of possible solutions is limited in contrast to using the reconstruction of the whole object.

For SLAM to work as expected in the proposed setup, the scene has to be static. Dynamic objects inhibt the quality of the estimated trajectory drastically. This can for example be observed in Figure 4.7d. In this test case, a person is walking through the field of view for a brief moment, and the quality of the estimated trajectory already decreases substantially. This can, however, be countered by employing additional sensors for the evaluation. The same sensor data (with the person walking through the scene) was used for a RTAB-Map run with additional LiDAR data, and the quality increased to the same level as the static test cases shown in figure 4.7.

4.3.4 Engineering Constraints

Finally, some remarks about the general implementation of this project: While SLAM is a crucial algorithm for various mobile robot tasks, it suffers from a lack of compatibility and documentation. Although a large number of SLAM and VSLAM implementations are available, many of them are difficult to build and use. However, if ROS packages are available, the difficulty of employing them is significantly reduced. Therefore, the SLAM community should concentrate on improving documentation and software version compatibility to facilitate further research and development in the field of mobile robotics and navigation.

In summary, the experiments conducted in this thesis met the expectations in general. The evaluation of the pose estimation using SLAM with RGB-D sensor data and 3D object reconstruction was successful, with most of the distance errors originating from the depth noise of the Kinect. The reconstructed objects were accurate enough to be used as input for a grasp pose detector. However, challenges arose when dealing with transparent objects or dynamic scenes, which remained unresolved.

CHAPTER 5

Conclusion

The goal of thesis was to evaluate visual SLAM algorithms on the mobile robot platform *Toyota HSR* and test the overall limits as well as the robustness against camera position and object configurations. Since the procurement of a ground truth for each robot trajectory requires an external room-level sensor array, it is not only complicated, but also very expensive. Therefore, the evaluation of VSLAM is done through the notion of 3D object reconstruction. In doing so the SLAM poses of each trajectory are used to create a 3D object model with the use of a TSDF. Those reconstructed objects are then compared against a high quality 3D object model from which the quality of the SLAM trajectory can be inferred. For this method a lot of ground truth is available (e.g. YCB dataset [CSB+17]) and therefore, it is simple and cheap in contrast to generating a ground truth of the trajectory itself.

The accompanying project integrates such SLAM and SfM techniques on the HSR. For the data collection, RGB as well as depth images were taken using the HSR's RGB-D camera. Furthermore, LiDAR data was recorded for a comprehensive comparison. Four different algorithm configurations were used to estimate the trajectory on several different test cases using a LiDAR only SLAM, a visual only SLAM, a combination of both as well as a SfM method. An object reconstruction pipeline was implemented to generate said 3D object reconstructions from the collected image data and the SLAM poses. The quality of those reconstructed objects was then compared against high quality models by registering them with the according ground truth object and issuing a point-to-point distance comparison.

The results of this thesis have indeed shown that it is possible to infer the quality of the robots trajectory by evaluating the corresponding reconstructed 3D object models. More sensor information with additional LiDAR data or more feature-rich objects also resulted in a better reconstruction quality. Therefore, it can be assumed that the pose estimate has been more precise which leads to an overall better trajectory.

In general $\sim 80\%$ of the points of each reconstructed point clouds have an error below $5mm$ using visual SLAM, going down to $\sim 70\%$ when only employing a LiDAR sensor (see Table 4.1). When increasing the number of objects or only using big and feature-rich objects this number even increases. In terms of robustness the results speak in favour of VSLAM. The camera position in z-direction as well as the number of objects influence the results only marginally. The fluctuations in 3D object quality mostly stem from differences in rosbag data quality. State-of-the-art visual SLAM methods can be used for automated 3D modelling for most applications. If, for example, the objects should be used in a grasping pipeline for everyday rigid objects like cracker boxes or mustard bottles, errors in the vicinity of $5mm$ are certainly low enough to employ such techniques. This can be observed in Figures 4.12 and 4.13. For high precision applications the error of the chosen algorithms together with the limitations of the depth sensor available on the HSR may be too large.

The presented method to evaluate estimated robot trajectories with the method of 3D reconstruction has been proven to work. Furthermore, it has been shown that the SLAMs on the HSR are robust in terms of vertical camera position, different object types, and object setups. This method can be used to accelerate the research in the area of personal robotics. The introduction of a fast and simple way to effectively evaluate trajectory estimations without the need of expensive external sensor can be used, after a support robot is deployed.

5.1 Outlook

Of course the work for researchers in this field is far from completed. Beside the further development of additional SLAM and SfM methods, additional research platforms are needed to test different algorithms for dissimilar application areas. Furthermore, the method presented here is limited by the constraints of the depth sensor used, as explained in Section 4.3.2. The low resolution of the Xtion PRO LIVE employed on the HSR limits its ability to perform certain high precision tasks, which, in turn, limits the accuracy of the trajectory quality estimation. Of course, RGB-D cameras are not the only possible sensors for SLAM systems as stereo or even monocular cameras can be used. Depth estimated with stereo cameras is less accurate than RGB-D, especially when the object distance increases. Stereo depth estimation methods can also be used with monocular cameras based on consecutive camera shots. A similar approach is already used in this thesis with SfM which introduces a scaling problem explained in Section 3.1.2. Increasing the accuracy of such depth estimation methods would be a good way to increase the applicability of the proposed trajectory quality estimation. Finally, none of the depth estimation methods are applicable to objects with non-lambertian surfaces. This includes transparent and reflective objects, for which obtaining depth is very challenging. Methods to estimate depth for such objects would enable the evaluation of trajectories for scenes containing only such objects.

Further work can certainly extend the results of this thesis by adding additional SLAM methods, test cases in different environments, additional sensor information and different object configurations. Another particularly interesting result would be to employ different cameras and compare those to results of the *Xtion PRO LIVE*. The results acquired in the experiments conducted suggest that further reconstruction quality is limited by the RGB-D camera's depth resolution as well as the depth noise. A direct comparison of the evaluation methods proposed in this thesis against a direct trajectory comparison using ground truth trajectories would also further deepen the knowledge gained by this thesis.

List of Figures

List of Tables

Bibliography

[AH16] Ali Ismail Awad and Mahmoud Hassaballah. Image feature detectors and descriptors. *Studies in Computational Intelligence. Springer International Publishing, Cham*, 2016.

[BB95] Steven S. Beauchemin and John L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466, 1995.

[BNG$^+$16] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[CCF$^+$23] Filippo Cavallo, John-John Cabibihan, Laura Fiorini, Alessandra Sorrentino, Hongsheng He, Xiaorui Liu, Yoshio Matsumoto, and Shuzhi Sam Ge. *Social Robotics: 14th International Conference, ICSR 2022, Florence, Italy, December 13–16, 2022, Proceedings, Part II*, volume 13818. Springer Nature, 2023.

[CER$^+$21] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[CL96] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[CSB+17] Berk Calli, Arjun Singh, James Bruce, Aaron Walsman, Kurt Konolige, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Yale-cmu-berkeley dataset for robotic manipulation research. *The International Journal of Robotics Research*, 36(3):261–268, 2017.

[Cso97] Michael Csorba. *Simultaneous localisation and map building*. PhD thesis, University of Oxford Oxford, 1997.

[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[EZW+06] Mark Everingham, Andrew Zisserman, Christopher KI Williams, Luc Van Gool, Moray Allan, Christopher M Bishop, Olivier Chapelle, Navneet Dalal, Thomas Deselaers, Gyuri Dorkó, et al. The 2005 pascal visual object classes challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment: First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, pages 117–176. Springer, 2006.

[Faw06] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. ROC Analysis in Pattern Recognition.

[FB81] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[FLB+23] Tim Field, Jeremy Leibs, James Bowman, Dirk Thomas, and Perron Jacob. rosbag. `http://wiki.ros.org/Bags`, Febuary 2023.

[Foo13] Tully Foote. tf: The transform library. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6, 2013.

[GZ21] Xiang Gao and Tao Zhang. *Introduction to Visual SLAM : From Theory to Practice*. Springer Singapore Imprint: Springer, Singapore, 1st ed. 2021. edition, 2021.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[HS$^+$88] Chris Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[IFR23] IFR. International federation of robotics. `https://ifr.org/`, 2023.

[Int21] IntRoLab. List of open source slam projects. `https://github.com/introlab/rtabmap/wiki/List-of-Open-Source-SLAM-projects`, July 2021.

[KMvSK11] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[Kre08] Verena Elisabeth Kremer. Quaternions and slerp. *Embots. dfki. de/doc/seminar_ca/Kremer_Quaternions. pdf*, 2008.

[LCW$^+$12] Jie Li, Lei Cheng, Huaiyu Wu, Ling Xiong, and Dongmei Wang. An overview of the simultaneous localization and mapping on mobile robot. In *2012 Proceedings of International Conference on Modelling, Identification and Control*, pages 358–364, 2012.

[Lin94] Tony Lindeberg. Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, 21:224–270, 09 1994.

[LK81] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI'81: 7th international joint conference on Artificial intelligence*, volume 2, pages 674–679, 1981.

[LM19] Mathieu Labbé and François Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[LWH10] Minjie Li, Liqiang Wang, and Ying Hao. Image matching based on sift features and kd-tree. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 4, pages V4–218–V4–222, 2010.

[MAMT15] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[mar22] markusltnr. gpd. `https://github.com/markusltnr/gpd`, June 2022.

[MAT17] Raúl Mur-Artal and Juan D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[MBMM$^+$22] Andréa Macario Barros, Maugan Michel, Yoann Moline, Gwenolé Corre, and Frédérick Carrel. A comprehensive survey of visual slam algorithms. *Robotics*, 11(1), 2022.

[ML14] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[Mur99] Kevin P Murphy. Bayesian map learning in dynamic environments. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[NIL12] Chuong V. Nguyen, Shahram Izadi, and David Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 524–530, 2012.

[NNB04] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I, 2004.

[QCG$^+$09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[RACC20] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1689–1696, 2020.

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 430–443. Springer, 2006.

[Rod21] Julian Rodriguez. A comparison of an rgb-d cameras performance and a stereo camera in relation to object recognition and spatial position determination. *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, 20:16, 01 2021.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[SC86] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.

[Sch23] r Johannes Schönberge. Colmap. `https://colmap.github.io/`, 2023.

[SEE$^+$12] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, 2012.

[SF11] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics & Automation Magazine*, 18(4):80–92, 2011.

[SF16] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[SRD$^+$21] Myriam Servieres, Valerie Renaudin, Alexis Dupuis, Nicolas Antigny, and Stelios Potirakis. Visual and visual-inertial slam: State of the art, classification, and experimental benchmarking. *Hindawi Journal of Sensors*, 2021.

[SSE$^+$17] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[Sze22] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[SZPF16] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[TPGSP17] Andreas Ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. Grasp pose detection in point clouds. *The International Journal of Robotics Research*, 36(13-14):1455–1473, 2017.

[vt22] v4r tuwien. Table plane extractor. `https://github.com/v4r-tuwien/table_plane_extractor`, June 2022.

[WAHW14] Diana Werner, Ayoub Al-Hamadi, and Philipp Werner. Truncated signed distance function: Experiments on voxel size. In Aurélio Campilho and Mohamed Kamel, editors, *Image Analysis and Recognition*, pages 357–364, Cham, 2014. Springer International Publishing.

[YNK$^+$18] Takashi Yamamoto, Tamaki Nishino, Hideki Kajima, Mitsunori Ohta, and Koichi Ikeda. Human support robot (hsr). In *ACM SIGGRAPH 2018 emerging technologies*, pages 1–2. 2018.

[ZPK18] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[ZZT21] Shishun Zhang, Longyu Zheng, and Wenbing Tao. Survey and evaluation of rgb-d slam. *IEEE Access*, 9:21367–21387, 2021.

[Ča20] Karel Čapek. *R.U.R (Rossum's Universal Robots)*. 1920.