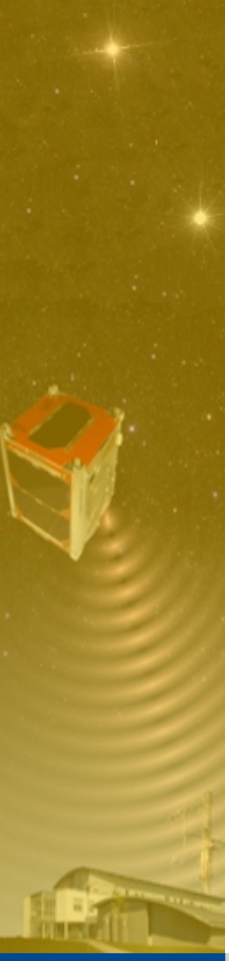
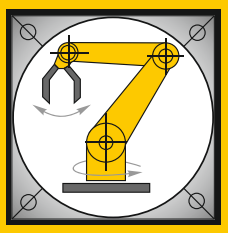


Institut für Informatik
Lehrstuhl für Robotik und Telematik
Prof. Dr. K. Schilling
Prof. Dr. A. Nüchter



Würzburger Forschungsberichte
in Robotik und Telematik

Uni Wuerzburg Research Notes
in Robotics and Telematics

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Band 16

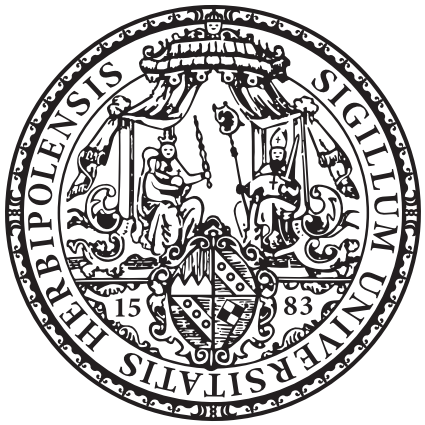
Rainer Koch

Sensor Fusion for Presice Mapping of Transparent and Specular Reflective Objects

Doctoral Thesis / *Dissertation*
for the doctoral degree / *zur Erlangung des Doktorgrads*
Doctor rerum naturalium (Dr. rer. nat.)

Sensor Fusion for Precise Mapping of Transparent and Specular Reflective Objects

Sensorfusion zur präzisen Kartierung von transparenten und reflektierender Objekten



Submitted by / *Vorgelegt von*
Rainer Koch
from / aus
Dettelbach, Germany
Würzburg, June 2018

Julius-Maximilians-Universität Würzburg
Graduate School of Science and Technology

Submitted on / *Eingereicht am:* 12^{th} *January* 2018

Day of thesis defense / *Tag des Promotionskolloquiums:* 16^{th} *May* 2018

Members of thesis committee / *Mitglieder des Promotionskomitees*

Chairperson / Vorsitz: Prof. Dr. Reiner Kolla

1. Reviewer and Examiner / 1. Gutachter und Prüfer: Prof. Dr. Stefan May

2. Reviewer and Examiner / 2. Gutachter und Prüfer: Prof. Dr. Andreas Nüchter

3. Examiner / 3. Prüfer: Prof. Dr. Sergio Montenegro

Additional Examiners / Weitere Prüfer: none

Affidavit

I hereby confirm that my thesis entitled "Sensor fusion for precise mapping of reflective objects" is the result of my own work. I did not receive any help or support from commercial consultants. All sources and / or materials applied are listed and specified in the thesis.

Furthermore, I confirm that this thesis has not yet been submitted as part of another examination process neither in identical nor in similar form.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, die Dissertation "Sensor fusion zur präzisen Kartierung reflektierender Objekte" eigenständig, d.h. insbesondere selbständig und ohne Hilfe eines kommerziellen Promotionsberaters, angefertigt und keine anderen als die von mir angegebenen Quellen und Hilfsmittel verwendet zu haben.

Ich erkläre außerdem, dass die Dissertation weder in gleicher noch in ähnlicher Form bereits in einem anderen Prüfungsverfahren vorgelegen hat.

Zusammenfassung

Fast schon wöchentlich füllen Meldungen über Erdbeben, Wirbelstürme, Tsunamis oder Waldbrände die Nachrichten. Es ist hart anzusehen, aber noch viel härter trifft es die Rettungskräfte, welche dorthin zum Einsatz gerufen werden. Diese müssen gut trainiert sein, um sich schnell einen Überblick verschaffen zu können und um den zerstörten Bereich nach Opfern zu durchsuchen. Zeit ist hier ein seltenes Gut, denn die Überlebenschancen sinken, je länger es dauert bis Hilfe eintrifft. Für eine effektive Teamkoordination werden alle Informationen in der Einsatzzentrale gesammelt. In Trupps werden die zerstörten Gebäude nach Opfern durchsucht und alle Hohlräume inspiziert. Dabei können die Helfer oft nicht darauf vertrauen, dass die Gebäude stabil sind und nicht noch vollständig kollabieren. Hier sind Rettungsroboter eine willkommene Hilfe. Sie sind ersetzbar und können für gefährliche Aufgaben verwendet werden. Dies macht die Arbeit der Rettungstrupps sicherer. Allerdings gibt es solche Roboter noch nicht von der Stange.

Sie müssten gewisse Anforderungen erfüllen, dass sie in solchen Szenarien einsetztbar sind. Neben Ansprüchen an die Mechanik, müsste eine 3D-Karte des Einsatzgebietes erstellt werden. Diese ist Grundlage für eine erfolgreiche Navigation (durch unebenes Terrain), sowie zur Beeinflussung der Umgebung (z.B. Tür öffnen). Die Umgebungserfassung wird über Sensoren am Roboter durchgeführt. Heutzutage werden bevorzugt Laserscanner dafür verwendet, da sie präzise Messdaten liefern und über einen großen Messbereich verfügen. Unglücklicherweise werden Messdaten durch transparente (z.B. Glas, transparenter Kunststoff) und reflektierende Objekte (z.B. Spiegel, glänzendes Metall) verfälscht. Eine Vorbehandlung der Umgebung (z.B. Abdecken der Flächen), um diese Einflüsse zu verhindern, ist verständlicherweise nicht möglich. Zusätzliche Sensoren zu verwenden birgt zwar Vorteile, aber auch Nachteile.

Das Problem dieser Objekte liegt darin, dass sie nur teilweise sichtbar sind. Dies ist abhängig vom Einfallwinkel des Laserstrahls auf die Oberfläche und vom Typ des Objektes. Dementsprechend könnnen die Messwerte bei transparenten Flächen von der Oberfläche oder von Objekten dahinter resultieren. Im Gegensatz dazu können die Messwerte bei reflektierenden Oberflächen von der Oberfläche selbst oder von einem gespiegelten Objekt resultieren. Gespiegelte Objekte werden dabei hinter der reflektierenden Objerfläche dargestellt, was falsch ist. Um eine präzise Kartierung zu erlangen, müssen die Oberflächen zuverlässig eingetragen werden. Andernfalls würde der Roboter in diese navigieren und kollidieren. Weiterhin sollten Punkte hinter der Oberfläche abhängig von der Oberfläche behandelt werden. Bei einer transparenten Oberfläche müssen die Punkte in die Karte eingetragen werden, weil sie ein reelles Objekt darstellen. Im Gegensatz dazu, müssen bei einer reflektierenden Oberfläche die Messdaten dahinter gelöscht werden. Dafür ist eine Unterscheidung der Objekte zwingend. Diese Anforderungen erfüllen die momentan verfügbaren Algorithmen jedoch nicht.

Aus diesem Grund befasst sich folgende Doktorarbeit mit der Problematik der Erkennung und Identifizierung transparenter und spiegelnder Objekte, sowie deren Einflüsse. Um dem Leser einen Einstieg zu geben, beschreiben die ersten Kapitel: den theoretischen Hindergrund

bezüglich des Verhaltens von Licht; Sensorsysteme für die Distanzmessung; Kartierungsalgorithmen, welche in dieser Arbeit verwendet wurden; und den Stand der Technik bezüglich der Erkennung von transparenten und spiegelndend Objekten. Danach wird der Reflection-Identification-Algorithmus, welcher Basis dieser Arbeit ist, präsentiert. Hier wird eine 2D und eine 3D Implementierung beschrieben. Beide sind als ROS-Knoten verfügbar. Das anschließende Kapitel diskutiert Experimente, welche die Anwendbarkeit und Zuverlässigkeit des Algorithmus verifizieren. Für den 2D-Fall sind ein Vor- und ein Nachfilter-Modul notwendig. Nur mittels der Nachfilterung ist eine Klassifizierung der Objekte möglich. Im Gegensatz kann im 3D-Fall die Klassifizierung bereits mit der Vorfilterung erlangt werden. Dies beruht auf der höheren Anzahl an Messdaten. Weiterhin zeigt dieses Kapitel beispielhaft eine Adaptierung des TSD-SLAM Algorithmus, so dass der Roboter auf einer aktualisierten Karte navigieren kann. Dies erspart die Erstellung von zwei unabhängigen Karten und eine anschließende Fusionierung. Im letzten Kapitel werden die Ergebnisse der Arbeit zusammengefasst und ein Ausblick mit Anregungen zur Weiterarbeit gegeben.

Abstract

Almost once a week broadcasts about earthquakes, hurricanes, tsunamis, or forest fires are filling the news. While oneself feels it is hard to watch such news, it is even harder for rescue troops to enter these areas. They need some skills to get a quick overview of the devastated area and find victims. Time is ticking, since the chance for survival shrinks the longer it takes till help is available. To coordinate the teams efficiently, all information needs to be collected at the command center. Therefore, teams investigate the destroyed houses and hollow spaces for victims. Doing so, they never can be sure that the building will not fully collapse while they are inside. Here, rescue robots are welcome helpers, as they are replaceable and make work more secure. Unfortunately, rescue robots are not usable off-the-shelf, yet.

There is no doubt, that such a robot has to fulfil essential requirements to successfully accomplish a rescue mission. Apart from the mechanical requirements it has to be able to build a 3D map of the environment. This is essential to navigate through rough terrain and fulfil manipulation tasks (e.g. open doors). To build a map and gather environmental information, robots are equipped with multiple sensors. Since laser scanners produce precise measurements and support a wide scanning range, they are common visual sensors utilized for mapping. Unfortunately, they produce erroneous measurements when scanning transparent (e.g. glass, transparent plastic) or specular reflective objects (e.g. mirror, shiny metal). It is understood that such objects can be everywhere and a pre-manipulation to prevent their influences is impossible. Using additional sensors also bear risks.

The problem is that these objects are occasionally visible, based on the incident angle of the laser beam, the surface, and the type of object. Hence, for transparent objects, measurements might result from the object surface or objects behind it. For specular reflective objects, measurements might result from the object surface or a mirrored object. These mirrored objects are illustrated behind the surface which is wrong. To obtain a precise map, the surfaces need to be recognised and mapped reliably. Otherwise, the robot navigates into it and crashes. Further, points behind the surface should be identified and treated based on the object type. Points behind a transparent surface should remain as they represent real objects. In contrast, points appending behind a specular reflective surface should be erased. To do so, the object type needs to be classified. Unfortunately, none of the current approaches is capable to fulfil these requirements.

Therefore, the following thesis addresses this problem to detect transparent and specular reflective objects and to identify their influences. To give the reader a start up, the first chapters describe: the theoretical background concerning propagation of light; sensor systems applied for range measurements; mapping approaches used in this work; and the state of the art concerning detection and identification of transparent and specular reflective objects. Afterwards, the Reflection-Identification-Approach, which is the core of subject thesis is presented. It describes a 2D and a 3D implementation to detect and classify such objects. Both are available as ROS-nodes. In the next chapter, various experiments demonstrate the applicability and reliability of these nodes. It proves that transparent and specular reflective objects can be detected and classified. Therefore, a Pre- and Post-Filter-module is required in 2D. In 3D, classification is possible solely with the Pre-Filter. This is due to the higher amount of measurements. An example shows that an updatable mapping module allows the robot navigation to rely on refined maps. Otherwise, two individual maps are build which require a fusion afterwards. Finally, the last chapter summarizes the results and proposes suggestions for future work.

Preface

Everybody is happy if someone holds the door open for you. In this case, it is understood as a simple act of politeness. In case of a car accident fewer and fewer people are ready to help; they rather take pictures and gape. In fact, helping people in need is an essential social duty. This is already embodied in our law (§ 323c StGB). Everyone is obliged to provide assistance as long as they do not put themselves at risk. A simple phone call can be enough when arriving at the scene of the car accident.

Nevertheless, more than one million voluntary fire fighters, in Germany alone, decided to do more than just meeting the "standard" responsibility. They spend their leisure time to train hard and risk their lives in order to be prepared for upcoming duties. No matter if during the day or in the middle of the night. I myself have been doing this for more than 20 years.

When receiving a message from an alarm annunciator in the middle of the night immediate action is required: I get dressed, hurry to the fire department, prepare for the mission, and drive to the place of duty. In the short period between the fire station and the location of the incident, our team needs to get ready and organized to provide help. Teams of two gather to act as a buddy team. Entering a place on fire demands full trust of each other, full concentration, as well as physical fitness. Every mistake is fatal and might result in deadly danger for us and for the victims. Top priority is to locate and to rescue them. In fact, this is not as easy as said, as people act strange when facing danger. People often try to find shelter under tables or in wardrobes. In case of fire, vision is often limited while fire rages. The fear to miss someone is an always recurring thought in the mind of a fire fighter. Danger rises if there is no information about the number of victims which is the case most of the time.

To watch the news concerning such disasters is hard for all of us. Even worse are broadcasts about hurricanes, tsunamis, forest fire, or earthquakes since the impact is worse. Also, one has to keep in mind that such tragedies can hit us too. How do I act in such a scenario, especially when being in command and responsible for a team. Getting a quick overview is essential to make the right decision. As a fire fighter, but also as a scientist, I often ask myself what knowledge and equipment will help me in such a situation. Mobile robots surely provide potential to make our lives easier and saver. Having a team of autonomous robots to search a disaster zone for victims and also deliver a reliable map would be a great help. It saves time – which is most critical for victims. Further, robots make our work saver since objects threaten to collapse and they are replaceable. Unfortunately, these robots are not yet available off-the-shelf.

I find it a special challenge and pleasure to be involved in this topic, job-related too. Being occupied with a PhD thesis concerning sensor fusion for precise mapping opens a unique opportunity to support my rescue colleagues all over the world. As mentioned, a reliable map is most important to keep the overview. Many approaches already demonstrate mapping for mobile robots. It is a pity that maps still suffer from erroneous measurements even when they are build up by expensive and precise sensors such as laser scanners.

Here, transparent and specular reflective objects such as glass objects, shiny metal surfaces, and mirrors result in unwanted influences. Besides, they are occasionally visible. This creates multiple problems. First, the surface is not always visible which can cause a crash of the robot. Most of us already went through this experience when running into a closed glass door as we did not see the glass. Second, objects behind the transparent or specular reflective object are measured. Based on the surface, these objects result from a true object (behind glass) or a reflection (behind a mirror). Even for us, it is often difficult to distinguish between these two effects. That is why I find it an interesting challenge to concentrate my studies on these effects and work on a solution to handle them.

In the below thesis I present experiences I made as well as my implemented Reflection-Identification-Approach to recognise transparent and specular reflective objects with their influences. The experiments demonstrate that these objects can be classified resulting in a refined map. While the surface as well as the points behind a transparent surface remain in the map, reflections are erased. This is achieved by multiple investigation methods to verify the object type. It is shown that reflections can be back-projected and then used for mapping as well. The first chapters lead the reader to the topic. First, the theoretical background is covered. Then, sensor systems and mapping algorithms are presented. Afterwards, the state-of-the-art covering transparent and specular reflective objects is presented and discussed. At the end of the thesis, the results summarized and suggestions for future work are presented.

Finally, I want to thank my supervisors Prof. Dr. Andreas Nüchter, Prof. Dr.-Ing. Sergio Montenegro, and Prof. Dr. Stefan May who supported and guided me during my PhD thesis. I also would like to thank my colleges for the good cooperation.

I also want to thank Ursula Pfefferlein and Cassandra Christ for their willingness in proof-reading my publications.

Last but not least, many thanks to my wife Wanyi for her appreciation and patience during the last few years.

Contents

Chapter 1

Introduction

Japan is still suffering from the huge tsunami and the resulting atomic disaster in Fukushima in 2011. If people think disasters are only an occasional incident they are wrong. The Atlantic hurricane season just started and the Caribbean island as well as America are already suffering from huge devastations and flooding. While hurricane Harvey hit mainly Texas and Louisiana in the end of August, hurricane Irma devastated Florida in beginning of September. In compare, Mexico suffered from a huge earthquake. But also Europe suffers from extreme weather conditions - huge heat in the south (e.g. Spain, Portugal, Italy) and plenty of rain in the north (e.g. Germany). Besides, the regions of Lazio, Umbria, and March suffer from earthquakes again and again. Since a huge earthquake in August 2016 until now there have been more than 49.000 earthquakes [Wikipedia, 2017] in this region. There are multiple strong ones, e.g. in October 2016, January 2017, July 2017, and August 2017 and it is not over yet. Picture 1.1 illustrates disaster area of the city of Amatrice after the earthquake on 18th January 2017.

Figure 1.1: Disaster area of Amatrice in Italy after the earthquake on 18th January 2017. [20minuten.ch, 2017]

Rescue teams have a typical procedure to search such disaster areas for victims. It is understood that a situation like this brings the rescue teams to their limits. There is almost no information how many people are still in there, where they are located, or what their injuries are. Every help is welcome since there are not enough helpers and time is ticking. The longer it takes to find the victim, the lower is the chance for him/her to survive.

Just reducing the scenario to a single house makes it obvious that rescue troops are facing hard conditions. In troops of two people they explore and search the area part by part. Each room of a building has to be inspected. Therefore, the troop is exploring the room by crawling on the ground counterclockwise to locate victims, animals, or other objects of interest (e.g. gas bottles, explosive chemicals, etc.). Typical places for victims to find shelter are inside of cabinets, under tables, under beds, or behind doors. These places have to be found by the rescue teams and inspected, even when the vision is limited by smoke. After a room has been searched, it is marked by a sign on the door to indicate its status (not checked, clean, under checking). It is understood that the rescue teams operate under hazardous circumstances and risk their lives all the time. Since robots are replaceable they can make work safer and more secure for men. Further, it is possible to equip them with various sensor units which help to improve the mission. Unfortunately, such robots are not available off-the-shelf.

A robot has to fulfil essential requirements to successfully accomplish a rescue mission. First, it needs to be able to navigate through rough terrain, get into buildings, climb stairs. It must use its manipulator to open doors, remove small parts, or bring equipment. For this reason, a reliable and durable chassis with actors (wheels, chains, manipulator, clamp, etc.) is required. Compared to a human, the chassis with its actors can be understood as the body and muscles of the robot.

Second, the robot needs to investigate, build a map of the disaster area, communicate with the operator, search and identify victims, as well as other objects of interest. This is accomplished by a sophisticated software system with reliable sensor units. While sensor units of the robot represent the sensing of a human body (eyes, ears, nose, etc.), the software system can be called its brain. Typical visual sensors for robots are RGB-cameras, laser scanner, ultrasonic sensors, thermal cameras, stereo cameras, and Time-of-Flight-cameras (ToF). Besides, there are sensors like a microphone to get a voice feedback, Global-Positioning-System sensor (GPS) to get a global location, Inertial-Measurement-Unit (IMU) to receive a feedback of movement, etc..

机器人上安装的主要的传感器

As previously mentioned, mapping is a main task of the robot since a map is required to get an overview of the area and localize victims, objects and the robot itself. A lot of research has been done in mapping, but there are still limitations. In the past, at competitions like RoboCup, mapping was performed in 2D [SSRR et al., 2013]. Cartography in 3D opens new opportunities. It gives a better view of the ambiance, it can be used to do obstacle avoidance [Andriluka et al., 2009; Holz et al., 2010]. Besides, it is necessary to navigate through uneven terrain as this is usually the case for disaster areas.

Maps are based on distance measurements but they are not always accurate. System based drawbacks as well as errors caused by external influences occur. Laser scanners are state-of-the-art sensors used for mapping as they allow a wide scanning range, precise measurements, and are applicable indoor and outdoor. Therefore, a mapping module delivers detailed and high resolution maps to the robot.

Difficulties in laser scanning result from transparent and specular reflective objects e.g. mirrors, windows, shiny metals which cause erroneous and dubious measurements. Figure 1.2 illustrates erroneous measurements (highlighted in red) caused by a mirror (highlighted in blue).

In case of specular reflective objects, e.g. at mirrors, the laser beams get reflected and rerouted to an object located in front of the mirror. Therefore, the returned measurements

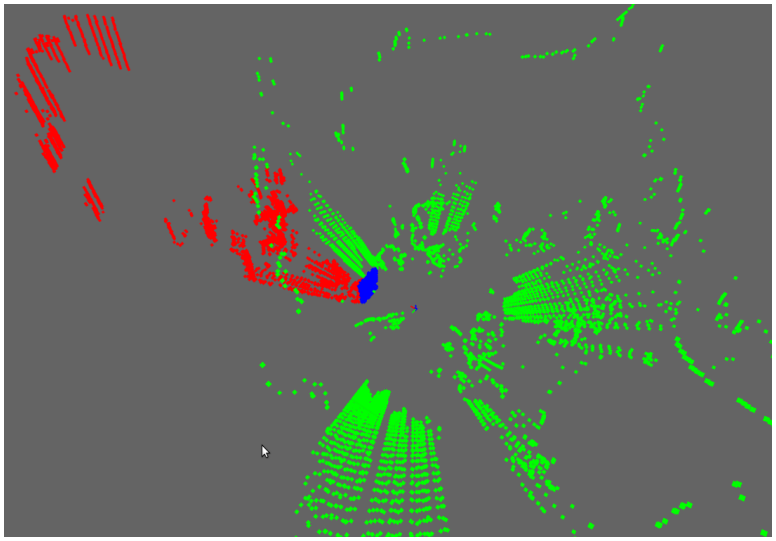


Figure 1.2: Robot facing a mirror. The resulting point cloud of the laser scanner shows the valid point (highlighted in green), the mirror (highlighted in blue), and the mirrored points (highlighted in red).

result from a mirrored object. That is why the "illustrated" location of the object is wrong. Besides, the mirror surface is not mapped or only occasionally mapped.

In case of transparent objects, e.g. at glass doors, the measurements of the laser beams partly result from the transparent surface and partly from the objects behind the surface, depending on the incident angle of the laser beam. It is understood that such erroneous measurements lead to a faulty map and therefore difficulties in navigation. No matter if the object surface is not seen at all or if it is seen occasionally it does not show up on the map. That is why the robot might manoeuvre into the object and crash.

Unfortunately, most environments include transparent or specular reflective objects like glass doors, windows, mirrors, or shiny metal surfaces. To prevent erroneous measurements from such objects in laser scans there are two commonly used techniques. The first technique is to customize the environment and cover these objects. This is unwanted, as it changes the "real" environment. Besides, it is not always possible, e.g. when operating in rescue scenario. Because of this many approaches employ the second technique – a sensor fusion. Here, a second sensor principle, like ultrasonic arrays, is fused with the laser scanner to respect these troublesome objects. Ultrasonic arrays are capable to detect transparent and specular reflective objects but they suffer from imprecise measurements and low measurement range. Besides, it is necessary to deal with two sensor units. This results in extra costs, an additional source of hardware failure, and requires calibration. Hence, this method is unwanted as well. This raises below questions which are basis for the following work:

- Is it possible to detect transparent and specular reflective objects and their influences solely with a laser scanner?
- How can a distinction be made between a transparent and a specular reflective object?
- If the influences can be recognised and characterised, is this possible on the fly, during the mission, or does it require a post-processing?

1.1 Structure of the Thesis

The thesis is structured as follows:

In the beginning, **Chapter 2** describes the theoretical background. It explains reflection and refraction of light as well as resulting intensity curves of light at different materials.

Chapter 3 outlines common sensors applied for 2D and 3D-mapping at mobile robotics. It outpoints their measurement principle and discusses advantages and drawbacks concerning transparent and specular reflective influences.

Chapter 4 gives an overview of Simultaneous-Localization-and-Mapping (SLAM) algorithms used in this thesis. It describes the changes made at Truncated-Signed-Distance-SLAM (TSD-SLAM) which was the preferred SLAM-method as it was implemented in our lab.

Chapter 5 analyses implementations of state-of-the-art approaches related to transparent and specular reflective influences in mapping. It discusses their advantages and drawbacks and points out the need of the thesis.

Chapter 6 depicts flow charts of the 2D and 3D-Mirror-Identifier-Approach and elucidates the implementations. To do so, it explains the main steps and the corresponding functions.

Chapter 7 describes the applied experiments which where made. In the first step, two experiments are carried out to develop intensity models applied to distinguish between transparent and specular reflective objects. In the second step, multiple experiments demonstrate the applicability and reliability of the Mirror-Identifier-Approach in 2D and 3D.

In the end, **Chapter 8** gives a résumé of the entire work. It concludes the results and applicability of the implemented approach to detect transparent and specular reflective objects and to distinguish between them. It also discusses drawbacks of the approach and makes suggestions for further work.

Chapter 2

Propagation of Light

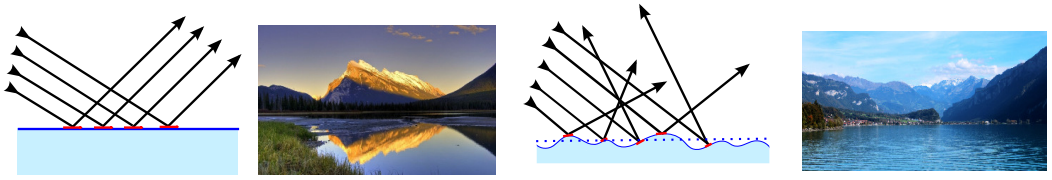
This chapter describes the theoretical background of light propagation. Light can be characterized at two levels: the lower level by the electromagnetic wave properties and the higher level by geometrical object properties. This work refers to the behaviour of a laser beam hitting a transparent or specular reflective object and therefore discusses reflection and refraction. So, only the higher level is required and pointed out in this chapter. Therefore, the first section deals with the reflection of light, while the second section concerns refraction of light.

It has to be mentioned that both effects always occur, mainly based on the surface quality (planarity and cleanness), the refraction index, and the incident angle. This is why, under certain circumstances, the surface of transparent and specular reflective objects can be seen, too.

The applied laser scanner, an Hokuyo UTM-30LX-EW, delivers distance measurements as well as intensity information of the returning laser beam. This is why Section 2.3 of this chapter deals with intensity characteristic of light.

2.1 Reflection of Light

Reflection occurs as soon as light (in this work a laser beam) hits an object surface. Also, an abrupt change in direction of the beam takes place at the surface of the two media (air and object surface). This effect can be investigated when looking at a lake, cf. Figure 2.1. When a light beam hits the surface of a lake (medium air and medium water). Based on the surface condition the light is reflected diffuse (Figure 2.1b) or specular (Figure 2.1a). A smooth water surface results specular reflective properties. All incoming light is reflected in the same direction. Since the light beams remain in parallel the image is not disturbed. The reason for that lies in the fact that the local surface (red surface line) orientation is the same as the global (blue surface line) Because of that, the hill occurs on the surface as a mirrored picture. In contrast, a rough water surface (e.g. caused by wind) impacts diffuse reflective properties. The light is reflected in all directions, based on the local surface orientation (red surface line) the individual light beam faces. The local surface orientations (red surface line) do not match the global surface orientation (blue dotted line) As a result, the hill cannot be identified clearly.



(a) At a specular reflection the hill is seen on the surface of the lake.

(b) At a diffuse reflection the hill cannot be seen on the surface of the lake. The local surface of each beam is marked by a red line while the global surface is marked by a blue dotted line.

Figure 2.1: Effect of diffuse and specular reflective surface and the resulting visible impact of a mountain on a lake. The local surface of each beam is marked by a red line while the global surface is marked by a blue dotted line.

The specular reflection is based on the law that the angle ϵ of the incoming light beam equals the angle of the outgoing light beam ϵ_r.

$$\epsilon = \epsilon_r \tag{2.1}$$

The angles span up between the surface normal and the incoming and outgoing laser beam, cf. Figure 2.2.

In mapping, specular reflective surfaces as mirrors, shiny metal, etc. disturb measurements since they cause unwanted redirection of the laser beam. This produces measurements of a "fake" object, further called mirrored object which results from an object located in front of the reflective surface and mirrored w.r.t. the reflective surface. Figure 2.2 illustrates this effect for a single data point $\vec{p}_{real}$ and its measured value $\vec{p}_{measure}$. The length from the laser scanner to the mirrored object $d_{measure}$ equals the sum of the length between the laser scanner to the surface $d_{surface}$ and the redirected length of the laser beam $d_{redirect}$.

$$d_{measure} = d_{surface} + d_{redirect} \tag{2.2}$$

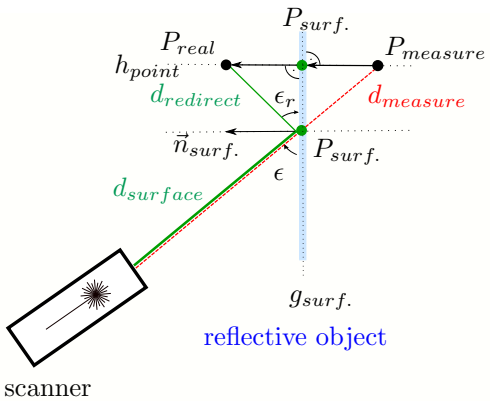


Figure 2.2: Reflection of an object point $\vec{p}_{real}$ on a reflective object surface and its resulting measurement point $\vec{p}_{measure}$ behind the reflective object surface.

Assuming that

- the distance $d_{surf.}$ and $d_{measure}$ result from the scanner
- the surface normal $\vec{n}_{surf.}$ can be determined by taking multiple points on the surface
- ϵ can be determined by the scanner location and the angle of the outgoing laser beam

then the point $\vec{p}_{real}$ can be recalculated by

$$\vec{p}_{real} = \vec{p}_{surf.} + \mathbf{R} \cdot \overrightarrow{P_{measure}P_{surf.}} \tag{2.3}$$

with $\overrightarrow{P_{measure}P_{surf.}}$ is the vector between the measured point $P_{measure}$ and the perpendicular point on the reflective surface $P_{perpen.}$ and $\mathbf{R}$ is the rotation matrix

$$\mathbf{R} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

This leads to difficulties for spherical reflective surfaces, cf. Figure 2.3. For concave shaped mirrors (blue line) the mirror objects get scaled down and illustrated in front of the mirror surface. For convex shaped mirrors (red dotted line) the object gets scaled down as well, but illustrated behind the mirror surface.

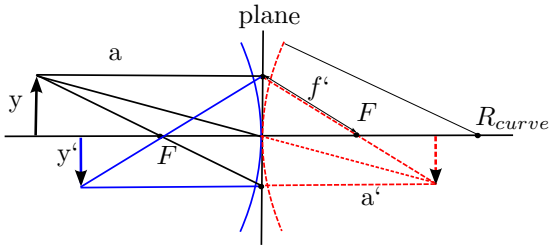


Figure 2.3: Reflection of an object on a concave/convex shaped mirror.

Hammer et al. [2012] uses the curvature radius of the mirror R_{curve} to determine the focal length $f‘ = {}^{R_{curve}}/_{2}$ between the surface plane y and the focal point F.

$$-\frac{1}{a} + \frac{1}{a‘} = \frac{1}{f‘} \tag{2.4}$$

$$\beta‘ = \frac{y‘}{y} = \frac{a‘}{a} \tag{2.5}$$

2.2 Refraction of Light

In compare to reflections, refraction of light occurs as soon as a light beam travels at an angle α_{in} into a substance with a different refractive index n (optical density), illustrated in Figure 2.4. The refraction index describes how light propagates through this medium.

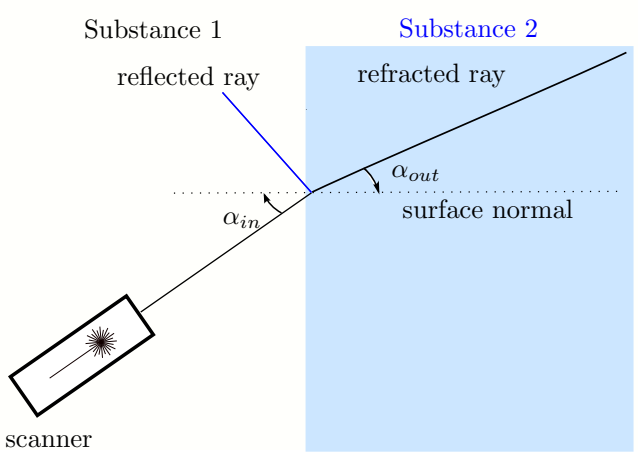


Figure 2.4: Refraction of light

As soon as the light beam hits the surface of Substance 2 it gets refracted by the angle α_{out} and continues its path through the surface. The relationship between the incoming beam and the outgoing beam is described by Nobach [2012] as

$$\frac{\sin(\alpha_{in})}{\sin(\alpha_{out})} = \frac{c_{in}}{c_{out}} = \frac{n_{out}}{n_{in}} \tag{2.6}$$

with

$$n_* = \frac{c_0}{c_*}$$

while α_{in} is the angle between the surface normal and the incoming light beam, α_{out} is the angle between the surface normal and the outgoing light beam, c_0 is the speed of light in vacuum, c_* the speed of light in material $*$, and n_* is the refraction index of material $*$. A greater n results from an optically denser medium while an optically thinner medium has a smaller n.

As mentioned before, reflection always takes place, even for transparent objects. The amount of each effect depends on the surface quality, the incident angle, and the refraction index of both materials. A special case occurs if

$$\sin(\epsilon_g) = \frac{n_2}{n_1}$$

which results to the angle of total reflection ϵ_g.

This makes it challenging to measure such surfaces since measurements can result from an object behind (in case of transparent behaviour) or an object in front (in case of mirror behaviour).

Talking about a window in a mapping scenario, it is understood that the light beam underlies two refractions, cf. Figure 2.5. As a result, the original beam is shifted by $d_{displace}$. With the assumption that $(d_{surface} + d_{object}) \gg d_{glass}$ it can be assumed that $d_{displace} \ll d_{measure}$. Hence, $d_{displace}$ can be ignored for mapping.

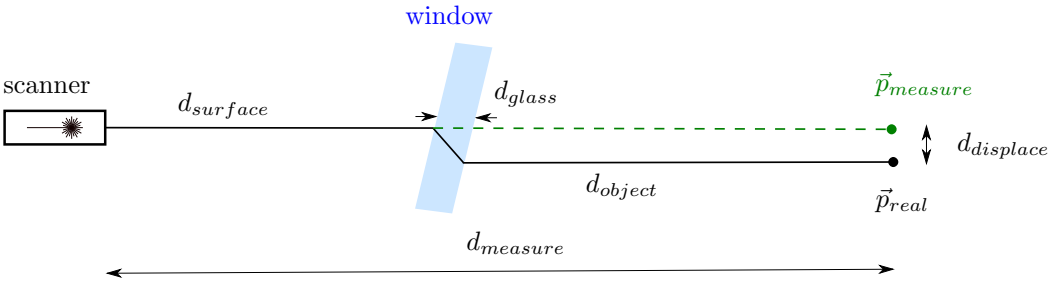
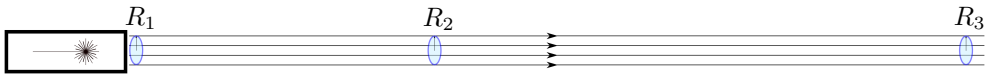


Figure 2.5: Double refraction of light at a window and the resulting displacement $d_{displace}$ of the object point.

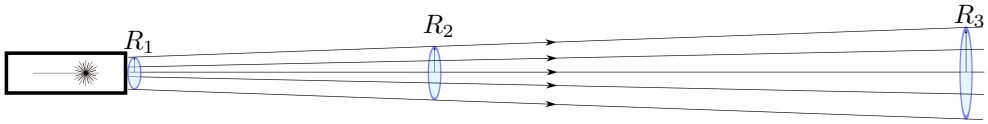
2.3 Intensity Characteristics of Light

The intensity values are an indication of the amount of light returning form the scanned object. They depend on the distance, the surface properties (specular reflective or diffuse reflective), incident angle, the permeability of the medium the laser beam is running through (refraction index, temperature, pressure, etc.). For the following we assume that the permeability of the medium can be neglected. Hence, the remaining three influences remain (distance dependency, surface dependency, and angle dependency) and are described in detail.

Surface and distance dependency:
A laser is a focused light beam with a bundle of parallel light rays (cf. Figure 2.6a, but it suffers from an widening effect over the distance (cf. Figure 2.6b).



(a) Ideal laser beam which consists of parallel light. $\Rightarrow R_1 = R_2 = R_3$



(b) Real laser beam which consists of an opening angle. $\Rightarrow R_1 < R_2 < R_3$

Figure 2.6: Ideal laser beam vs. real laser beam which suffer from a widening effect.

Due to this widening, the intensity covers a bigger area for a point further away. So, for the returning intensity I_{return} follows:

$$I_{return} \propto \frac{1}{d^2} \tag{2.7}$$

with d being the distance between the object surface and the laser scanner. That is why the returned maximal intensity shrinks with a greater distance.

In addition, the surface characteristic (reflective, diffuse, transparent) impacts the returning intensity amount. For diffuse reflective objects a huge amount of the light is reflected all over, cf. Figure 2.1b. Therefore, only a small amount returns to the scanner. In compare, most of the light, for specular reflective objects, returns to the scanner. This results in a dependency of the returning intensity on the object surface.

Figure 2.7 illustrates the intensity values at an incident angle $\alpha_{in} = 0°$ for various surfaces. It was recorded during the Experiment 7.1 which will be explained in detail in Chapter 7. It has to be mentioned that the measurement for the mirror at a distance $d = 0.5\ m$ can be assumed as incorrect.

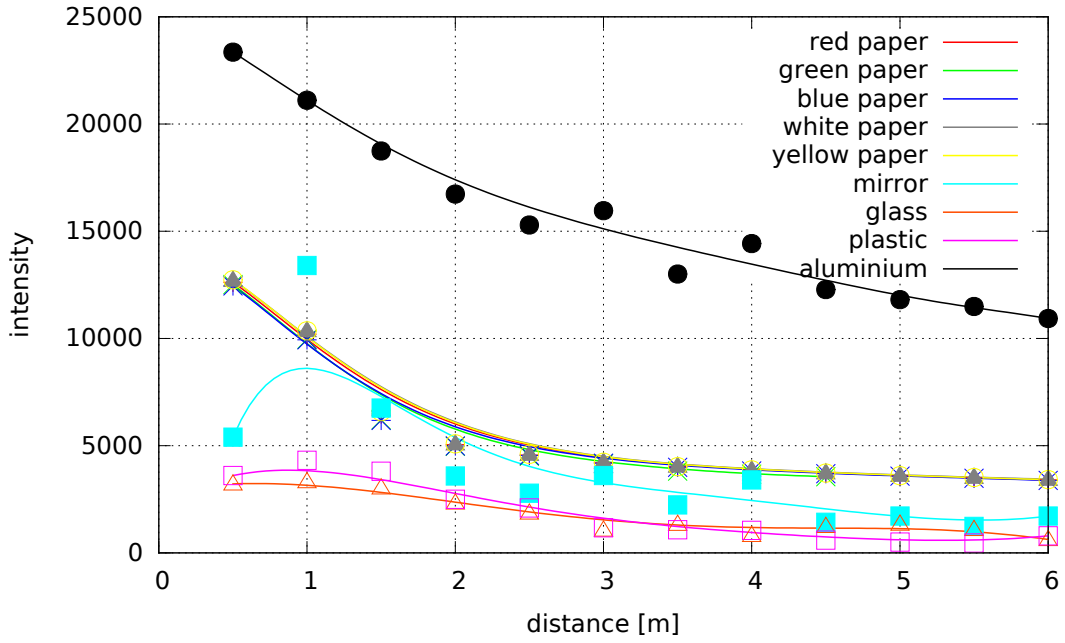
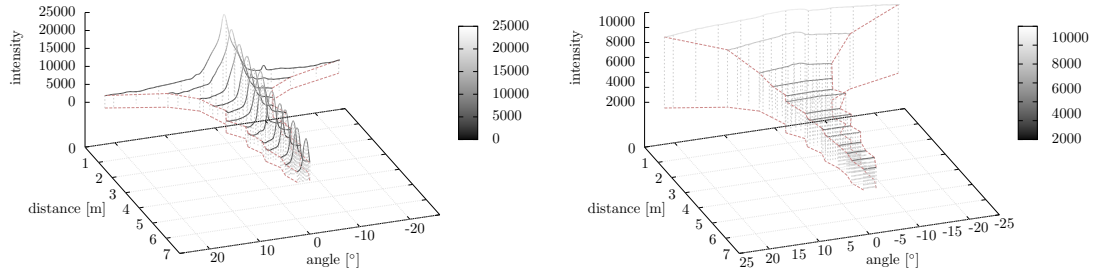


Figure 2.7: Intensity dependency for various surfaces and distances.

The curves of the transparent samples (glass and plastic) show results lower than the curve of diffuse reflective samples (colored papers). This results from the fact that most of the intensity bypasses the glass surface and hits an object behind. These objects have diffuse reflective properties. Thus, the light gets spread and the intensity value results are low.

Surface and angle dependency:
Angle dependency results from the reflection properties of light. For specular reflective surfaces (mirror, shiny metal) $\alpha_{in} = \alpha_{out}$. Hence, if the laser beam hits the surface perpendicular all light is returning to the laser scanner - the laser scanner sees itself. For a bigger angle the laser beam hits an object nearby. Since the object might be further away and most of the objects are diffuse reflective the returned intensity value is low. In compare, for diffuse surfaces the laser beam gets spread all over for every incoming angle. Thus, the effect on the dependency on the angle is less.

Figure 2.8a illustrates the angle and distance dependency at a specular reflective object (aluminium, a shiny metal) while Figure 2.8b illustrates it on an diffuse reflective object (white paper). Both curves result from Experiment 7.1 which will be explained in detail in Chapter 7. It have to be mentioned that the maximum value of the aluminium curve is about 2.5-times bigger than the maximum value of the white paper curve.



(a) Intensity values depending on the incident angle and the distance of an aluminium surface to the laser scanner.

(b) Intensity values depending on the incident angle and the distance of a white paper surface to the laser scanner.

Figure 2.8: Comparison of intensity values depending on the incident angle and the distance of the object surfaces (shiny aluminium with a specular reflective surface vs. white paper with a diffuse reflective surface) to the laser scanner.

Chapter 3

Sensor Systems for Range Measurements

In this chapter the focuses lie on range measurement principles and applied sensors for 2D and 3D-mapping. The first section explains range measuring principles utilized by common sensors. After, two sections describe 2D and 3D sensors used for mapping in robotics. The functional principle of each sensor is given as well as its advantages and disadvantages concerning mapping and measurement errors. Finally, the behaviour on measuring transparent and specular reflective objects is discussed.

3.1 Range Measurement Principles

Figure 3.1, an extraction of Nobach [2012], illustrates the typical two range measurement principals, their measurement range, and their typical accuracy. In the field of mobile robotics seldomly sensors with a measuring range $< 1m$ are deployed. For that reason, they are not taken into account in this work. Also stationary measuring systems are not considered since they are huge, heavy, or require a fixed measuring scene. That is why they are not applicable on mobile robots. Thus, range measurement principles are split up into two techniques: Time-of-Flight and Triangulation.

ToF is an active measuring principle which always applies a transmitter and a receiver unit. In contrast, triangulation based sensors can be active or passive as described later in Section 3.1.2.

It is understood, that sensors can be also categorized to be passive or active. Passive sensors rely only on a measuring unit, e.g. an RGB-camera detecting the “emitted light” from an object.

In compare, active sensors have a transmitter to send out a signal and a receiver to detect the returning signal. Active or passive is an important attribute when talking about power consumption. Besides, a passive system requires that the measured object “emits” the needed signal. One can simply say - no color photo at night, unless a flash light is used. Hence, an RGB-camera is not usable in darkness. In compare passive sensors have advantages at strong light influences. This strong light overlaps the emitted light of the active sensor system. So, it cannot measure anything.

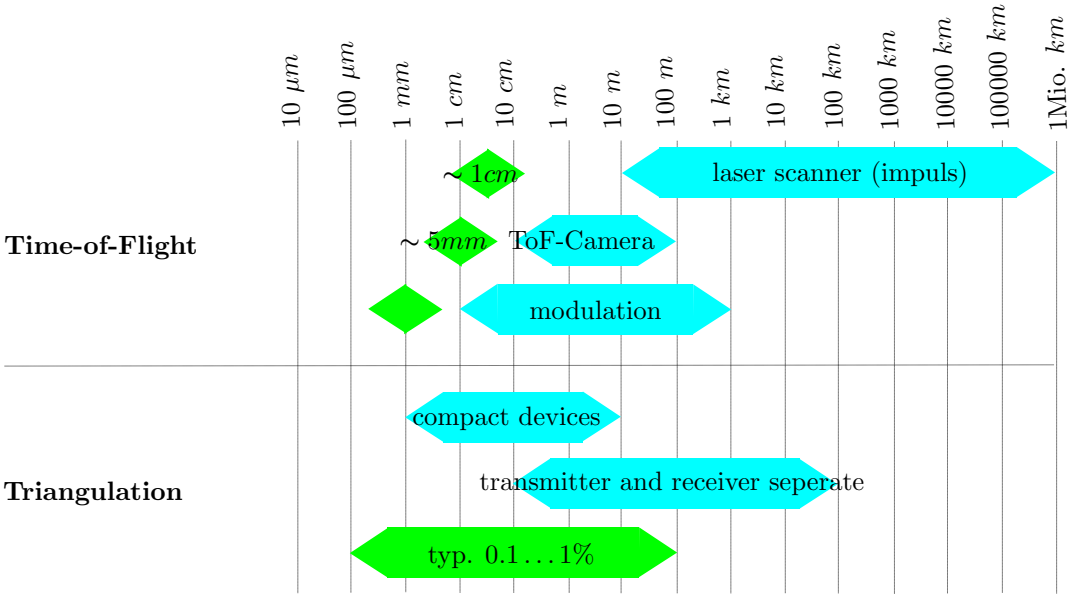


Figure 3.1: Overview of sensor principles with their measuring ranges (in light blue) and measurement precision range (in light green), an extraction of Nobach [2012].

3.1.1 Time-of-Flight

The basic ToF-sensor setup is illustrated in Figure 3.2. It consists of a transmitter, a receiver, and a controller to process the data.

The transmitter sends out a pulsed signal which is generated by the controller to the object. This signal is reflected on the surface and is returned to the receiver. The controller calculates the distance by the time difference Δt which is the time between the signal was sent out t_{send} and the time the signal was received t_{res}.

$$\Delta t = t_{res} - t_{send} \tag{3.1}$$

Based on the time difference Δt and the speed of the signal $v_{material}$ the signal travelling distance d_{signal} can be calculated by

$$d_{signal} = v_{material} \cdot \Delta t. \tag{3.2}$$

Since the object is half way of the signal path the distance to the object d_{object} results in

$$d_{object} = \frac{d_{signal}}{2}. \tag{3.3}$$

Common mobile robotic sensors apply light (e.g. laser scanner, ToF-camera), radio waves (e.g. radar), or sound waves (e.g. ultrasonic sensor, sonar) as a measurement signal. This results in a signal speed $v_{material}$ for light or radar $v_{material} = c = 3 \cdot 10^8$ m/s and for sound $v_{material} = v_{sound} = 343$ m/s. Light and radar are electromagnetic waves at different frequencies. That is why they have the same propagation speed.

It has to be mentioned that this assumes the signal travels through air. For other media (e.g. water, gases) the speed varies [Hammer et al., 2012]. Besides, the speed is also affected by temperature and (air) pressure. Nevertheless, in most cases these effects are neglected.

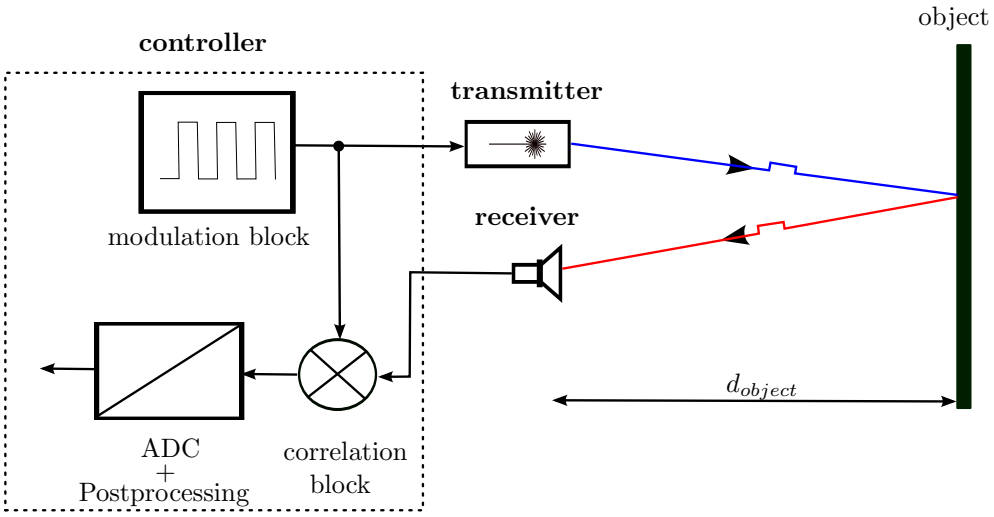


Figure 3.2: Time-of-Flight principle with its transmitter, receiver, and controller to process the data.

It is understood that time measuring is critical in this application, especially when a light signal is utilized. Light speed is high and a small measurement error leads to a huge inaccuracy in distance since they are directly related ($d_{signal} \sim \Delta t$). A precise time measurement requires costly and complex electronics. Remedy provides amplitude modulation (AM) or frequency modulation (FM) which is commonly utilized in light based systems ($v_{material} = c$).

Amplitude modulation:
Instead of sending a pulsed signal and await its return, at AM a continuous signal is broadcast. The signal with a frequency f_{AM} and an amplitude $\hat{a}_{AM}$ is modulated by vary the power. According to Jiang and Bunke [1997], the reference signal and the measurement signal vary in their amplitude $\hat{a}_{AM}$ but not in their frequency f_{AM}, cf. Figure 3.3. Since the light requires time Δt to travel to the object and back, a phase difference $\Delta\Phi$ results as:

$$\Delta\Phi = \frac{\Delta t}{\frac{1}{f_{AM}}} \cdot 2\pi = \frac{4\pi \cdot f_{AM} \cdot d_{object}}{c} \qquad \text{with} \qquad \Delta t = \frac{2d_{object}}{c}. \tag{3.4}$$

By tracking the difference in the signals amplitude the distance d_{object} can be calculated by

$$d_{object} = \frac{\Delta\Phi \cdot c}{4\pi \cdot 5 f_{AM}}. \tag{3.5}$$

Since the phase shifting is only possible by modulo 2π the uniqueness of the distance d^*_{object} is

$$d^*_{object} = \frac{c}{2f_{AM}} = \frac{\lambda_{AM}}{2}, \tag{3.6}$$

which is half of the wavelength λ_{AM}. Hence, points with a distance of

$$d_{object} + \frac{k \cdot \lambda_{AM}}{2}, \qquad d_{object} < d^*_{object}, \qquad k = 1, 2, 3, ... \tag{3.7}$$

create the same phase shifting

$$\Delta\Phi = \frac{4\pi \cdot f_{AM} \cdot d_{object}}{c} \qquad \text{(modulo } 2\pi\text{)} \tag{3.8}$$

and therefore it is not possible to distinguish between them.

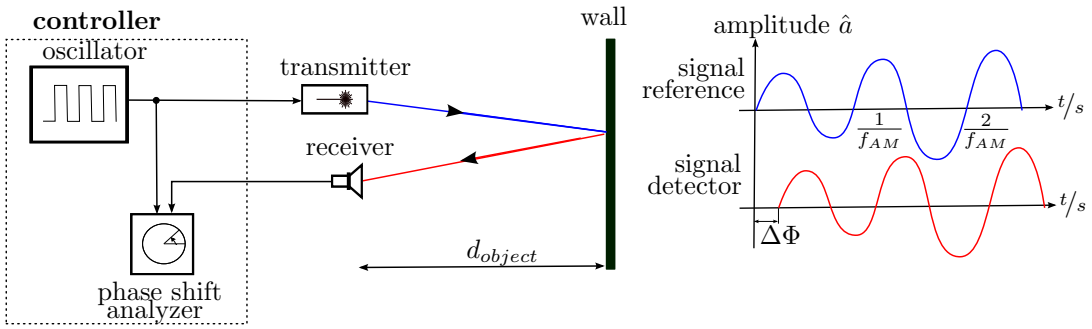


Figure 3.3: Amplitude modulator for ToF with its reference, detector, and the multiplexed signal.

Frequency modulation:
Apply a FM, cf. Figure 3.4, is a second way to relief the drawbacks of a standard ToF. Therefore, the current and the temperature of the emitting diode is controlled. Jiang and Bunke [1997] describes the overlap of both frequencies (f_{detect} and f_{ref}) creates a beat frequency

$$f_b = f_{detect} - f_{ref}. \tag{3.9}$$

Using the triangle relationship

$$\overline{BB'} = \frac{\overline{CC'}}{\overline{AC'}} \cdot \overline{AB} \tag{3.10}$$

and consider following correlations:

$$\overline{BB'} = f_b, \qquad \overline{AB} = \frac{2d_{object}}{c}, \qquad \overline{CC'} = \frac{\Delta f}{2}, \text{ and} \qquad \overline{AC} = \frac{1}{4f_{FM}}, \tag{3.11}$$

The distance can be calculated by

$$d_{object} = \frac{c \cdot f_b}{4f_{FM} \cdot \Delta f}. \tag{3.12}$$

It has to be mentioned that interferences between f_{detect} and f_{ref} are only possible for

$$d_{object,max} \ll \frac{c}{f_{FM}}.$$

Besides, for stationary systems, a reference distance measurement d_{ref} can be taken to determine exact values of f_{FM} and Δf. This method is easier than measuring f_{AM} and Δf directly.

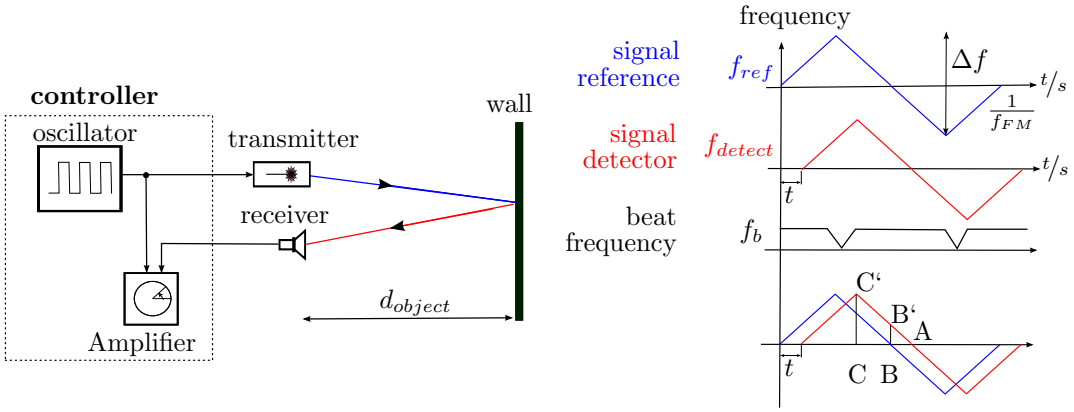


Figure 3.4: Frequency modulator for ToF with its reference and detector signal.

3.1.2 Triangulation

As previously mentioned, triangulation can be applied on an active or a passive sensor system. Active systems, also known as structured light, consist of: a light source which projects a line or a pattern onto an object; a receiver which detects the light pattern; and a controller to calculate the distance based on the distortion of the pattern. Passive systems, also known under the designation stereoscopy, consist of two cameras which record a scene from two different poses. They determine the distance based on displacements of points in the two perspectives. The basic idea of triangulation is to calculate the distance of the object d_{object} based on the triangle relationship, cf. Figure 3.5. The variables l_1 and l_2 are the lengths between the corresponding sensor and the point $\vec{p_0}$ from the hardware setup of the measurement system. Assuming that the base length b and the angles α and β, between the base line and the distances l_1 and l_2 are known, the distance d_{object} between the point and the point projected perpendicular on the base line can be calculated by

$$d_{object} = \frac{\sin(\alpha) \cdot \sin(\beta)}{\sin(\alpha + \beta)} \cdot b. \tag{3.13}$$

Figure 3.5 also illustrates that the working field of the system is based on the orientation of both cameras, the field of view (FOV) of both cameras, as well as the base length. Besides, the inner configuration of the system affects the calculations. It is understood that this is different for stereoscopy and for structured light based sensors.

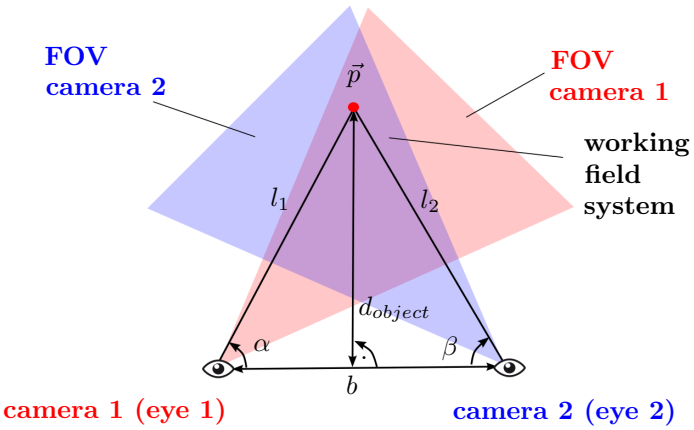


Figure 3.5: Principle of triangulation with two cameras.

Stereoscopy:
Stereoscopy, described by Nüchter [2009], is based on triangulation with two passive sensors, e.g. two RGB-cameras. Its functional principle is similar to the human vision system (their eyes) which sees objects from two perspectives. The points of the left and the right camera (eye) build up an epipolar plane (2D picture). The base line (distance between the two cameras) is known and therefore for each point $\vec{p}_i$ a triangle can be build up, cf. Figure 3.6. A human brain builds a 3D representation with the two pictures. While humans are only estimating distances of objects, in computer vision the coordinates of a point $\vec{p} = (x, y, z)$ can be calculated by:

$$\frac{z}{b} = \frac{f}{d} \Rightarrow z = \frac{b \cdot f}{d} \tag{3.14}$$

while f is the focal length of the cameras and d is disparity between two corresponding points on the image plane $d = x_l - x_r$. Further the coordinates x and y can be calculated by:

$$\begin{aligned} \tfrac{x+b/2}{z} &= \tfrac{x_l}{f} \\ \Rightarrow x &= \left(\tfrac{b}{2}\right) \cdot \left(\tfrac{x_l + x_r}{d}\right) \end{aligned} \tag{3.15}$$

and with $y^* = y_l = y_r$

$$y = \left(\frac{b}{2}\right) \cdot \left(\frac{2y}{d}\right) = \frac{b \cdot y^*}{d} \,. \tag{3.16}$$

It is understood that this technique requires that each point is seen with both cameras (cf. Figure 3.5). Furthermore, it needs a calibration to determine the exact orientation and location of both cameras to each other which are called the extrinsic parameters (rotation **R** and translation **Tr**). Besides the extrinsic parameters, a calibration determines also the intrinsic parameters (f, k_x, k_y, s) of each camera. Here, f describes the focal length of the camera, s describes the shear, k_x and k_y describe the pixel size of the camera in x and y direction. They are used to describe the inner configuration of a camera to transform from image plane coordinates to pixel coordinates. These are modulated in the pinhole camera model and the

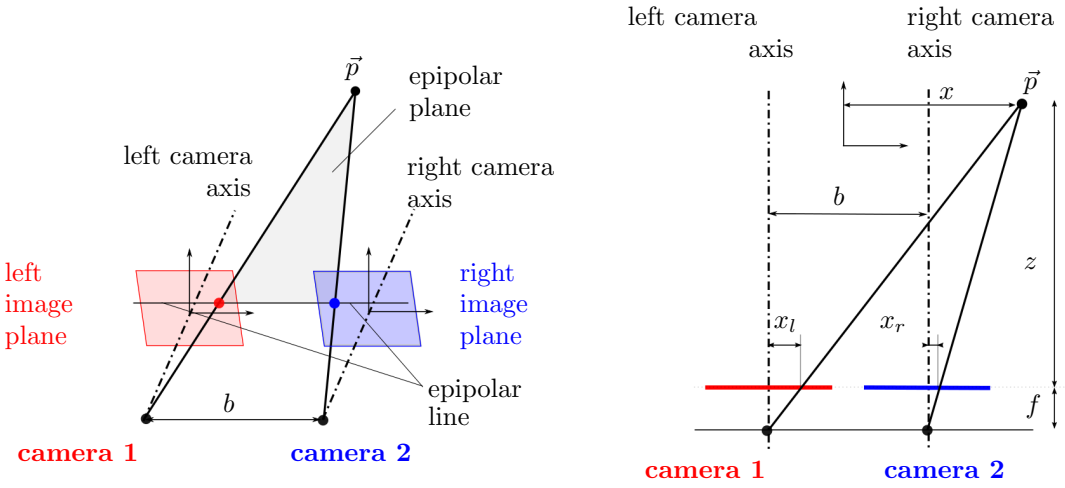


Figure 3.6: Principle of stereo vision.

image plane coordinates (u, v) are calculated by:

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \begin{pmatrix} f/k_x & s & u_0 & 0 \\ 0 & f/k_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.17}$$

while the image coordinates (u, v) then are calculated by:

$$u = u'/w' \tag{3.18}$$

$$v = v'/w' \tag{3.19}$$

Passive sensors do not require a light source but use ambient light. That is why they stand out with low power consumption. Besides, they do not broadcast a light pattern which can be cross-faded by external light. Hence, external light influences are less critical. Nevertheless, it is understood that complete darkness (no light at all) or direct strong sunlight shine into the lens (oversaturation sensor) bring such sensors to their limits.

Structured light:
Structured light sensors are active systems by using an emitter and a receiver. Depending on the dimensions of the emitted signal, the measurement dimension results in: a single light spot for a single measurement point (cf. Figure 3.7a), a line of light points for a line of measurement points for(cf. Figure 3.7b), and a light pattern for an area of measurement points (with only one camera picture) (cf. Figure 3.7c).

For a single point (1D) Jiang and Bunke [1997] describes triangulation similar to stereoscopy. The point $\vec{p} = (x, y, z)$ in the world coordinate system (WKS) is measured in the camera plane $\vec{p'} = (u, v)$. Based on the assumption that the x- and y-axis of the WKS are parallel to the u- and v-axis of the camera coordinate system and that the z-axis of the WKS is parallel with the optical axis f of the camera, the following correlation applies:

$$\frac{x}{z} = \frac{u}{-f} \qquad and \qquad \frac{y}{z} = \frac{v}{-f}. \tag{3.20}$$

Considering the line l_1 in the xz-plane, the coordinate z follows as

$$z = -\frac{f}{u} \cdot x. \tag{3.21}$$

Further, the line l_2 in the xz-plane spans up the angle α with the x-axis. Hence, the coordinate z follows as

$$z = (x - b) \cdot \tan(\alpha). \tag{3.22}$$

That is why the coordinates of point $\vec{p}$ can be determined by combining the equations 3.20, 3.21, and 3.22 to:

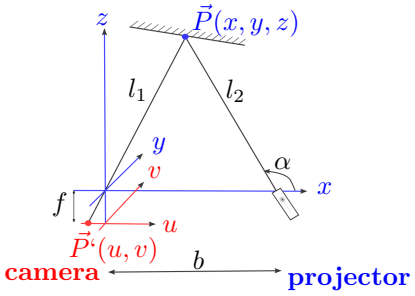
$$x = \frac{b \cdot \tan(\alpha)}{u \cdot \tan(\alpha) + f} \cdot u, \tag{3.23}$$

$$y = \frac{b \cdot \tan(\alpha)}{u \cdot \tan(\alpha) + f} \cdot v, \tag{3.24}$$

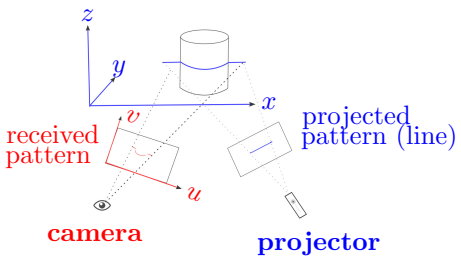
$$z = \frac{-b \cdot \tan(\alpha)}{u \cdot \tan(\alpha) + f} \cdot f. \tag{3.25}$$

The movement d_x and d_y of the source (the measurement signal point) in the WKS result in a change in the camera coordinate system, calculated by

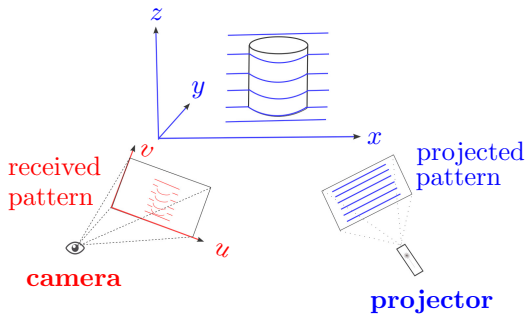
$$u = d_x \cdot i \qquad \text{and} \qquad v = d_y \cdot i. \tag{3.26}$$



(a) Setup of a structured light sensor for 1D measurements.



(b) Setup of a structured light sensor for 2D measurements.



(c) Setup of a structured light sensor for 3D measurements.

Figure 3.7: Measurement principle of structured light sensor.

3.2 2D Sensors

The second section describes 2D sensors applied in mapping. It is understood that for advanced navigation a 3D reconstruction of the environment is necessary. Nevertheless, many robots are still based on 2D mapping and navigation because 2D is easier to handle. Besides, it requires less computing power and therefore it is faster. One should mention that every 3D Sensor can be used for 2D as well. Hence, in the point cloud a defined cut in parallel to the ground plane is enforced. This cut represents a scan plane as it is seen from a 2D sensor. Such a scan plane looks like a floor plan in buildings.

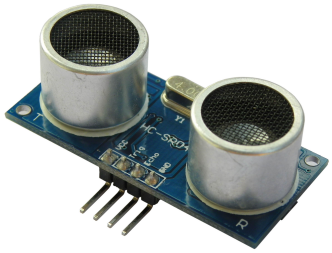
3.2.1 Ultrasonic Sensors

Ultrasonic sensors are using sound waves at a frequency of 20 kHz to several GHz. Based on the sensing purpose their measuring principle relies on Time of Flight (distance), Doppler shift (velocity), or amplitude modulation (distance, directionality, or attenuation coefficient).

This sub-section concentrates on ToF based ultrasonic sensors as they are established in robotics. In the field of mapping they are used for collision avoidance and distance measuring. They are also used for position control, filling level detection, or object detection. For this purpose two types exist - double head sensors (cf. Figure 3.8a) and single head sensors (cf. Figure 3.8c and Figure 3.8b). HC-SR04 (Figure 3.8c and the HG-M40DIA (Figure 3.8b) are low cost sensors in compare to the UBE1000-18GM40-SE2-V1 (Figure 3.8a). They are typically employed with Arduino microprocessors [Arduino, 2017] and various manufacturers offer them.

(a) Double head sensor: PEPPERL+FUCHS through-beam ultrasonic barrier UBE1000-18GM40-SE2-V1 [Pepperl+Fuchs, 2017].

(b) Single head sensor with switching transceiver and receiver unit: Ultrasonic Ranging module HG-M40DAI [Arduino, 2017].



(c) Single head sensor with seperate transceiver and receiver unit: Ultrasonic Ranging module HC-SR04 [Arduino, 2017].

Figure 3.8: Examples of different ultrasonic sensors.

Double head sensors have a separate transceiver and receiver unit. Usually, the units are mounted opposite to each other to build up a barrier. A typical application field is overtravel protection, e.g. for a vacuum cleaning robot, to limit his working area. This is not part of the following work and therefore double head sensors are not considered further.

In contrast to double head sensors, the transceiver and receiver unit is combined in one unit (cf. Figure 3.8b) at single head sensors. In sending mode a voltage source stimulates the internal piezo-crystal to send out a wave. After, the sensor switches to receiving mode to detect the returning wave by measuring the voltage on the piezo-crystal. It is understood that this requires time which results in a measurement limitation. Only if objects are farther away than

the travelling distance of this switching time they are detectable. To overcome this problem, some sensors employ an individual transceiver and receiver unit both of which are placed on the same board next to each other (cf. Figure 3.8c). The transmitter sends out a signal and the receiver awaits its return as described in Section 3.1.1.

Ultrasonic sensors suffer from four disadvantages. First, the reflected wave strongly depends on the incident angle and the roughness of the measured object surface. It is hard to detect sound-absorbing materials (soft material), round objects, and objects with a rough surface. Also thin foils have an absorbing effect. In contrast, it is possible to detect solid objects, fluids, as well as mirrors and transparent objects.

Another disadvantage of ultrasonic sensors is their huge opening angle of the sound beam. The farther away the object is the more the beam spreads. This is similar to the real laser beam (cf. Figure 2.6b) except the opening angle is much bigger. As a result, the returning wave is an overlay of a measured area but not of a single point. That makes it hard to detect small objects, objects far away, and to obtain a high resolution.

A third drawback is the slow propagation speed of ultrasonic waves ($v_{material} = v_{sound} = 343^{m}/_{s}$), especially in compare to light ($v_{material} = c = 3 \cdot 10^{8 m}/_{s}$). Since the measurement speed is based on the propagation time of the ultrasonic wave it takes a long time to determine each measurement. While the sensor is awaiting the returning signal another signal cannot be transmitted. Otherwise, an overlay of both signals results in erroneous data.

As a last disadvantage, sound waves are strongly affected by gas properties. This influences the flow of the wave at the borders between different gases as well as the propagation speed. In case of gas with less density the transfer is worse and vice versa. Higher temperatures of gas or a lower pressure also lower the density of the gas. Hence, the transfer gets worse too. This results in inaccuracies in the distance determination.

Sonar is similar to ultrasonic sensors but specialized for underwater applications.

3.2.2 2D Laser Scanners

In mobile robotics laser scanners are also called Light Detection and Ranging sensor (LIDAR). Most of them are based on semiconductor technology and therefore apply a diode to generate the laser beam. That is why only this technology is considered further. The laser beam hits a mirror which rotates with constant speed and redirects the beam. A receiver awaits the returning light signal. Depending on its measuring speed and the rotating speed of the mirror a plane with $\{n = i + 1\}$ points spans up, cf. Figure 3.9.

Some laser scanners, also called multi-echo laser scanners, are capable to await multiple echoes of each scan point. The distance to each point is determined multiple times. For transparent or specular reflective objects the distances of each echo vary. Deployment of intensity values of the returning signal is another common feature. This provides information about the material of the scanned object which was mentioned in Section 2.3. Both features are employed in Chapter 6 to identify transparent and specular reflective influences and to distinguish between the different objects.

Laser scanners are based on the ToF principle explained in Section 3.1.1. As they use light as a signal, they provide measurements quickly with high accuracy. Due to the high intensity of the beam, laser scanners are less affected by external light sources (e.g. sun light) than ToF-cameras or stereo cameras. This makes them operable indoor and outdoor. They also have a wide scanning range. That is why they are a favoured sensors in mobile robotics though they are expensive.

One big drawback of laser scanners is that their measurements are influenced by transparent and specular reflective objects. For example, mirrors can cause a reflection and therefore

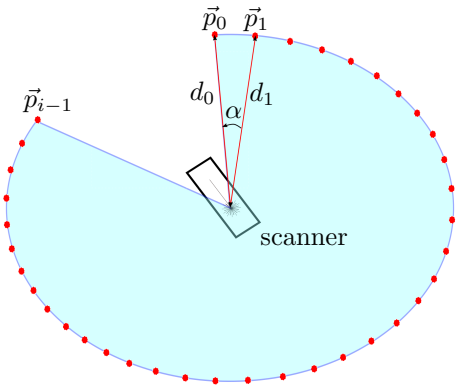


Figure 3.9: Scan plane of 2D laser scanner.

the measurements result from mirrored objects. In contrast, glass can cause refractions and therefore measurements result from an object behind. The surface itself is only occasionally visible. All depends on the incident angle of the laser beam as described in detail in Chapter 2.

Figure 3.10 illustrates the Hokuyo UTM-30LX-EW multi-echo laser scanner which was used for the experiments in Chapter 7. It delivers for each measured point $\vec{p}_i$ a tuple with three echoes and intensities. It delivers every 25 ms a set of 1081 scan points. Its scanning plane spans up an area of 270° which results in an angular resolution of 0.25° with a scanning range between $0.1 - 30.0$ m.

Figure 3.10: Hokuyo UTM-30LX-EW multi-echo laser scanner [Hokuyo, 2017].

3.3 3D Sensors

As mentioned, 2D sensors deliver a parallel cut to the ground plane of the environment. In compare, 3D sensors result in a point cloud which is a three dimensional representation of the environment. A three dimensional mapping is required to act and react in real world scenarios like navigate on uneven terrain, use manipulators to grab objects, etc. Although many approaches still rely on 2D it is just a matter of time till 3D approaches will overcome them. Despite the fact that they are more complex and need more computing power. The following section describes common 3D sensors applied for mapping.

3.3.1 Stereo Cameras

Figure 3.11 illustrates the Bumblebee2 stereo vision camera from FLIR [2017]. The measuring principle is based on triangulation (see Section 3.1.2) which is similar to the human vision. Based on the model and the focal length of the applied aperture it results in a horizontal field of view (FOV) between $43° - 97°$. Regarding the applied CCD sensor a monochrome or coloured resolution of 1032×776 pixels for 20 FPS (frames per second) and 648×488 pixels for 48 FPS result.

Figure 3.11: FLIR Bumblebee 2 stereo camera [FLIR, 2017].

A cheap stereo camera with an high resolution can be built up simply by mounting two RGB-cameras (mostly based on CMOS) together. As a drawback, such a camera needs to be calibrated to determine intrinsic and extrinsic parameters before usage. This can be achieved, e.g. by using a calibration board (chess board). After taking measurements from multiple positions the required equations can be solved and the parameters determined.

Another drawback lies in the dependency of the measurement distance from the base length b. For great base lengths the FOV of both cameras results to good far distance measurements. On the other hand, for a small base length the angle Θ is small and therefore objects far away are not recognisable. That is why some stereo cameras consist of three cameras resulting in three different base lengths b_1, b_2, and b_3. One base length accomplishes near distance measuring, one middle distance measuring, and one for far distance measuring.

Even though stereo cameras are operable indoor and outdoor they suffer from strong light influences. Too strong light overexposes the CMOS or CCD sensor of the cameras which means that no picture results. In contrast, they are not influenced by transparent and specular reflective objects.

3.3.2 Structured Light Sensors / RGB-D-Camera

Structured light sensors, also called projection scanner or projection 3D scanner, use a transmitter (projector) and a receiver (camera) to determine distances. Their measuring principle is also based on triangulation (see Section 3.1.2). Compared to stereo cameras they apply active measuring. Based on the projector pattern following measurements can be established:

- 1D scanner: projection is a single spot, e.g. laser spot
- 2D scanner: projection is a line, e.g. line laser
- 3D scanner: projection is a pattern, e.g. infrared pattern (IR)

Figure 3.12: Microsoft Kinect camera V1 for XBox 360 (gaming box) [Microsoft, 2017].

Structured light sensors suffer from influences of external light as it cross-fades the projected light pattern. Hence, measurements are faulty or not available at all. This makes the sensors hardly usable outdoors. Besides, the measuring range is limited as it strongly depends on the power of the light source.

Nevertheless, the outcome of the Microsoft Kinect camera [Microsoft, 2017] in 2010, cf. Figure 3.12 with its low cost made these sensors attractive. In fact, the measuring principle of the Kinect V1 from 2010 is based on triangulation while Kinect V2 from 2013 is based on ToF. Kinect cameras have a measuring range of $0.3 - 4\ m$ (depending on the configuration). For each measurement point $\vec{p_i}$ they deliver depth as well as RGB information. That is why they also are called RGB-D-sensor. Note that the pattern of the Kinect is not visible for human eyes as its wave length is in the IR range. That is why a combination of RGB and depth information is possible without any disturbances caused by the pattern (changed colors of the object caused by the light pattern). It is understood that such a sensor needs to be calibrated to determine intrinsic and extrinsic parameters. These parameters are internally stored and the data pre-processed in the sensor. Similar to an RGB-camera the Kinect has a limited FOV ($43° \times 57°$).

Compared with a 3D laser scanner, the update rate of 30 samples per second is quite high, but their measurements also suffer from transparent and specular object influences. This is due the fact that the light pattern relies on the same regularities as the laser beam of a laser scanner. Further, it is understood that the broadcast pattern underlies disturbances when using multiple cameras. A synchronisation is required to assure that one camera after the other one measures.

3.3.3 Time of Flight-Camera

Most ToF-cameras use an LED array for emitting and a CMOS camera for sensing. Their FOV is similar to structured light cameras. Figure 3.13 illustrates the Kinect Xbox V2 and the SwissRanger 4000, a gaming vs. an industrial ToF-sensor.

(a) Microsoft Kinect camera V2 for XBox 360 (gaming box) [Microsoft, 2017].

(b) Swiss Ranger SR4000 ToF camera [MESA, 2017].

Figure 3.13: Examples of common ToF cameras.

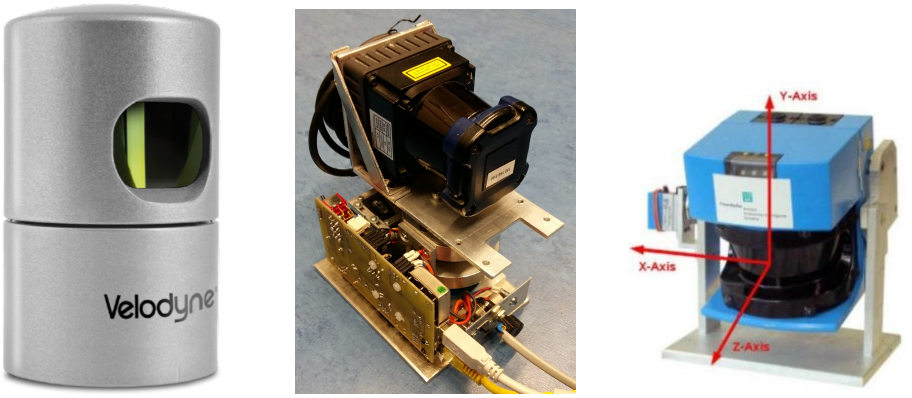
The cameras are based on the phase-shift measurement principle. They illuminate the object with a light in the near-infrared range (NIR) and the phase shift is measured for each point. In compare to laser scanners, ToF-cameras have a short measuring time as they measure multiple points with one shot. In contrast, they suffer from external light influences, e.g. sunlight, as the light source is illuminating an area and not a single point as this is the case for a laser scanner. This makes it difficult to operate them outdoor. Disturbances also occur when multiple cameras are used because of crosstalk. This is caused by an overlay of the illumination patterns of the individual cameras. A synchronisation is required to assure that only one camera measures at a time. Further, the measuring distance depends on the power of the light source which needs to be strong enough to illuminate the scene. Similar to laser scanners and structured light sensors, measurements of ToF-cameras get influenced by transparent and specular reflective objects.

3.3.4 3D Laser Scanners

The difference between a 2D and a 3D laser scanner lies in an additional axis which is commonly achieved by rotating a 2D scan plane. Figure 3.14 illustrates three different 3D laser scanners. The Velodyne HDL32 from Velodyne LIDAR (Figure 3.14a) uses 32 laser beams across a 40° vertical FOV. This set of laser beams is rotated to achieve a 3D image. The scanner has an accuracy of $\pm 2\ cm$ at a scanning range up to $100m$ [Velodyne, 2017].

In compare to the Velodyne, the rotating Hokuyo (Figure 3.14b) and the pitching Sick (Figure 3.14c) are "self-made" 3D laser scanners on the basis of commercial a 2D laser scanner. The Hokuyo rotates at a constant velocity which results in a measurement plane for each scan. The individual scan planes are then stitched together, similar to the Velodyne. The angular resolution results from the rotating velocity and the scan time for one plane. It is understood that the movement while scanning results in a minor error. It can be recalculated if the rotational speed is known or neglected, if the rotational speed is much slower than the scan speed for one scan plane.

In compare to the rotating Hokuyo, the pitching Sick is moved at a predefined angle; stopped till a scan was taken; and moved again to the next scan position. This process takes time, but prevents a recalculation of the point since there is no movement while scanning. A similar error



(a) Velodyne HDL-32E laser rangefinder [Velodyne, 2017].

(b) Turnable Hokuyo UTM-30LX-EW multi-echo laser scanner.

(c) Pitching Sick LMS-200 laser scanner [Surmann et al., 2003].

Figure 3.14: Rotating and pitching laser scanner.

occurs for both scanners (the rotating and the pitching scanners) when the robot is moving while scanning. Exemplary, a movement error is illustrated for a 2D laser scanner in Figure 3.15. The robot is moving with a constant velocity v. That is why each measurement point is recorded from a different position. Since this is not taking into account, the measurement distance of the points are wrong. Based on the direction of the movement the points appear farther away or closer resulting in a deformation of the circle. This is a drawback of the 3D laser scanner as for stereo cameras or ToF-cameras these moving errors can be neglected (due to slow moving speed in compare to measurement time). To prevent this effect, the robot should be stopped for taking a scan with the 3D laser scanner as this was done during the experiments.

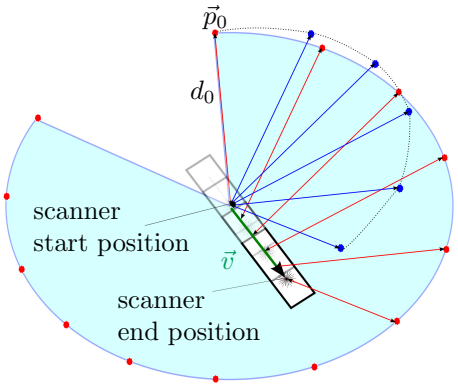


Figure 3.15: Movement errors at a laser scanners. If the scanner remains at the start position it will measure the red points. As it moves with a constant velocity $\vec{v}$, each point is measured from a different position (illustrated as a red arrow). The distance varies in compare to the distance from the start position. Since the angular step is constant and the scanner refers all measured points to the start position, the blue points result.

Even then, “self-made” 3D laser scanners have two major advantages in compare with commercial 3D laser scanner - lower cost and better capabilities. For example, the Velodyne costs several ten thousands Euro while the self-made scanner is less than ten thousand Euro. The vertical resolution of the “self-made” laser scanner is higher than the resolution of the Velodyne (self-made ∼1081 points, Velodyne ∼32 points). This means a higher density of points and therefore more detailed information. Furthermore, the FOV, especially of the rotating Hokuyo, is greater (270°) than the FOV of the Velodyne (40°). This stems from the alignment of the 2D laser scanner, cf. Figure 3.16.

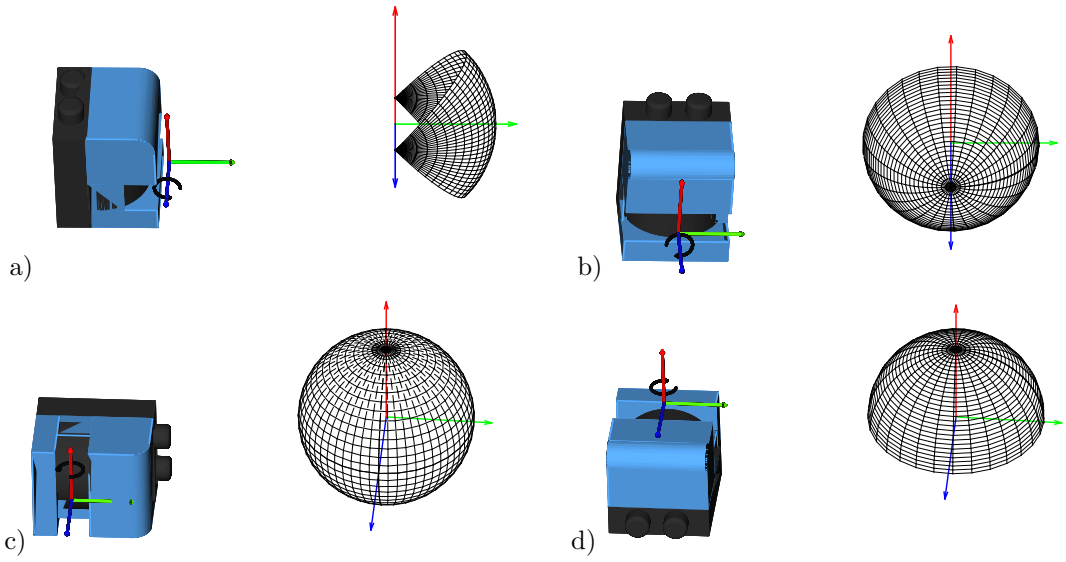


Figure 3.16: Scanning schemes (left) and measurement density distribution (right) of rotating 2D laser scanner (SICK LMS) to acomplish 3D measurements: (a) pitching scan, (b) rolling scan, (c) yawing scan, and (d) yawing scan top, reproduced from [Wulf and Wagner, 2003].

Unfortunately, two drawbacks arise: higher computing power and a calibration is required. This is due the fact that a misalignment of the scanner w.r.t. the rotating axis results in an error. One way to handle this is presented by Koch et al. [2017a].

As mentioned before, 3D laser scanners require more time to measure (for one set of points) than other sensors, e.g. RGB-camera, ToF-camera, etc. Nevertheless, they are favoured since their measurements are more precise, their scanning range is greater, and their susceptibility to external light is negligible.

Chapter 4

Simultaneous-Localization-and-Mapping-Approaches (SLAM)

As pointed out in the introduction, mapping is an essential task for mobile robots. Without it, the robot is not capable to operate reliably in a real world scenario: e.g. navigate to a position or proceed with manipulation tasks.

This chapter describes 2D and 3D Simultaneous-Localization-and-Mapping (SLAM) algorithms which are utilized for the experiments in Chapter 7. The Mirror-Identifier-Approach, core of this thesis is an add-on to any mapping algorithm. It processes data received from the laser scanner to detect, identify, and purge transparent and specular reflective influences.

SLAM is an hen-egg-problem. On the one hand, a map is needed to localize the robot. On the other hand, the position of the robot is required to build the map. Basically, two concepts are available to deal with this: graphical SLAM and probabilistic SLAM. The following section describes the individual applied method to overcome this problem. Furthermore, assets and drawbacks of the applied mapping approaches are pointed out. Since the applied robot does not support any odometry a visual localization is required. Therefore, a pose estimation based on the laser scanner is utilized. This is the reason why only approaches which do not rely on odometry data are considered.

4.1 CRSM-SLAM

Critical-Ray-Scan-Match-Simultaneous-Localization-and-Mapping (CRSM-SLAM) from Tsardoulias and Petrou ([Tsardoulias and Loukas, 2013] and [Tsardoulias, 2015]) is a probabilistic based 2D mapping approach. It is an extended version of Scan-Matching-Genetic-SLAM (SMG-SLAM) which was presented by Mingas et al. [2012]. The main idea of CRSM-SLAM relies in downsizing matching effort to reduce calculation power and time. This is achieved by using only selected critical rays which are essential rays (measurements) needed for the hill climbing matching function. For example, these rays consist of a large displacement or have larger measurements. To adjust the approach and make it parameterizable to the environment, CRSM offers different ray selection methods. The resulting maps are occupancy grid based with a probability p_{cell} ($p_{cell} = 1$ for unexplored space, $p_{cell} = 0.5$ for free space, and $p_{cell} = 0$ for obstacles).

In compare to other SLAM algorithms, the current scan is not matched to a previous scan but to the entire map. This reduces cumulative errors and therefore loop-closure is not required. Besides, changes in the environment will be respected as the CRSM-SLAM is a dynamic mapping algorithm.

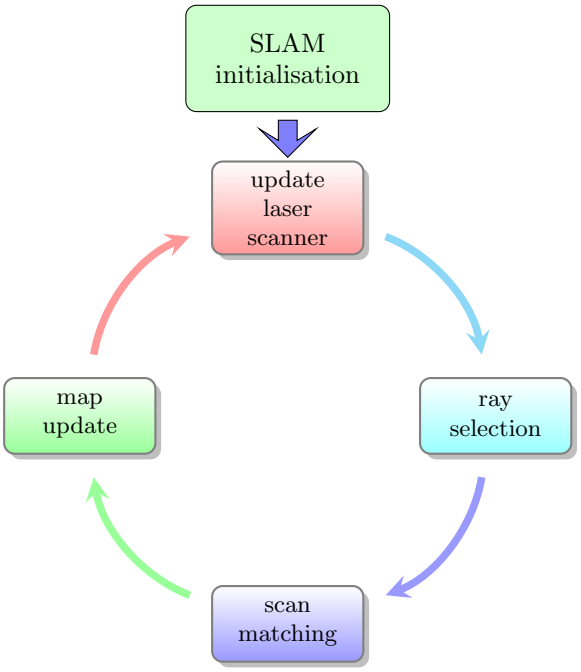


Figure 4.1: Procedure of CRSM-SLAM, reproduced from [Tsardoulias and Loukas, 2013].

Figure 4.1 illustrates the procedure of the CRSM-SLAM. After initialization, it runs in a loop consisting of four modules: laser scanner update, ray selection, scan matching, and map update.

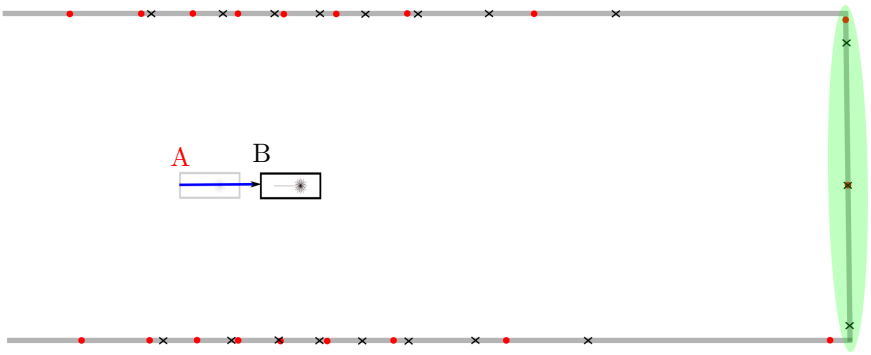
4.1.1 Laser Scanner Update

The first module (highlighted in red) fetches new measurements from the laser scanner. It converts the distance $d_i = \{d_i | i = 1, \ldots, N\}$ with N scan points into map cells. This results in a grid position $\vec{p}_i = \{\vec{p}_i | i = 1, \ldots, N\}$ with $\vec{p}_i = (x, y)$ located at the end of each ray. The center is the current position of the robot. Rays with a distance greater than $0,95 \cdot d_{maxScanner}$ are filtered out.

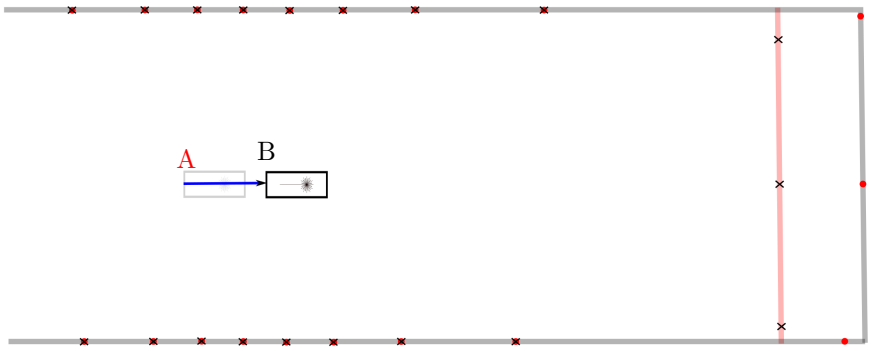
4.1.2 Ray Selection

The second module (highlighted in turquoise) selects the critical rays from the measurements. This reduces data, calculation power, and time for the third module - scan matching (highlighted in blue). Four methods are selectable to minor the amount of rays: all rays, uniform sub-sampling, density based selection, and density based with sub-sampling.

1^{st} method: "all rays selected":
Using all rays is most common in scan matching. This method leads to good results since all information is available for the hill climbing step which determines the robot's pose $\vec{P}_R = (x, y, \theta)$. It is understood that this also demands maximum calculation time due to the large amount of scan points N. Because of the long calculation time this method is not applicable online. Besides, it has drawbacks when mapping environments with narrow corridors.



(a) Matching of two scans (scan 1 = red points, scan 2 = black crosses) as it should be (highlighted in light green).



(b) Matching of two scans (scan 1 = red points, scan 2 = black crosses) and the resulting mismatched wall (red line).

Figure 4.2: Mismatching of two scans in a narrow corridor. The robot took scan 1 at location A (red points) and moved to location B, as illustrated by the blue arrow. At location B scan 2 (black crosses) was taken. Since the matcher minimizes the error over all points, the points at the end of the corridor are given less consideration than the points close to the robot. The points located on the end of the corridor of scan 2 are mismatched. That is why a second wall (red line) appears.

For narrow corridors the scan points are unfavourably aligned, see Figure 4.2a. The red points result from scan 1 (at Position A) while the black crosses result from scan 2 (at Position B). Based on the intercept theorem, close to the robot more measurements result from the wall. The longer the corridor is the farther away they spread. Only a few measurements result from the end of the corridor. When matching, the algorithm minimizes the total distance error

$$d_{error} = \sum_{i=0}^{N-1} (\|\vec{p}_i - \vec{p}_{i+1}\|). \tag{4.1}$$

Because of that, the higher amount of points close to the robot are given greater consideration than the few points at the end of the corridor. As a result, the two scans are matched incorrectly which is illustrated in Figure 4.2b.

2^{nd} method: "uniform sub-sampling":
The second method applies the initial concept of SMG-SLAM [Mingas et al., 2012] (previous version of CRSM-SLAM). It is a simple and fast method as it has less complexity than feature based sub-sampling. The reduction is achieved by choosing every $M_{SkipStep}$-ray, of N supplied rays from the laser scanner. The amount of rays is reduced to

$$N_{step} = \frac{N}{M_{SkipStep}} \tag{4.2}$$

and each ray with

$$PickIndices_i = M_{SkipStep} \cdot i, \qquad 0 \leq i \leq N_{step} \tag{4.3}$$

is selected for the scan matching. In compare to the first method ("all rays selected"), the calculation time of the hill climbing step function is $M_{SkipStep}$-times faster. The matching quality is worse as this method tends to favour close obstacles. This is due the fact that close obstacles result in more measurements. Figure 4.3 illustrates this effect. The size of obstacle O_1 and O_2 is equal. Since obstacle O_1 is closer to the laser scanner it results in more measurements (six) than obstacle O_2 (three) which is farther away. The reason for that is based on the consistent angle step size between two measurements. The distance between the scan points w double for twice the distance from the laser scanner d:

$$\frac{d_1}{d_2} = \frac{w_1}{w_2} \tag{4.4}$$

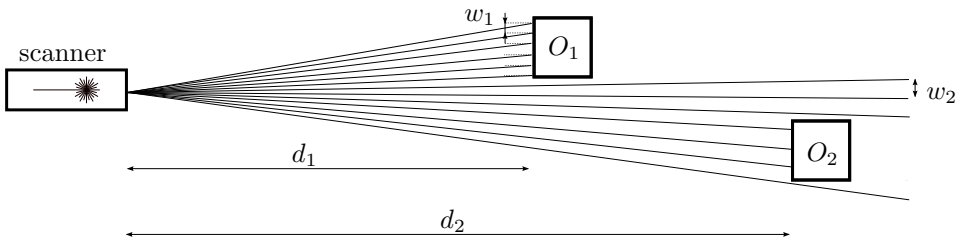


Figure 4.3: Amount of measurements of an obstacle located close to the scanner in compare to an obstacle located far away.

Also for this method narrow environments lead to failures when matching new scans.

3^{rd} method: "density based selection":
In contrast to the first two methods, the density based selection method gives measurements close to the robot less confidence than measurements farther away. To do so, a fixed distance $d_{rayDisplacement}$ between two selected measurements is used to sub-sample. This results in a uniform sub-sampling of measurements based on the density. For that, the total distance over all rays d_{total} needs to be calculated by

$$d_{total} = \sum_{i=0}^{N-1} D_{rayDisplacement,i}. \tag{4.5}$$

The ray displacement matrix $D_{rayDisplacement}$ for each point N is calculated by

$$D_{rayDisplacement,i} = \sqrt{d_i^2 + d_{i+1}^2 - 2d_i \cdot d_{i+1} \cdot \cos(\alpha_{Step}/N)}. \tag{4.6}$$

Here, d are the measured distances of the points i and $(i+1)$ and α_{Step} is the step size of the laser scanners scan angle. It results from the angular scan range α_{Laser} and the amount of scan points N to

$$\alpha_{Step} = \frac{\alpha_{Laser}}{N}. \tag{4.7}$$

Following, the mean displacement

$$d_{mean} = \frac{d_{total}}{(N-1)} \tag{4.8}$$

is calculated. To select the $M_{request}$ requested rays, the indices for the chosen measurements are determined by

$$\begin{aligned} PickIndices_i = index, (d_{total}/M_{request}) \cdot (i-1) \\ \leq D_{rayDisplacement,i} \\ \leq (d_{total}/M_{request}) \cdot (i) \qquad \text{for } 1 \leq i \leq M_{request}. \end{aligned} \tag{4.9}$$

This method also tends to fail for environments with corridors. Reason being that the number of selected measurements at the end of the corridor is low in compare to the selected measurements on the walls.

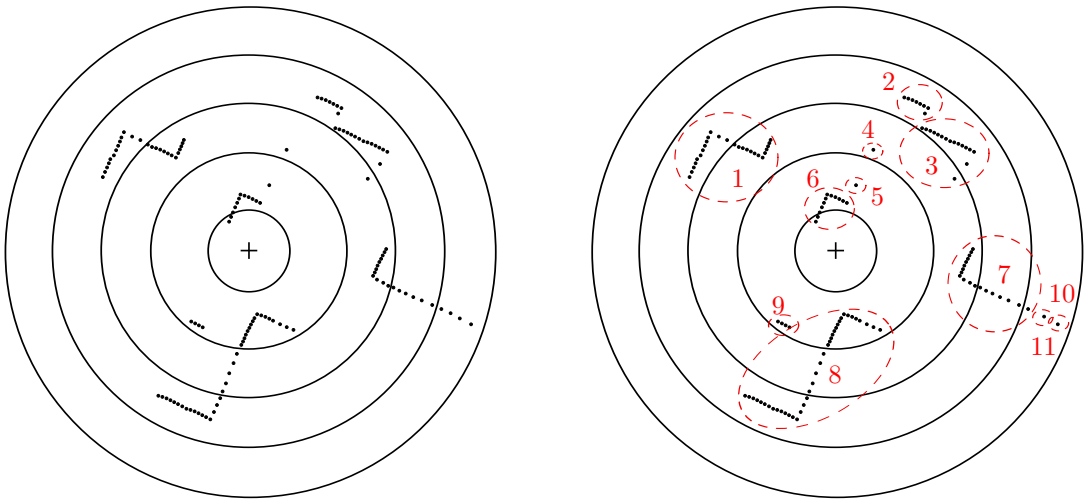
4^{th} method: "density based with scan segments":
The last method is designed to fix previous mentioned drawbacks. Now, the segmentation is applied based on edges, distance to the scanner, and density of scan points as illustrated in Figure 4.4b. Three steps are provided: First, edges in the ray displacement matrix $D_{rayDisplacement}$ are identified by checking its values for $D_{rayDisplacement,i} > \epsilon_{edge}$. This results in the elements of the feature edge matrix $F_{featureEdge}$, cf. Figure 4.4a.

Second, rays are selected based on the density as previously described. For each detected feature the mean spareness S_{spar} is determined by

$$S_{spar} = \frac{\sum_{j=\text{Start}}^{\text{End}} D_{rayDisplacement,j}}{N_{sizeFeature}} \tag{4.10}$$

with $START = F_{featureEdge,i}$, $END = F_{featureEdge,i+1}$, and $N_{sizeFeature} = END - START$. Rays are selected according to Algorithm 1. D_{local} is the local displacement, ρ is a parameter to adapt the algorithm to various environments, $d_{rel,i}$ is the relative distance value of a measurement, d_{Max} is the maximum measured distance in the scan, and ϵ_{RAY_Thres} is the minimum step size which should proceeded between two selected rays.

The comparison in line 5 of Algorithm 1 can be simplified to "$D_{local} \geq S_{Spar}$". This is similar to the classical density based selection and results in rays close to $START$ and END. To take into account that points further away from the robot are preferred, the snippet "$\frac{1}{e^{d_i}}$" is added.



(a) Measurements in feature edge matrix $F_{featureEdge}$.

(b) Exemplary segmentation of a laser scan, based on edge detection and distance.

Figure 4.4: Segmentation of critical rays based on scan segments and density [Tsardoulias and Loukas, 2013].

Algorithm 1 CRSM-SLAM - select rays [Tsardoulias and Loukas, 2013]

1: **procedure** SELECTRAYS()
2: $D_{local} = 0$
3: **for** $(i = START; i < END; i$++$)$ **do**
4: $D_{local} = D_{local} + D_{rayDisplacement,i}$
5: **if** $D_{local} \geq \frac{\rho_{env} \cdot S_{Spar}}{\frac{e^{d_i}}{\sqrt{d_{Max}}}}$ **then**
6: pick ray
7: $D_{local} = 0$
8: **end if**
9: **end for**
10: **for** each i **do**
11: **if** $(PickedIndices_{i+1} - PickedIndices_i) > \epsilon_{RAY\_Thres}$ **then**
12: pick ray with $index = \frac{PickedIndices_{i+1} + PickedIndices_i}{2}$
13: **end if**
14: **end for**
15: **end procedure**

The measurement summation has good results for environments which are huge and rich of features. But, a small amount of selected rays result for narrow environments. To overcome this drawback the snippet "$\sqrt{d_{Max}}$" is added. Last but not least, variable ρ_{env} is included to perform the formula for various environments automatically. Featureless environments result in a small number of selected rays. Therefore, a high value of ρ_{env} is required and vice versa.

The third step of the "density based with scan segments"-method patches the results of the uniform sub-sampling. This is illustrated in Algorithm 1 lines 11 and 12. It selects additional rays if the differences of the indices is greater than a threshold ϵ_{RAY_Thres}. This occurs especially in narrow environments.

4.1.3 Scan Matching

The scan matching module (highlighted in blue) is based on the Random-Restart-Hill-Climbing (RRHC), also called Shotgun Hill Climbing [Russell and Norvig, 2009]. It is used to find the correct transformation $\vec{Tr} = (x, y, \theta)$ between the current scan and the existing map. This is performed in three steps: generate offspring, calculate fitness function, and compare result. First, the currently known best solution is used to generate randomly an offspring $\vec{Genome} = (D_x, D_y, D_\theta)$ where the indices x, y, and θ stand for the coordinates x and y and θ for the orientation.

Applying the fitness function illustrated in Algorithm 2 the offspring is compared to its parent. To determine the maximum $fitnessValue$ the local coordinates $[Ray_x^i, Ray_y^i]$ are calculated, see Algorithm 2 line 4 and 5. In the next step, the local coordinates are transformed to the global coordinate system and later called $[GRay_x^i, GRay_y^i]$. To do so, the robot pose $[R_x, R_y, R_\theta]$ as well as the generated random offspring $[D_x, D_y, D_\theta]$ are used. In case of an unknown cell $(Map[GRay_x^i][GRay_y^i] = 127)$ the algorithm decreases the $fitnesValue$. If the cell is known, the $fitnessValue$ is increased by summing the possibility of the cell and its neighbour cells, see Algorithm 2 line 11.

In case the offspring results in a better $fitnessValue$ than its parent it is used as the new parent for the next iteration. This procedure performs until N_{HC} iterations are proceeded or the best solution has been found $(fitnessValue < \epsilon_{fitness})$. The resulting values $D_{x,Best}$,

Algorithm 2 CRSM-SLAM - fitness function for Random-Restart-Hill-Climbing [Tsardoulias and Loukas, 2013]

procedure FITNESSFUNCTION()
  $fitnessValue = 0$
  for each $PickedIndices$ as i **do**
    $Ray_x^i = d_i \cdot \cos(Ray_\theta^i)$
    $Ray_y^i = d_i \cdot \sin(Ray_\theta^i)$
    $\begin{bmatrix} GRay_x^i \\ GRay_y^i \end{bmatrix} = \begin{bmatrix} \cos(R_\theta + D_\theta) - \sin(R_\theta + D_\theta) \\ \sin(R_\theta + D_\theta) + \cos(R_\theta + D_\theta) \end{bmatrix} \cdot \begin{bmatrix} Ray_x^i \\ Ray_y^i \end{bmatrix} + \begin{bmatrix} R_x + D_x \\ R_y + D_y \end{bmatrix}$
    if $Map[GRay_x^i][GRay_y^i] == 127$ **then**
      $fitnessValue = fintessValue - 100$
      continiue to next i
    end if
    $fitnessValue = fitnessValue$
      $+10 \cdot (255 - Map[GRay_x^i][GRay_y^i]$
      $+(255 - Map[GRay_x^i - 1][GRay_y^i])$
      $+(255 - Map[GRay_x^i + 1][GRay_y^i])$
      $+(255 - Map[GRay_x^i][GRay_y^i - 1])$
      $+(255 - Map[GRay_x^i][GRay_y^i + 1])$
  end for
end procedure

$D_{y,Best}$, and $D_{\theta,Best}$, are used to calculate the new robot pose by:

$$\vec{Trans} = \begin{bmatrix} R_x \\ R_y \\ R_\theta \end{bmatrix} = \begin{bmatrix} R_x + D_{x,Best} \\ R_y + D_{y,Best} \\ R_\theta + D_{\theta,Best} \end{bmatrix} \tag{4.11}$$

4.1.4 Map Update

The fourth module (highlighted in green) updates the map. The determined pose $\vec{Trans}$ is used to determine the robot‘s field of view. A raycaster is applied to select only the points in the map which are located in the current FOV.

To give the RRHC a better reference, points which are located in an occupied space are updated in a more intense way than points which are located in unoccupied space. This is proceeded by:

$$f(p_{cell}\text{‘}) = \begin{cases} p_{cell}\text{‘} = p_{cell} + (1.0 - p_{cell}) \cdot S_{spar} & \text{if} \quad p_{cell} = 0.5 \quad (free\ cell) \\ p_{cell}\text{‘} = p_{cell} - p_{cell} \cdot S_{spar} \cdot \rho_{occu} & \text{if} \quad p_{cell} \neq 0.5 \quad (occupied\ cell) \end{cases} \tag{4.12}$$

Here, p_{cell} is the previous value of the cell, S_{spar} is the mean spareness of the scan (see Equation 4.10), and ρ_{occu} is the coefficient to increase the update of an occupied cell ($\rho_{occu} > 1$).

4.2 HECTOR-SLAM

Heterogeneous Cooperating Team of Robots (HECTOR), from Kohlbrecher and Meyer [2015], is a ROS-package containing nodes (represented by rectangles in Figure 4.5) for 2D localization, mapping, autonomous exploration and navigation, as well as object tracking. To manage urban environments, HECTOR uses a 2.5D static mapping which does not respect changes in the environment. A common occupancy grid is combined with an elevation map. The experiments in Section 7 only apply the SLAM of the HECTOR-package.

That is why following section focuses on the mapping-node, the map server, and the pose-estimation-node, cf. Figure 4.6. This might be of interest when using additional sensors for localization (highlighted in brown): e.g. IMU, GPS, compass, etc.

4.2.1 Mapping-node

The Mapping-node (highlighted in yellow) is based exclusively on data supported by a 2D laser scanner, cf. Figure 4.6. Hence, no additional sensor is required (highlighted in brown). To provide a 3D pose and reflect 6-Degree-of-Freedom (DoF) for an improved mapping (highlighted in gray), additional sensors can be integrated. SLAM (highlighted in orange) and 3D-pose-estimation (highlighted in blue) are loosely coupled and synchronized over time. Best performance is achieved by exchanging information bidirectionally. To handle the 6-DoF motion of the robot, the laser scanner needs to be stabilized. HECTOR is using a right-handed system with following 3D-state representation

$$\vec{x} = \begin{pmatrix} \vec{\Omega}^T & \vec{p}^T & \vec{v}^T \end{pmatrix}^T \tag{4.13}$$

with the euler angles $\left(\vec{\Omega} = (\Phi, \Theta, \Psi)^T\right)$, the position $\left(\vec{p} = \begin{pmatrix} p_x, p_y, p_z \end{pmatrix}^T\right)$ and the velocity $\left(\vec{v} = \begin{pmatrix} v_x, v_y, v_z \end{pmatrix}^T\right)$.

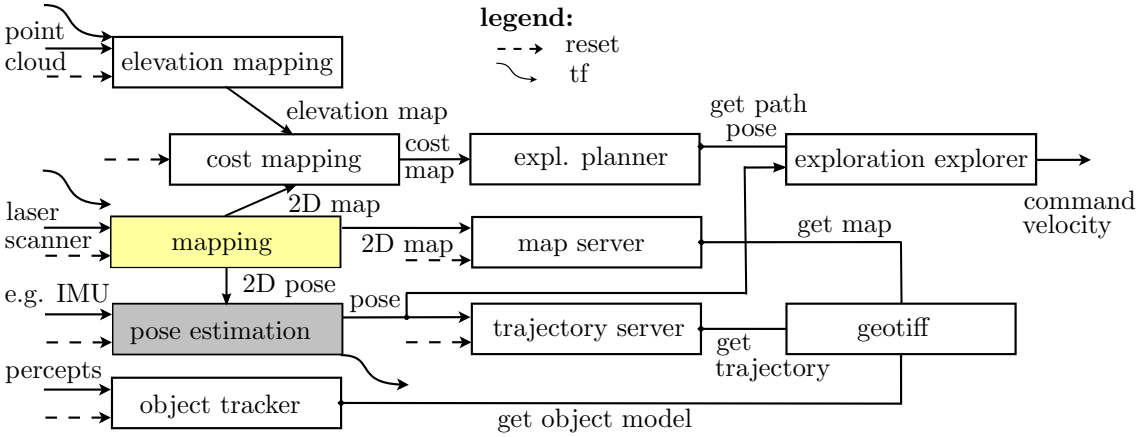


Figure 4.5: Procedure of the HECTOR-Package with its nodes to control a robot, reproduced from [Kohlbrecher et al., 2013].

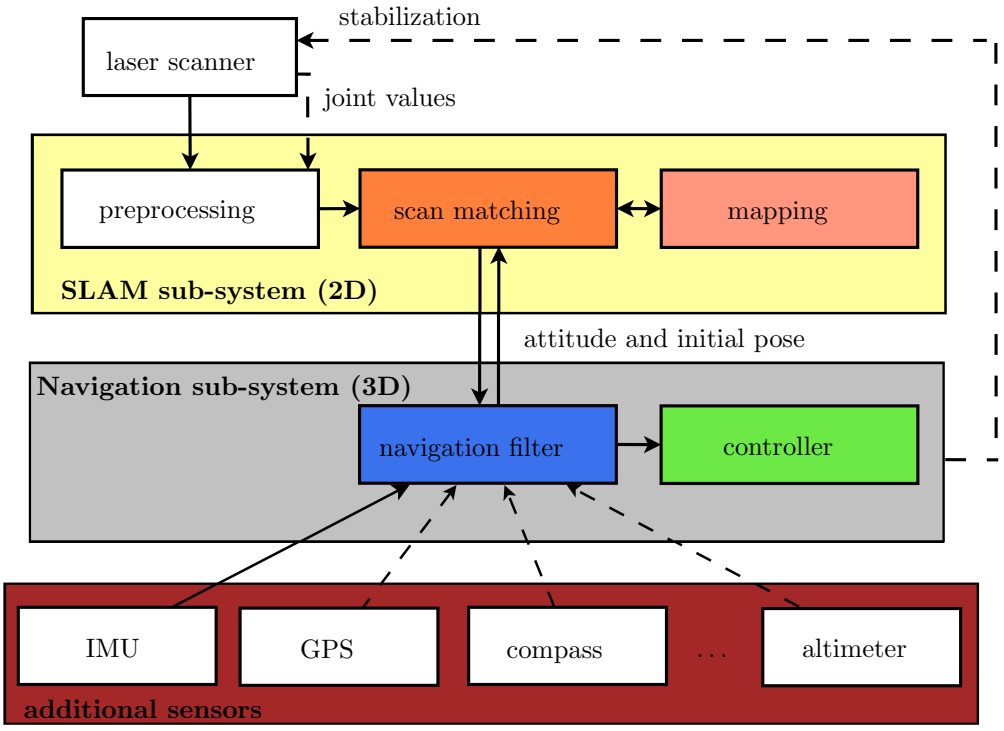


Figure 4.6: Procedure of HECTOR-mapping-node and HECTOR-pose-estimator-node, reproduced from [Kohlbrecher et al., 2011].

Map access:
To merge a new scan S with an existing map $\vec{P}$ (highlighted in orange), HECTORs approach downsamples both in multiple sub-grid resolutions. It starts to merge them with the lowest resolution. In contrast to common image processing approaches it keeps different scaled maps in memory. The maps are simultaneously updated and aligned. This prevents that the approach is stuck in a local minima.

To downsample them, the map coordinates $\vec{P}_m$, the occupancy value $M(\vec{P}_m)$, and the gradient

$$\nabla M(\vec{P}_m) = (\frac{\partial M}{\partial x}(\vec{P}_m), \frac{\partial M}{\partial y}(\vec{P}_m)) \tag{4.14}$$

are linear interpolated by the four closest integer coordinates $\vec{P}_{00..11}$, (cf. Figure 4.7). Hence, $M(\vec{P})$ follows as

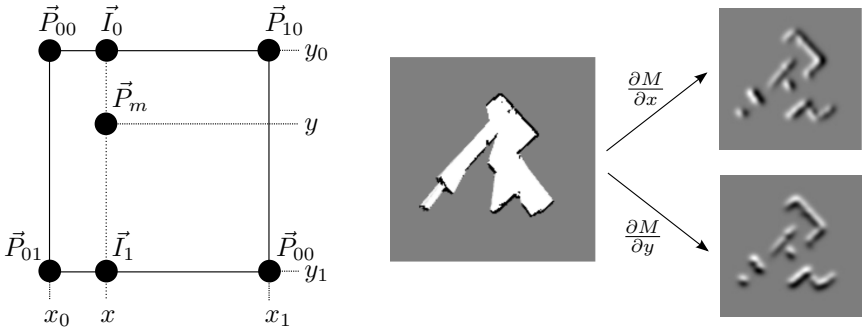
$$M(\vec{P}_m) \approx \frac{y - y_0}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0}M(\vec{P}_{11}) + \frac{x_1 - x}{x_1 - x_0}M(\vec{P}_{01})\right) + \frac{y_1 - y}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0}M(\vec{P}_{10}) + \frac{x_1 - x}{x_1 - x_0}M(\vec{P}_{00})\right) \tag{4.15}$$

with its derivatives

$$\frac{\partial M}{\partial x}(\vec{P}_m) \approx \frac{y - y_0}{y_1 - y_0}\left(M(\vec{P}_{11}) - M(\vec{P}_{01})\right) + \frac{y_1 - y}{y_1 - y_0}\left(M(\vec{P}_{10}) - M(\vec{P}_{00})\right) \tag{4.16}$$

$$\frac{\partial M}{\partial y}(\vec{P}_m) \approx \frac{x - x_0}{x_1 - x_0}\left(M(\vec{P}_{11}) - M(\vec{P}_{10})\right) + \frac{x_1 - x}{x_1 - x_0}\left(M(\vec{P}_{01}) - M(\vec{P}_{00})\right). \tag{4.17}$$

To simplify the equation for gradient approximation the regular grid with distance of 1 (for map coordinates) is chosen.



(a) Points $\vec{P}_{00..11}$ are interpolated to result point $\vec{P}_m$ for a bilinear filtering of the occupancy grid.

(b) Illustration of occupancy grid and its spatial derivatives.

Figure 4.7: a) Bilinear filtering of the occupancy grid map. Point $\vec{P}_m$ is the point whose value shall be interpolated. b) Spatial derivatives of an occupancy grid map, reproduced from [Kohlbrecher et al., 2011]

Scan matching:
Scan matching is based on a Gauss-Newton approach [Lucas and Kanade, 1981] which fits the laser end points with the existing map. Thus, data associated search is not required. The existing map and the scan get aligned by minimizing the rigid transformation $\vec{\xi} = \left(p_x, p_y, \psi\right)^T$ as

$$\vec{\xi}^* = \underset{\xi}{\operatorname{argmin}} \sum_{i=1}^{n} [1 - M(\vec{S}_i(\vec{\xi}))]^2. \tag{4.18}$$

with $\vec{S}_i(\vec{\xi})$ are world coordinates of scan end points $\vec{s_i} = \left(s_{i,x}, s_{i,y}\right)^T$. They depend on $\vec{\xi}$

$$\vec{S}_i(\vec{\xi}) = \begin{pmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{pmatrix} \begin{pmatrix} s_{i,x} \\ s_{i,y} \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix}. \tag{4.19}$$

The function $M(\vec{S}_i(\vec{\xi}))$ returns the map value at the coordinates given by $\vec{S}_i(\vec{\xi})$, with a start estimation of $\vec{\xi}$ to estimate $\Delta\vec{\xi}$. This is used to minimize the error. Therefore,

$$\sum_{i=1}^{n} [1 - M(\vec{S}_i(\vec{\xi} + \Delta\vec{\xi}))]^2 \to 0 \tag{4.20}$$

using the first Taylor expansion of $M(\vec{S}_i(\vec{\xi}) + \Delta\vec{\xi})$ the equation can be described by

$$\sum_{i=1}^{n} \left[1 - M(\vec{S}_i(\vec{\xi})) - \nabla M(\vec{S}_i(\vec{\xi})) \frac{\partial \vec{S}_i(\vec{\xi})}{\partial \vec{\xi}} \Delta\vec{\xi}\right]^2 \to 0. \tag{4.21}$$

Further, the partial derivative w.r.t. $\Delta\vec{\xi}$ is set. Thus, it follows

$$2 \cdot \sum_{i=1}^{n} \left[\nabla M(\vec{S}_i(\vec{\xi})) \frac{\partial \vec{S}_i(\vec{\xi})}{\partial \vec{\xi}}\right]^T \cdot \left[1 - M(\vec{S}_i(\vec{\xi})) - \nabla M(\vec{S}_i(\vec{\xi})) \frac{\partial \vec{S}_i(\vec{\xi})}{\partial \vec{\xi}} \Delta\vec{\xi}\right] = 0. \tag{4.22}$$

Finally, $\Delta\vec{\xi}$ is minimized by using the Gauss-Newton equation

$$\Delta\vec{\xi} = \vec{H}^{-1} \sum_{i=1}^{n} \left[\nabla M(\vec{S}_i(\vec{\xi})) \frac{\partial \vec{S}_i(\vec{\xi})}{\partial \vec{\xi}}\right]^T \cdot [1 - M(\vec{S}_i(\vec{\xi}))] \tag{4.23}$$

with

$$\vec{H} = \left[\nabla M(\vec{S}_i(\vec{\xi})) \frac{\partial \vec{S}_i(\vec{\xi})}{\partial \vec{\xi}}\right]^T \cdot \left[\nabla M(\vec{S}_i(\vec{\xi})) \frac{\partial \vec{S}_i(\vec{\xi})}{\partial \vec{\xi}}\right]. \tag{4.24}$$

Using the Equations 4.14 and 4.19 the approximation follows to

$$\frac{\partial \vec{S}_i(\vec{\xi})}{\partial \vec{\xi}} = \begin{pmatrix} 1 & 0 & -\sin(\psi) \cdot s_{i,x} - \cos(\psi) \cdot s_{i,y} \\ 0 & 1 & \cos(\psi) \cdot s_{i,x} - \sin(\psi) \cdot s_{i,y} \end{pmatrix}. \tag{4.25}$$

Using $\nabla M(\vec{S}_i(\vec{\xi}))$ and $\frac{\partial \vec{S}_i(\vec{\xi})}{\partial \vec{\xi}}$ the Gauss-Newton equation 4.23 is evaluated step by step to a minimum $\Delta\vec{\xi}$. It has to be mentioned that the minimum cannot be guaranteed because of the non-smooth linear approximations of the map gradient $\nabla M(\vec{S}_i(\vec{\xi}))$. The covariance is approximated by

$$R = Var\xi = \sigma^2 \cdot H^{-1} \tag{4.26}$$

where σ is a scaling factor depending on the laser scanner properties.

4.2.2 Pose-Estimation

To build elevation maps, a 6-DoF pose estimation is required. That is why HECTOR-pose-estimator-node (cf. Figure 4.5) applies measurements form an IMU. It processes an Extended-Kalman-Filter (EKF) to determine the pose.

The IMU delivers the input vector $\left(\vec{u} = \begin{pmatrix}\vec{w}^T & \vec{a}^T\end{pmatrix}^T\right)$ with the angular rates $\left(\vec{w} = \begin{pmatrix}w_x, w_y, w_z\end{pmatrix}^T\right)$ and the accelerations $\left(\vec{a} = \begin{pmatrix}a_x, a_y, a_z\end{pmatrix}^T\right)$. Hence, the system is described by a non-linear differential equation system

$$\dot{\vec{\Omega}} = \vec{E}_\Omega \cdot \vec{w} \tag{4.27}$$

$$\dot{\vec{p}} = \vec{v} \tag{4.28}$$

$$\dot{\vec{v}} = \vec{R}_\Omega \cdot \vec{a} + \vec{g} \tag{4.29}$$

with direction cosine matrix $\vec{R}_\Omega$ transfers from the body frame in the navigation frame and $\vec{E}_\Omega$ transfer body fixed angular rates to the derivatives of the Euler angles with the gravity vector $\vec{g}$ [Kuipers, 1999].

The resulting 6-DoF pose is independently calculated from the 2-DoF pose. Nevertheless, if a 6-DoF pose is available, it is projected into the 2D plane and used as a start estimation for the scan matching. As previously mentioned, this rises performance.

It can also be used for other mapping modules which do not support a localization (e.g. OctoMap).

4.3 OctoMap

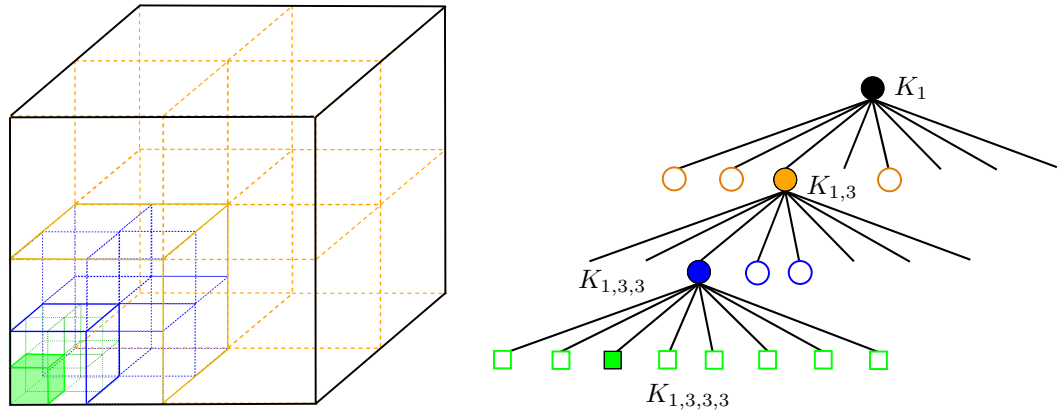
Similar to HECTOR, OctoMap is a framework containing a ROS-package with several nodes regarding octree based 3D mapping. It was presented by Wurm et al. [2010] and successively extended [Hornung et al., 2010, 2013; Wurm et al., 2011].

OctoMap-Mapping-node is solely a mapping node without any localization. This is in contrast to the other presented mapping approaches (SLAM approaches). Hence, OctoMap requires an external localization-node. The ROS-package supplies such a node for 3D localization [Hornung et al., 2010]. It is based on 2D laser scans, joint encoders, as well as an IMU. Since only a laser scanner is available in this work, this node is not applicable and therefore not considered further.

The OctoMap-Mapping-node applies an octree representation to store measured data. Each point is stored in a cube (voxel) of predefined size. Exemplarily, Figure 4.8 illustrates a voxel with its sub-divisions and illustrates the corresponding tree. Each voxel (parent) consists of eight sub-divisions (children) and six neighbours. The children convert to a parent if it is further fragmented. In the tree representation parents are illustrated as circles. The parent contains solely information of its children (eight pointers). Only the children in the end of the tree (the leaf) carry information about occupancy, colour, temperature, or other properties. It represents the smallest voxel and therefore the resolution. Children of occupied voxels are pictured as green squares, children of free voxels as white squares, and children of unknown voxels (not discovered area) as a simple line. To minimize storage requirements, OctoMap applies solely one pointer for parent. This pointer leads to an array which contains the pointers to the eight children. It is initialized as soon as a child is required. In this case, the old child

transfers to a new parent. The depth of the tree as well as the resolution of the map rises. Due to the structure octrees allow a fast and efficient point search.

The OctoMap-Mapping is optimized to perform four requirements: full 3D model representation, updatable, flexible, and compact. For a full 3D representation occupied, free, as well as unknown areas need to be modeled in 3D. The resolution depends on the voxelsize. Especially, the distinction between free and occupied space is fundamental to assure safe navigation and autonomous exploration. Classically, this is achieved by a boolean mask for initialized voxels (free or occupied). Uninitialized voxels are simply determined to be unoccupied space. However, OctoMap applies a probabilistic model for occupancy which gets compared to a threshold to rate occupied or free voxels.



(a) Location of voxel $K_{1,3,3,3}$ in voxel space K_1. The searched, occupied voxel $K_{1,3,3,3}$ is marked by a green square. Unoccupied voxels are marked by white squares. Undiscovered voxels are marked by white circles.

(b) Octree search example for voxel $K_{1,3,3,3}$ starting at K_1.

Figure 4.8: Example of voxel representation of the TSD-SLAM with a search tree for voxel $K_{1,3,3,3}$.

This leads to the second requirement: updatable. The probabilistic model allows to respect changes in environment as well as sensor noise because occupancy is not answered with a simple yes or no. The fusion of multiple measurements over time results a likelihood to be occupied or free. This makes it possible to remove disappeared objects and enter new ones. Such an update should be proceeded quickly. Otherwise, the robot relies on old data. As a drawback this leads to "unstable" maps for noisy sensor measurements. The occupied voxels will "jump" due to different values. To prevent this, a point measured k-times occupied should be measured k-times unoccupied to erase it. This demand stands in contrast to the dynamic mapping which claims for quick changes. Hence, an upper and lower limit is implemented in OctoMap.

The third requirement demands flexibility which stands for dynamical extension of the map size as well as multi-resolution maps. Both should be adjustable during mission based on the current requirement. This leads to efficient memory and disk usage as they are extended only when required.

Flexibility goes hand in hand with the last requirement: compactness. It means that the map should demand minimal storage in memory and on disk. As a result, resources are available for other processes. Further, transfer of data requires minimal bandwidth. To achieve this, OctoMap fuses voxels based on their occupancy probability. If all children of a voxel carry the same probability and the values considered are stable, the children are fused. They are erased

and the parent voxel carries the probability as a child of bigger size. This results in voxels of varying sizes. Only if new measurements occur and the probability value changes the voxel gets fragmented again. This refinement is restricted to a pre-defined depth.

Figure 4.9 illustrates the basic flow chart of the OctoMap-Mapping-node. In the first step (highlighted in red), the algorithm receives a point cloud s_{cloud} from a 3D-sensor. It is necessary that the utilized sensor is modeled in the following step (raycast).

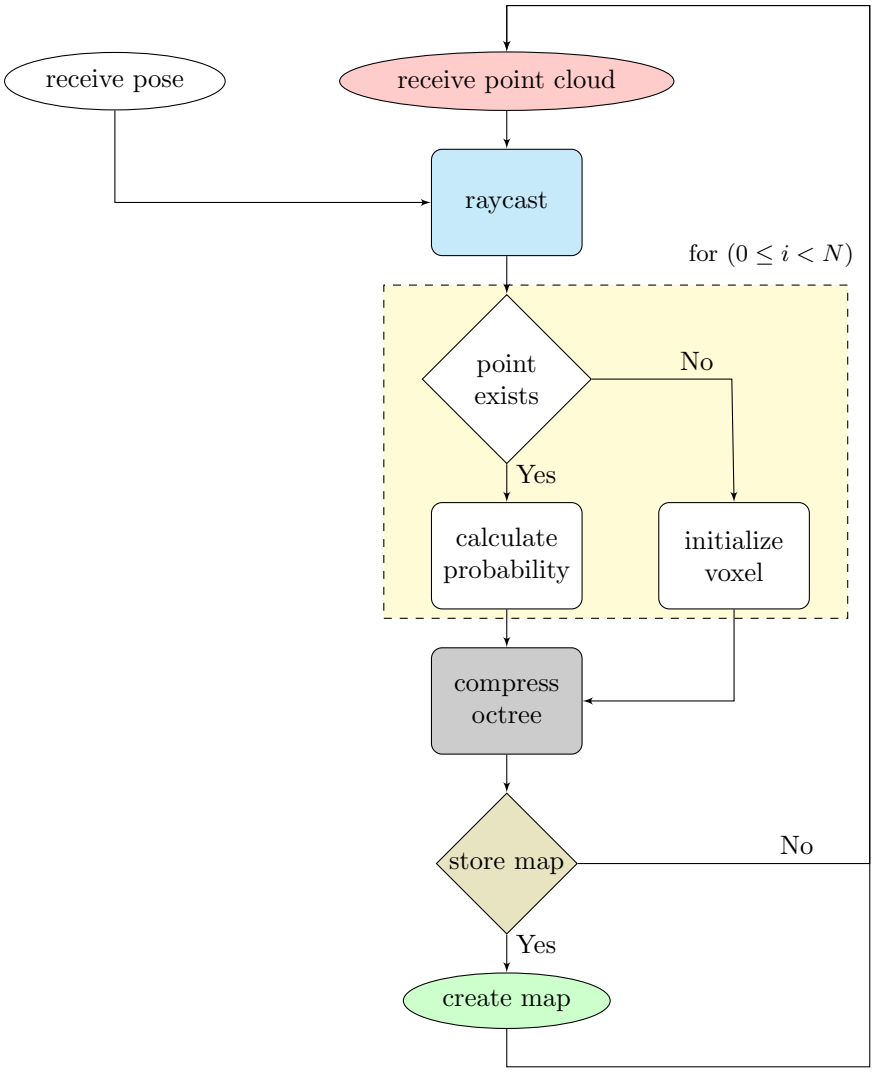


Figure 4.9: Flow chart of OctoMap-Mapping-node.

The second step (highlighted in turquoise), requires the current pose $\vec{T} = (\vec{Tr}, \vec{R})$ with its current position $\vec{Tr} = (x, y, z)$ and orientation $\vec{R} = (\Phi, \Theta, \Psi)$ to build a model m_{model} from the existing map. This model illustrates the view of the robot from this pose regarding the map. Building such a model is achieved by raycasting. From the current position, a ray is sent out, similar to the applied sensor measuring the environment. This is why the sensor model is required. In the existing map, the ray passes each voxel on its way. If the voxel is empty, it

continues to the next voxel. If the voxel is occupied, it is included into the model. The resulting model represents the environment which the robot assumes to see.

The model m_{model} and the scene s_{cloud} are fused in the third step (highlighted in yellow). To do so, for each point p_i $(i \in 0, \ldots, N)$ in the point cloud the corresponding voxel in the model m_{model} is searched. If the voxel is not initialized, the point represents a new area. It gets created and it is assigned to be occupied. To do so, OcotMap does not use the probability directly, but a log-odds notation:

$$L(n) = \log\left[\frac{P(n)}{1 - P(n)}\right] \tag{4.30}$$

This notation simplifies the calculation for existing points as it changes a multiplication to a summation. If the voxel exists, the new occupancy probability at the time t is determined by

$$L(n|z_{1:t}) = \max(\min(L(n|z_{1:t-1}) + L(n|z_t), l_{max}), l_{min}). \tag{4.31}$$

As previously mentioned, the restrictions l_{min} and l_{max} ensure a quick adaptation for dynamic environments. The snippet "$L(n|z_{1:t-1}) + L(n|z_t)$" calculates the new probability by a simple summation of the new value $L(n|z_t)$ and the previous value in the voxel $L(n|z_{1:t-1})$. Written with probabilities, the new value also can be determined by

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \cdot \frac{1 - P(n|z_{t-1})}{P(n|z_{t-1})} \cdot \frac{P(n)}{1 - P(n)}\right]^{-1}. \tag{4.32}$$

In the next step (highlighted in gray), the octree is compressed for minimal memory and disk requirement. This is performed if all children of a parent can be considered to be stable. This is assumed if l_{max} or l_{min} is reached. Therefore, the eight children (of a leaf) are fused together and the parent carries the probability. It reduces the required data to $\sim 1/8$. It also fastens the raycast as it demands less computing time to validate the emptiness of a huge voxel in compare to multiple small voxels.

In the last step (highlighted in olive), the map or rather maps are stored. Due to the structure of the octree it is easy to save individual objects and various resolutions, as illustrated in Figure 4.10. The environment is mapped with a low resolution as it is not important for the robot. In contrast, the table has a higher resolution to determine the surface of the table. Objects on the table are modeled with the highest resolution that the robot is capable to perform gripping tasks. The resolution corresponds to the chosen depth of the tree while the objects correspond to its branch.

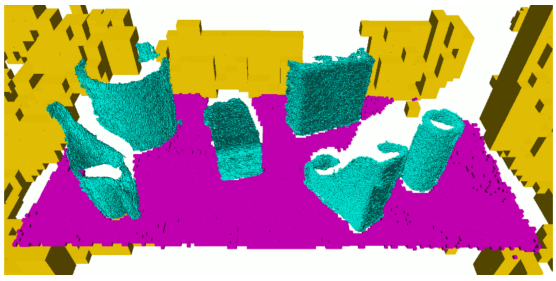


Figure 4.10: Example of a multi-resolution OctoMap Hornung et al. [2013].

4.4 TSD-SLAM

TSD-SLAM [Koch et al., 2015a; May et al., 2014] uses depth measurements from arbitrary 2D and 3D sensor units to build a map based on the signed distance function. The underlying framework generalizes the KinectFusion-approach (KinFu) [Newcombe et al., 2011] with an object-oriented model to respect different sensor modalities. For instance, measurements of laser scanner, ToF-sensors and RGB-D cameras are integrated into the same representation. To respect these various sensor models, a general sensor class is implemented.

This solves two drawbacks of KinFu. First, different kind of sensors can be applied if their sensor model is implemented in the class. It allows multiple sensors to insert measurements into the same map. These sensors can be attached to one or multiple robots. This makes TSD-SLAM also compsatible for multi-robot-SLAM [Koch et al., 2015b, 2016].

Second, sensor models also respect sensors with a wider viewing range, e.g. 3D laser scanner with 360° FOV. This reduces the problem of matching huge planes, as in KinFu, because the wider FOV increases the chance of structures. Another difference, in compare to KinFu, is the fact that TSD-SLAM employs a power-saving CPU instead of a GPU.

In compare to other SLAM algorithms, the current scan is not matched to a previous but to the entire map. This reduces cumulative errors and therefore loop-closure is not required. It makes TSD-SLAM a dynamic mapping algorithm which respects changes in the environment. Due to visual localization via Iterative Closest Point (ICP) it does not require additional sensors. Nevertheless, it is possible to use pose information, e.g. supported by IMU, odometry, or dead reckoning, as a start estimation. This speeds up the approach and lowers the risk of stucking in a local minima.

The map of TSD-SLAM is based on voxels aligned in an octograph, similar to Wurm et al. [2010]. For 3D-mapping a cube has eight "neighbours" and can be split up into eight "children" as soon as a point is located in it. For 2D-mapping it simplifies to four "neighbours" and each cell can be split to four "children". The splitting is done until a maximum resolution is achieved. This allows a fast access as the graph based arrangement does not require to search all points. To accelerate the TSD-SLAM and reduce computing power, voxels of different size are generated. Only if a measurement is located in an empty voxel, smaller partitions are allocated.

Algorithm 3 illustrates the basic procedure of TSD-SLAM. As soon as new sensor data D_{sensor} are available, the function *RAYCAST()* creates a model from the map. This is based on the previous location. The new sensor scene s_{scene} is matched to the model m_{model} by an ICP algorithm [Besl and McKay, 1992]. If the sensor movement is greater than a defined threshold ϵ_{move}, the function *PUSH()* inserts the scene into the existing map.

Algorithm 3 Registration and integration of new measurement data.[May et al., 2014]

1: **procedure** onSensorDataReveive(D_{sensor})
2: $m_{model} \leftarrow$ rayCast(D_{sensor}) ▷ Algorithm 4
3: $s_{scene} \leftarrow$ get data from sensor
4: $\mathbf{T}_{icp} \leftarrow$ icp registration of model m_{model} and scene s_{scene}
5: $\mathbf{T}_{sensor} \leftarrow \mathbf{T}_{icp}\mathbf{T}_{sensor}$ ▷ update sensor pose
6: **if** $(\mathbf{T}_{sensor}\mathbf{T}^{-1}_{last\_push} > \epsilon_{move})$ **then**
7: $\mathbf{T}_{last\_push} \leftarrow \mathbf{T}_{sensor}$
8: push(D_{sensor}) ▷ Algorithm 6
9: **end if**
10: **end procedure**

4.4.1 Create a Model by Raycasting

Raycasting is the first step after new sensor data have been received. Based on an initial position, the raycaster, illustrated in Algorithm 4, delivers data from the current map for the matcher. It has a generic sensor model to be compatible with different types of sensors (e.g. with pinhole model, polar model, etc.). Subsequently, it runs through the TSD-Space to identify sign changes in the TSD-Function $f(d, \rho, \epsilon_{noise})$ between neighbours. The position of the sensor $\vec{p}$, the center of the arbitrary element $\vec{v}$, and the distance measurement m in direction of the given element are considered to result $d(\vec{v})$ to

$$d(\vec{v}) = m - |\vec{p} - \vec{v}|. \tag{4.33}$$

Respecting the penetration depth ρ and the sensor noise ϵ_{noise} granularity can be distinguished. Applying this to the exponential model, the TSD-function $f(d, \rho, \epsilon_{noise})$ can be described by

$$f(d, \rho, \epsilon_{noise}) = \begin{cases} 1 & \text{if} \quad d \geq -\epsilon_{noise} \\ e^{-\sigma(d-\epsilon_{noise})^2} & \text{if} \quad (d < -\epsilon_{noise}) \wedge (d > -\rho) \\ 0 & \text{if} \quad d < -\rho \end{cases} \tag{4.34}$$

For elements which are not visible, $f(d, \rho, \epsilon_{noise})$ has a negative value (behind a wall). In the area of $(d < -\epsilon_{noise}) \wedge (d > -\rho)$, the value lowers the closer the object is (in front of a wall). The point (wall) is located at the zero crossing. For explored space which is unoccupied $f(d, \rho, \epsilon_{noise}) = 1$. It returns coordinates as well as normals of the identified point. One needs to mention that the raycaster benefits in speed as the TSD-SLAM is based on generic voxels.

Algorithm 4 Raycasting through TSD-Space[May et al., 2014]

```
1:  procedure RayCast(D_sensor)
2:      R ← get rays from D_sensor
3:      for each ray r⃗ ∈ R do
4:          V ← first element in TSD-Space along r⃗
5:          tsd_prev ← NAN
6:          while (V is inside TSD-Space) do
7:              tsd ← tsd_V                          ▷ assign property of V
8:              if ((tsd ≤ 0) ∧ (tsd_prev > 0)) then
9:                  c⃗ ← extract coordinates for TSD-Space
10:                 n⃗ ← extract normals for TSD-Space
11:                 break
12:             end if
13:             tsd_prev ← tsd
14:             V ← next element in TSD-Space along ray
15:         end while
16:     end for
17: end procedure
```

4.4.2 Determine Movement of Sensor

The movement of the scanner w.r.t. the environment (map) is determined by ICP registration, illustrated in Algorithm 5. As stated above the modular design of TSD-SLAM allows an exchange with other matching algorithms, e.g. Normal Distribution Transform (NDT)[Biber and Strasser, 2003; Magnusson, 2009]. Since it was not used for the experiments of this thesis, it is not discussed further.

Algorithm 5 TSD-Integration of new data.[May et al., 2014]

procedure DATAINTEGRATION(D_{sensor})
  $scene \leftarrow scan$
  $\mathbf{T}_{ICP} \leftarrow ICP(scene_{prev}, scene)$
  $\vec{\lambda} \leftarrow$ calculate Eigen vales for $\mathbf{T}_{ICP}$
  if (all eigenvalues $\|\lambda_i - 1\| < \lambda_{th}$) **then**
    $\mathbf{T}_{ICP} \leftarrow ICP(model, scene)$
    $\mathbf{T}_{Pose} = \mathbf{T}_{ICP}\mathbf{T}_{Pose}$
    $model \leftarrow scan$
  end if
  $scene_{prev} \leftarrow scan$
end procedure

During matching, TSD-SLAM faces multiple interests: process a huge data flow to support poses as precise as possible, assure reliable matching, and return fast matching results. It is understood that the first two interests stand in contrast to the third. That is why for a new scan the movement of the sensor is limited. To minimize the data flow, a greater movement of the scanner is required. To achieve precise pose information and prevent mismatches, a maximal movement of the scanner is allowed. The matching speed can be accelerated by using additional sensors (e.g. IMU) to support a start estimation. For the matcher, this also reduces the danger of stucking in a local minima.

4.4.3 Integration of New Data

New data is inserted into the TSD-Space as soon as a signifiacant movement is detected (see Algorithm 3). Then the new scan is matched to the model, as illustrated in Algorithm 6. All measurements are masked to indicate valid or invalid values, e.g. if they are out of measuring range or have low reflectivity values.

Before inserting new values a back-projection is proceeded. Old data in the map are fused with new data from the sensor. This procedure allows a dynamic mapping. If a measurement is smaller than the back-projected point, a new point is entered. This is due to the fact that the new point is located in front of the old point and therefore a new object appeared. If a measurement is greater than the back-projected point, then the old object has disappeared. If the measurement equals an existing point, it confirms this point. In any way, the TSD value is updated. Since the update is based on an averaging process changes in the environment need some time to be inserted into the map. On the other hand, it reduces impacts of incorrect measurements.

Algorithm 6 TSD-Integration of generic sensor.[May et al., 2014]

procedure PUSH(D_{sensor})
  $\vec{p} \leftarrow$ get current position from D_{sensor}
  $data \leftarrow$ get data from D_{sensor}
  $mask \leftarrow$ get mask from D_{sensor}
  for (each element V in TSD-Space) **do**
    $\vec{v} \leftarrow$ obtain coordinates of V
    $idx \leftarrow$ project $\vec{v}$ back to measurement index
    if ($mask[idx]$) **then**
      $distance \leftarrow \|\vec{p} - \vec{v}\|$
      $d \leftarrow data[idx] - distance$
      if ($d \geq -\rho$) **then**
        $tsd \leftarrow min(\frac{d}{\rho}, 1.0)$
        $w \leftarrow f(d, \rho, \epsilon_{noise})$
        $tsd_V \leftarrow \frac{tsd_V \cdot w_V + tsd \cdot w}{w_V + w}$
        $w_V \leftarrow w_V + w$
      end if
    end if
  end for
end procedure

4.4.4 Modifications at TSD-SLAM for Experiments in Chapter 7

During the experiments the TSD-SLAM was modified. This allows an update of the map in case of specular reflective or transparent object occurrence [Koch et al., 2017b]. It demonstrates that two separate mapping stages (one for the 2D-Pre- and one for the 2D-Post-Filter) as it was proceeded at the 2D-Mirror-Detector-Approach V1 can be prevented. Besides, an updated map allows the robot to navigate on most reliable information. The detailed description of the modifications are described in Section 6.1.5.

Chapter 5

State-of-the-Art of Reflection Recognition

Influences from transparent and specular reflective objects are not a new phenomena in mapping. Nevertheless, since 2D-mapping was sufficient in the past, it was easier to handle such disturbing objects. Covering all transparent or specular reflective objects is still a common practice to prevent them. It is understood that this is unwanted since it changes the "real" environment. Besides, it gets difficult or even impossible when 3D mapping is applied. This would require to manipulate the entire environment. For 2D only the objects which are at the scan plane level need to be covered. Imagine a disaster area where rescue teams need to go through the entire area to prepare it for a rescue robot to search for victims. In such a case a robot will not help to save time of searching for victims but consume time. Furthermore, it will not secure the work of the rescue teams because the first ones entering the hazard zone are still the rescue troops.

This chapter outpoints work concerning transparent and specular reflective influences. It is divided into three sections: stationary systems, mobile systems, and a summary. The first section covers stationary systems, even it is understood that this represents an ideal case. Most methods have less impact on this work as they demand a predefined scene or huge instruments which cannot be used at mobile robots. The second section covers mobile systems. It is split into multiple sub-sections which describe different methods. The last section summarizes mobile approaches, outpoints the drawbacks, and describes the differences to the approach presented in Chapter 6.

5.1 Stationary Systems

To prevent the need to cover disturbing objects, Ihrke et al. [2010] presents multiple methods for stationary systems to reconstruct transparent and specular reflective objects as well as influences caused by fire, smoke, and interstellar nebulae. They classify image based on the object type (opaque, translucent/transparent, and inhomogeneous) which describes the material property, cf. Figure 5.1. These object types are further sub-classified based on their surface and volume property. Hence, an image formation is achieved which describes the different influences on light.

object type	surface / volume type	class	image formation
opaque	surface, rough	1	diffuse or near diffuse reflectance
	surface, glossy	2	mixed diffuse and specular reflectane
translucent / transparent	surface, smooth	3	ideal or near ideal specular reflectance
	surface, sub-surface scattering	4	multiple scattering underneath surface
	surface, smooth	5	ideal or near ideal specular refraction
	volume, emission / absorbtion	6	integration along viewing ray
	volume, single scattering	7	integration along viewing ray
	volume, multiple scattering	8	full global light transport without occluders
inhomoge-neous	mixed scenes, containing many / all of the above	9	full global light transport

Figure 5.1: Classification based on surface type, volume type, and resulting effect, reproduced from [Ihrke et al., 2010]. The complexity rises with the rising class number.

In Figure 5.2, Ihrke et al. [2010] assign multiple approaches to their application field concerning their light influence and the assigned classes. This overview illustrates that there is no overall technique for stationary systems to cover all effects. According to the authors, algorithms are still very specific and not generally applicable.

Stationary systems represent an ideal case for measuring as the environment is well known. The environment as well as the sensor system can be prepared for precise data acquisition. The work of Ihrke et al. [2010] outlines the difficulties for known environments. Further, it describes a comprehensive overview of algorithmic methods and sensor set-ups to identify transparent and specular reflective objects and influences. It is understood that an unknown environment, as it is present for mobile robots, e.g. at a rescue scenario, is more challenging.

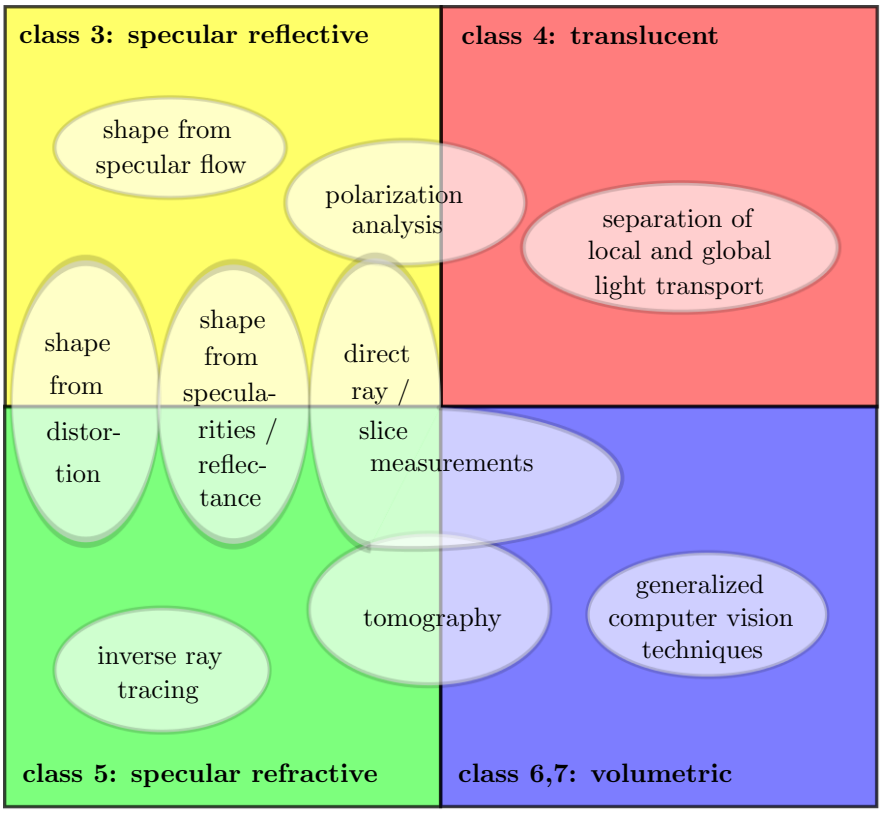


Figure 5.2: Overview of algorithms based on their application area based on a classification in surface and volume, reproduced from [Ihrke et al., 2010].

5.2 Mobile Systems

This section describes approaches concerning transparent and specular reflective influences for mobile robot systems. According to the application field (window detection in façades, 2D mapping, 3D mapping) and applied sensor system it splits into:

- Window detection in façades with solely an RGB-camera
- Window detection in façades with a laser scanner and an RGB-camera
- Window detection in façades with solely a laser scanner
- Laser scanner fused with ultrasonic sensor for 2D mapping
- Selective fusion of laser scanner with ultrasonic sensor for 2D mapping
- Mirror detection based on symmetry for 2D mapping
- Laser-based glass detection based on a density functions
- Visible Angle Grid for Glass Environments (VisAGGE)
- Glass detection based on the incident angle

- Glass detection by respecting different scan locations
- 3D mirror detection by jumping edge detection in panorama images
- Transparent object reconstruction in 3D

The first field, window detection in façades, applies mainly in the field of mapping for virtual tourism, urban planing, or cultural documentations. The focus is different, but it overlaps with 3D mapping.

5.2.1 Window Detection in Façades with Solely an RGB-Camera

Several window detection approaches are presented in the field of virtual tourism, urban planing, and cultural documentations. It is obvious to think they have a huge impact when talking about transparent object and mapping. Unfortunately, most of them are RGB-based, require façades to detect the window locations, or do not consider mapping. For example, Recky and Leberl [2010] presented an approach, using a k-mean color clustering method. Their approach analyses façades in CIE-LAB color space. After, a plane fitting algorithm models the façades. Its purpose relies in identification of levels containing windows, and detection of windows of a complex historical façades as well as modern façades. Another RGB-based approach from Ali et al. [2007] is based on a multi-scale Haar wavelet representation [Fujinoki, 2015]. After the wavelet representation is created, an adaptive Gentle Adaboost classifier is applied to identify windows.

Both approaches require façades to determine windows. That is why they are limited as they cannot be used inside a building where transparent and specular reflective objects also occur. They also are not designed to distinguish between the different objects, neither free standing objects, nor unframed objects. Besides, no laser scanner is applied as it is desired due to its advantages.

5.2.2 Window Detection in Façades with a Laser Scanner and an RGB-Camera

In later work, Ali et al. [2008] presented another approach to detect windows in façades by using a laser scanner and an RGB-camera. By using an adaptive threshold filter, the distance values of the laser scanner are searched for variations. These variations are assumed to be the result of specular reflective influences. In a second step, the windows are extracted by determining the borders in the RGB data. Due to the fact that it desires façades, it does not apply as it does not identify transparent and specular reflective objects inside a building. Further, it requires an additional RGB-camera.

5.2.3 Window Detection in Façades with Solely a Laser Scanner

Pu and Vosselman [2007] implemented a window extracting algorithm based solely on a laser scanner to reconstruct façades and classify building features (windows, walls, roofs, doors, extrusion, ...). In a previous paper, they presented an approach to detect walls, roofs, doors, and extrusions based on features like size, position, direction, and topology [Pu et al., 2006]. In their newer research they demonstrate that their previous approach cannot be applied to windows. This is due to the minimal amount of measurement points on the window surface as well as the minimal amount of points on the window framing. Even when the windows have curtains, the feature detection is not reliable. That is why they apply an hole-based extraction method.

To do so, for each wall segment, a triangulated irregular network (TIN) is generated. Outer boundaries (wall corners) and inner boundaries (holes) are identified based on the length of the TIN edge. In compare to outer boundaries which have two neighbouring triangles, inner boundaries have three neighbouring triangles. Next, the inner boundaries are clustered, considering the corresponding opening of the façade, in five steps:

1) Choose boundary point $\vec{p}_A$ which has no label value yet and label with integer value i.

2) Find all the long TIN edges which connect to this $\vec{p}_A$ and determine the end point $\vec{p}_B$ of the connected TIN.

3) If $\vec{p}_B$ is not labelled, label $\vec{p}_B$ with value i.

4) Set $\vec{p}_B$ as a new $\vec{p}_A$ and iterate step 2) to 4) until no more unlabelled points of the triangle are left.

5) Choose another unlabelled boundary point $\vec{p}_A$, set it to $i+1$, and repeat step 2) to 5) until all boundary points are labelled.

Next, holes resulting from doors or extrusions are filtered out. This also applies to small holes which are extremely long and narrow or which are very small. It is assumed that they result from noise. Finally, rectangles are fitted to the remaining holes by identifying the most left point $\vec{p}_{left}$, most right point $\vec{p}_{right}$, most top point $\vec{p}_{top}$, and most bottom point $\vec{p}_{bottom}$. Pu and Vosselman [2007] assume: windows are rectangles, left and right boundaries are aligned vertically to the ground, and walls are vertical. That is why

$$\begin{aligned}
\vec{c}_{TopLeft}(x,y,z) &= (x_{\vec{p}_{left}}, y_{\vec{p}_{left}}, z_{\vec{p}_{top}}) \\
\vec{c}_{TopRight}(x,y,z) &= (x_{\vec{p}_{right}}, y_{\vec{p}_{right}}, z_{\vec{p}_{top}}) \\
\vec{c}_{BottomLeft}(x,y,z) &= (x_{\vec{p}_{left}}, y_{\vec{p}_{left}}, z_{\vec{p}_{bottom}}) \\
\vec{c}_{BottomRight}(x,y,z) &= (x_{\vec{p}_{right}}, y_{\vec{p}_{right}}, z_{\vec{p}_{bottom}})
\end{aligned}$$

For windows which are not rectangular, a minimum bounding rectangle is fitted. The algorithm identifies and classifies windows in façades, but does not consider transparent and/or specular reflective influences in laser scans. No distiction between these influences is made and erroneous measurements are not filtered. This is why it is not applicable in a rescue scenario.

Also, Wang et al. [2010] present a window detection algorithm based solely on a laser scanner, illustrated in Figure 5.3. It contains a façade detection step (highlighted in light blue) and a window detection step (highlighted in light green). Façades of buildings are detected by a bottom-up and bottom-down method. First, data are sub-sampled and filtered in a pre-processing sub-step (highlighted in dark blue). Then, ground points are separated from the laser scan by assuming that they have an elevation of less than ±0.2 m. Therefore, an elevation histogram of all points is used to define the ground height.

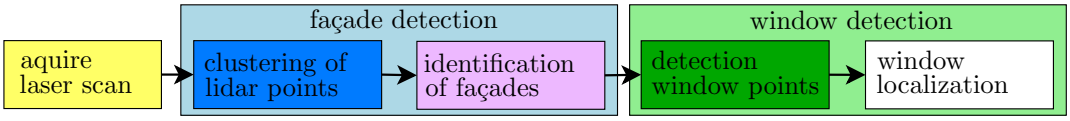


Figure 5.3: Flow chart of a window detection approach for façades, reproduced from Wang et al. [2010].

Following, the bottom-up method which is based on Principal Component Analysis (PCA), identifies potential façade regions (highlighted in purple). This is based on the assumption that façades have two major directions (vertical and horizontal). A set of neighbouring points $\{p_i\}_{i=1:N}$ is used to create a 3×3 positive semi-definite matrix

$$W = \frac{1}{N} \sum_{i=1}^{N} (p_i - \bar{p}) \otimes (p_i - \bar{p}) \tag{5.1}$$

with the centroid of all points $\bar{p}$

$$\bar{p} = \frac{1}{N} \sum_{i=1}^{N} p_i.$$

For $\lambda_1 \leq \lambda_2 \leq \lambda_3$ the corresponding Eigen values of W are assigned as Eigen vectors

$$\begin{aligned} \vec{v_1} &\leftarrow \lambda_1 \\ \vec{v_2} &\leftarrow \lambda_2 \\ \vec{v_3} &\leftarrow \lambda_3 \end{aligned}$$

Since the PCA does not have a defined normal direction (can be towards or against the light source), following the normals are calculated and reorientated so that a positive normal is directing towards the sensor. This allows an easy determination of surface orientation. After, all façade points are transformed into the robot system. To do so, it is assumed that façade normals are pointing perpendicularly to the robot (parallel to the ground).

Next, subsequently the bottom-down method which is based on a Random-Sample-Consensus-plane-fitting-algorithm (RANSAC-plane-fitting-algorithm) is used on the point cloud to extract invalid façades with their corresponding points. The potential window points are determined (highlighted in dark green). To overcome differences in structure between the ground level and the upper levels, Wang et al. [2010] separated the ground level of the façade. This is done by cutting the first $10 - 30\%$ from the determined façade. Then, four different types of window borders are defined: left window border (vertical), right window border (vertical), upper window border (horizontal), and lower window border (horizontal). Non-transparent window patterns, e.g. crossbars, are not taken into account which results in an error. This error can be neglected in most cases. After, a volumetric representation of the façade is created. That is why the borders are identified by the relation of points with their neighbours (highlighted in white). It is assumed, e.g. that the upper window border points do not have any neighbours below (based on a distance threshold), that left border points do not have any neighbours on the right side, etc.. This results in three groups of points

$$f(\vec{p}) = \begin{cases} \vec{p}_{horizWEdge}, & \text{if} \quad \{\{\sum_{k'=k}^{k'=k+inter} \sum_i \sum_j f(i,j,k')\} = 0 \\ & \qquad \wedge \quad \{\sum_{k'=k}^{k'=k-d} \sum_i \sum_j f(i,j,k')\} = d\} \\ & \vee \quad \{\{\sum_{k'=k}^{k'=k-inter} \sum_i \sum_j f(i,j,k')\} = 0 \\ & \qquad \wedge \quad \{\sum_{k'=k}^{k'=k+d} \sum_i \sum_j f(i,j,k')\} = d\} \\ \vec{p}_{vertWEdge}, & \text{if} \quad \{\{\sum_{i'=i}^{i'=i+inter} \sum_j \sum_k f(i',j,k)\} = 0 \\ & \qquad \wedge \quad \{\sum_{i'=i}^{i'=i-d} \sum_j \sum_k f(i',j,k)\} = d\} \\ & \vee \quad \{\{\sum_{i'=i}^{i'=i-inter} \sum_j \sum_k f(i',j,k)\} = 0 \\ & \qquad \wedge \quad \{\sum_{i'=i}^{i'=i+d} \sum_j \sum_k f(i',j,k)\} = d\} \\ \vec{p}_{nonWEdge}, & \text{otherwise} \end{cases} \tag{5.2}$$

Here $\vec{p}_{horizWEdge}$ contains the horizontal window border points, $\vec{p}_{vertWEdge}$ contains the vertical window border points, $\vec{p}_{nonWEdge}$ contains all other points, *inter* describes an interval value between the windows, and d is the width of the crossbars.

Next, a plane-sweep principle creates two projection profiles. Horizontal window border points $\vec{p}_{horizWEdge}$ are projected parallel to the ground and vertical window border points $\vec{p}_{vertWEdge}$ are projected perpendicular to the ground. In these profiles, peaks are assumed to be the window borders. Now the resulting 3D coordinates of each window can be determined.

The algorithm of Wang et al. [2010] is limited to square windows in façades to reconstruct them. Transparent objects inside a building cannot be identified. Besides, specular reflective effects (mirrors, shiny metal) are not examined. Furthermore, effects caused by transparent or specular reflective objects are not handled. This is why the approach does not apply to a disaster environment.

5.2.4 Laser Scanner Fused with Ultrasonic Sensor for 2D Mapping

Yang and Wang [2008] uses an online running sensor fusion of a laser scanner and an ultrasonic sensor to detect potential transparent and specular reflective objects. To fuse the data of both sensors, for each sensor an occupancy grid based map is created ($M^L_{x,y}$ for the laser and $M^S_{x,y}$ for the ultrasonic sensor). These maps are fused by using probabilities to describe their confidence in measurements:

$$M^L_{x,y} < \kappa^L \qquad \text{with } \kappa^L = 0.05 \tag{5.3}$$

and

$$M^S_{x,y} > \kappa^S \qquad \text{with } \kappa^S = 0.95 \tag{5.4}$$

At each time step $M_{x,y}$ results in

$$f(M_{x,y}) = \begin{cases} M_{x,y} = M^S_{x,y} & \text{if} \quad (M^L_{x,y} < \kappa^L) \wedge (M^S_{x,y} > \kappa^S) \\ M_{x,y} = M^L_{x,y} & \text{else} \end{cases} \tag{5.5}$$

By assuming that walls are smooth, gaps $G_{i,j}$ in the laser scan are defined as possible mirror locations. These discontinuities in the walls determine the end points $\vec{p}_i$ and $\vec{p}_j$ of a potential mirror. To determine the gaps, two measurements $\{z_i, z_j | 1 \leq i < j \leq n, j - i > 1\}$ are compared, such that

$$z_{i+1} - z_i > \epsilon_{gab} \tag{5.6}$$

$$z_{j-1} - z_j > \epsilon_{gab} \tag{5.7}$$

$$|z_k - z_{k+1}| \leq \epsilon_{gab} \qquad \text{for} \qquad i < k < j - 1 \tag{5.8}$$

with z_i is the i^{th} measurement in the laser scan, z_j is the j^{th} measurement in the laser scan, n is the cardinality $|z|$ of the observation z, ϵ_{gab} is the threshold to define a gap.

Since this is a 2D approach, flat mirror can be modeled as a line $e_{i,j}$ with two end points $\vec{p}_i$ and $\vec{p}_j$. These line end points are the corresponding Cartesian coordinates of z_i and z_j. They are stored as well as the points between $\{\vec{p}_{i+1}, \vec{p}_{i+2}, ..., \vec{p}_{j-1}\}$ and the open gap is defined as a potential reflective object $M_{i,j}$. Additionally the line $e_{O,k}$ is defined between the origin $\mathbf{O}$ and the point $\vec{p}_k$. The intersection point between the two lines is defined as $\vec{p}_{i,j,k}$. Following, the Euclidean distance function

$$\rho(\mathbf{O}, p_k) = \rho(\mathbf{O}, p_{i,j,k}) + \rho(p_{i,j,k}, \tilde{p}_k) \tag{5.9}$$

as well as the angle function

$$\angle(\mathbf{O}, p_{i,j,k}, p_i) = \angle(p_j, p_{i,j,k}, \tilde{p}_k) \tag{5.10}$$

are calculated. The likelihood $P_{l\_i,j}$ for the reflected scan points $\{\tilde{p}_{i+1}, \tilde{p}_{i+2}, \ldots, \tilde{p}_{j-1}\}$ is determined too. These points are the back-projected points $\{\vec{p}_{i+1}, \vec{p}_{i+2}, \ldots, \vec{p}_{j-1}\}$ w.r.t. the mirror line $e_{i,j}$. For a likelihood $P_{l\_i,j}$ greater than a threshold ϵ_l the gap is considered as a mirror otherwise as a regular gap. For a positive identified mirror, the end points as well as the line model are stored separately and the mean line segment state vector is calculated by

$$\mu^R_{M_{i,j}} = \begin{pmatrix} \alpha^R_{M_{i,j}} \\ \lambda^R_{M_{i,j}} \end{pmatrix} = \begin{pmatrix} \arctan\left(\frac{\bar{y}_{i,j,k}}{\bar{x}_{i,j,k}}\right) \\ \sqrt{\bar{x}^2_{i,j,k} + \bar{y}^2_{i,j,k}} \end{pmatrix} \tag{5.11}$$

where $\bar{x}^2_{i,j,k}$ and $\bar{y}^2_{i,j,k}$ are the closest points on the line to the origin. The uncertainty of the mirror prediction is based on the covariance matrix of an applied ICP matching. The covariance is defined by

$$\sum_{M_{i,j}}^{R} = \begin{pmatrix} \sigma^2_\alpha + \Delta\theta^2 & 0 \\ 0 & \sigma^2_\rho + \Delta x^2 + \Delta y^2 \end{pmatrix} \tag{5.12}$$

where σ_α (in °) and σ_ρ (in m) are pre-determined values of the measurement noise and $\Delta x, \Delta y,$ and $\Delta\theta$ is the registration result of the ICP.

To update and track the mirrors an EKF integrates the mirror predictions at every time step. Further, the line parameters as well as the end points are updated separately.

To update the line parameters, they are transformed into the global coordinate system by

$$\mu_{M_{i,j}} = \begin{pmatrix} \alpha^R_{M_{i,j}} + \theta_t \\ x_t \cos(\alpha^R_{M_{i,j}} + \theta_t) + y_t \sin(\alpha^R_{M_{i,j}} + \theta_t) \end{pmatrix} \tag{5.13}$$

with

$$\sum_{M_{i,j}} = J_{x_t} P_t J^T_{x_t} + J_{M_{i,j}} \sum_{M_{i,j}}^{R} J^T_{M_{i,j}} \tag{5.14}$$

where P_R is the covariance matrix of the robot pose, J_{x_t} and $J_{M_{i,j}}$ the Jacobian matrices of the line model w.r.t. the robot pose $\vec{P}_t = (x_t, y_t, \theta_t)^T$ and the line measurements. Following, the standard EKF process is applied to update the mean $\mu_{M_{i,j}}$ as well as the covariance of the estimated mirror $\sum_{M_{i,j}}$.

Next, the end points are updated and the line end points as well as the maximum extension of the line is calculated. This is done to overcome the drawback that transparent or specular reflective objects are occasionally visible. To do so, the points at the time step t and the time step $t+1$ are fused to

$$\{\hat{p}^{t+1}_1, \hat{p}^{t+1}_2\} = \text{argmax}_{p_1, p_2 \in P} \rho(p_1, p_2), \tag{5.15}$$

where $\hat{p}^{t+1}_1$ and $\hat{p}^{t+1}_2$ are the resulting end points of $\hat{M}^{t+1}$.

While mirrors are only tracked online during the mission, the resulting mirror locations are used in an offline procedure to reduce reflective effects. Hence, the mirror location is the basis to erase points behind. This approach will fail as soon as the mirror has no borders or has a non-planar shape. It also fails if there is no valid back-projection which results in a failure at the ICP. The back-projected points are not taken into account to update the map. The approach is specialized to identify specular reflective objects but not transparent objects. A distinction between transparent and specular reflective objects is not made either.

5.2.5 Selective Fusion of Laser Scanner with Ultrasonic Sensor for 2D Mapping

Lai et al. [2005] selectively "fuse" a laser scanner and an ultrasonic sensor to reduce transparent and specular reflective objects on the fly. In compare to Yang and Wang [2008] this approach does not build two individual grid maps and match them. Lai et al. [2005] assume that the laser scanner delivers precise measurements and therefore rely on it as a main sensor for data delivery. Only in case of specular transparent or specular reflective objects the laser scanner delivers incorrect measurements. In this case, the ultrasonic sensor data are taken into account. To combine the data, only the scan angle where both sensors overlap in front of the robot is considered. Here, the measurements are chosen based on the distance from the robot.

$$P(m_l^f|z_t^l, z_t^s, x_t) = \begin{cases} P(m_l^s|z_t^s, z_t^s, x_t), & \text{if} \quad z_t^s < \min_{i\in\Theta} z_t^l(i) \\ P(m_l^l|z_t^l, z_t^s, x_t), & \text{otherwise} \end{cases} \tag{5.16}$$

The approach does only map the surface of the transparent or specular reflective object, but does not consider objects behind or mirrored-objects. Further, it does not distinguish between the object types.

5.2.6 Mirror Detection Based on Symmetry for 2D Mapping

In further research Yang and Wang [2011] extended their algorithm ([Yang and Wang, 2008]) to merge a laser scanner with an ultrasonic sensor. In addition, their algorithm uses the advantage of the mirror symmetry property to identify reflective influences. Applying a Bayesian framework, their approach identifies spatial and temporal symmetry. While spatial symmetry results from a symmetry in the environment, temporal symmetry results from specular reflective objects. They assume that mirrors are framed, planar, and located on walls. They also assume that environments are smooth.

Hence, an open gap $G_{i,j}$ in a wall is located by its discontinuities in the environment and determined as a potential mirror. The two outer discontinuities z_i and z_j, with $\{z_i, z_j|1 \leq i < j \leq n, (j-i) > 1\}$, are identified by the difference between the neighbouring measurement, such that

$$z_{i+1} - z_i > \epsilon_{gab} \tag{5.17}$$

$$z_{j-1} - z_j > \epsilon_{gab}. \tag{5.18}$$

$$|z_k - z_{k+1}| \leq \epsilon_{gab} \qquad \text{for} \qquad i < k < j-1. \tag{5.19}$$

Here, z_i is the i^{th} measurement, z_j is the j^{th} measurement, n is the cardinality $|z|$ of the observation z, ϵ_{gab} is the threshold to define a gap. The line end points $\vec{p}_i$ and $\vec{p}_j$ are stored as well as the points between $\{z_{i+1}, z_{i+2}, ..., z_{j-1}\}$. This defines each open gap as a potential reflective object $M_{i,j}$. Then the local map is searched for symmetric objects w.r.t. the line by an ICP algorithm. In case of a positive result the gap is considered as a specular reflective object (mirror) by a likelihood l with $\{l \in \mathbb{R}_0^+ | \, 0 \leq l \leq 1\}$.

$$f(M_{x,y}) = \begin{cases} M_{x,y} = M_{x,y}^S & \text{if} \quad M_{x,y}^L < \kappa^L \wedge M_{x,y}^S < \kappa^S \\ M_{x,y} = M_{x,y}^L \end{cases} \tag{5.20}$$

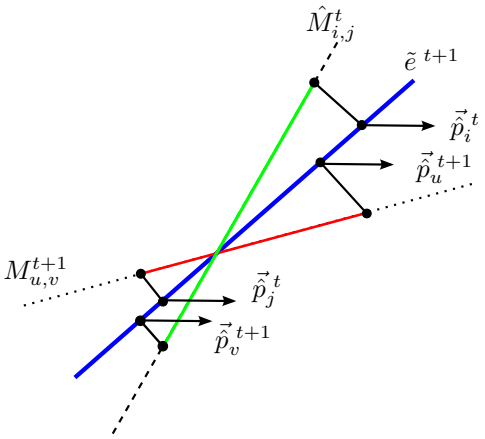


Figure 5.4: Update endpoints, reproduced from Yang and Wang [2011].

Therefore, a mean vector is defined by

$$\mu_{M_{i,j}} = \begin{pmatrix} \alpha^R_{M_{i,j}} + \theta_t \\ x_t \cos(\alpha^R_{M_{i,j}} + \theta_t) + y_t \sin(\alpha^R_{M_{i,j}} + \theta_t) \end{pmatrix} \tag{5.21}$$

as well as a covariance

$$\sum M_{i,j} = J_{x_t} P_t J^T_{x_t} + J_{M_{i,j}} \sum_{M_{i,j}}^{R} J^T_{M_{i,j}} \tag{5.22}$$

by the Jacobian matrices of the line Model J_{x_t} and $J_{M_{i,j}}$ w.r.t. the robot pose $\vec{P}_R = (x_t, y_t, \Theta_t)^T$ and the line measurements.

Now, the line and its end points are transformed into global coordinates, stored, tracked (see Algorithm 7), and updated (cf. Figure 5.4). For the updated line the outer lying points are considered.

Yang and Wang [2011] do not distinguish between transparent and specular reflective objects. Reflected object points are not back-projected to improve mapping. Since mirror identification relies on gap identification and symmetry, it will result in wrong mirrors for open gaps in a symmetric environment. Also, transparent objects will not be identified correctly as they do not have any back-projection.

Algorithm 7 Mirror tracking [Yang and Wang, 2011]

Input: $\hat{M}^t, M^{t+1}$: $\hat{M}^t$ is the updated mirror estimated at time t and M^{t+1} is the associated mirror at time $t+1$.

Output: $\hat{M}^{t+1}$: $\hat{M}^{t+1}$ is the updated mirror at time $t+1$

$\hat{M}^{t+1} \leftarrow \emptyset$ ▷ initialisation
for each $M_{i,j} \in \hat{M}^t$ **do**
  Find $M_{u,v}^{t+1} \in M^{t+1}$ associated with $M_{i,j}$
  if $M_{i,j}^{t+1}$ exists **then**
   Line update stage
   Perform Bayesian filtering
    Calculate $\hat{\mu}_{M_{i,j}}^{t+1}$ ▷ calculate mean
    Calculate $\hat{\sum}_{M_{i,j}}^{t+1}$ ▷ calculate covariance
    Calculate $\hat{e}^{t+1}$ ▷ build line model
   Find end point candidates
    Calculate $P = \{\vec{p}_i^{\,t}, \vec{p}_j^{\,t}, \vec{p}_u^{\,t+1}, \vec{p}_v^{\,t+1}\}$
    Calculate $\hat{P} = \{\vec{\hat{p}}_i^{\,t}, \vec{\hat{p}}_j^{\,t}, \vec{\hat{p}}_u^{\,t+1}, \vec{\hat{p}}_v^{\,t+1}\}$
   Endpoints update stage
    Calculate $\{\vec{\hat{p}}_1^{\,t+1}, \vec{\hat{p}}_2^{\,t+1}\}$ ▷ endpoints, cf. Figure 5.4
    Add $\{\hat{\mu}_{M_{i,j}}^{t+1}, \hat{\sum}_{M_{i,j}}^{t+1}, \vec{\hat{p}}_1^{\,t+1}, \vec{\hat{p}}_2^{\,t+1}\}$ to $\hat{M}^{t+1}$
  else
    Add $M_{i,j}^{t+1}$ to $\hat{M}^{t+1}$ ▷ new mirror
  end if
end for

5.2.7 Laser-Based Glass Detection Based on a Density Function

The approach of Awais [2009] is an extension of Thrun et al. [2005]. It is detecting glass by combining three probability density functions: direct reflection, normal reflection, and refraction (cf. Figure 5.5)

For the direct reflection the likelihood is calculated by

$$P(direct|angle) = e^{-(\Theta_i)^2 \cdot (2\sigma_{d_r}^2)^{-1}} \tag{5.23}$$

with $\sigma_{d_r}^2$ being the standard deviation (normally $\sim 3°$) of the measurement d_r, Θ_i the incident angle, and i the number of scan points. This is combined with the probability of the sensor observation z and its deviation σ_z. Therefore,

$$\begin{aligned} P(z, direct|angle) &= P(direct|angle) \cdot P(z|direct, angle) \\ P(z, direct|angle) &= P(direct|angle) \cdot \frac{1}{\sqrt{2\pi} \cdot \sigma_z} \cdot e^{-((z-d_g)^2) \cdot (2\sigma_z^2)^{-1}} \end{aligned} \tag{5.24}$$

therefore d_g is the distance of the glass from the laser and σ_z is the standard deviation of observation z which is used according to Thrun et al. [2005].

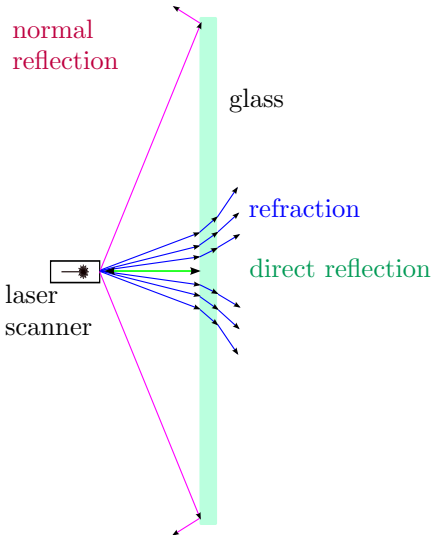


Figure 5.5: Reflections depending on their incident angle as they occur at a glass surface, reproduced from Awais [2009].

Following, the normal reflection likelihood is calculated by

$$P(norm|angle) = e^{-\lambda_{tune}(90^\circ - \Theta_i)} \tag{5.25}$$

while λ_{tune} is the tuning parameter and Θ_i the incident angle. Also, this is combined with the probability of sensor observation z and its deviation σ_z. So,

$$\begin{aligned} P(z, norm|angle) &= P(norm|angle) \cdot P(z|norm, angle) \\ P(z, norm|angle) &= P(norm|angle) \cdot \frac{1}{\sqrt{2\pi} \cdot \sigma_z} \cdot e^{-(z-D)^2 \cdot (2\sigma_z^2)^{-1}} \end{aligned} \tag{5.26}$$

with $D = d_o + d_g$, while d_o represents the distance from glass to a nearby object and d_g is the distance from the glass to the laser scanner.

Next, the refraction likelihood is calculated by

$$P(refrac|angle) = 1 - P(norm|angle) - P(direct|angle). \tag{5.27}$$

This assumes that the likelihood of the direct and normal reflection is low at all other angles and therefore the likelihood of the refraction will be high. Combined with the probability of the sensor observation z and its deviation σ_z it follows

$$\begin{aligned} P(z, refrac|angle) &= P(refrac|angle) \cdot P(z|refrac, angle) \\ P(z, refrac|angle) &= P(refrac|angle) \cdot \frac{1}{\sqrt{2\pi} \cdot \sigma_z} \cdot e^{-(z-\Phi)^2 \cdot (2\sigma_z^2)^{-1}} \end{aligned} \tag{5.28}$$

with $\Phi = (d_{real} + offset)$, where d_{real} is the distance to the closest obstacle.

Finally, all three probabilities are combined by the De Morgan‘s law to

$$P(z, x, m) = P(z, direc|angle) \vee P(z, norm|angle) \vee P(z, refrac|angle). \tag{5.29}$$

To integrate the glass detection into a particle filter based mapping algorithm, three versions are proposed: core heuristic, direct reflection model, and direct reflection model combined with Markov Random Field (MRF) [Li, 2009]. Best results are shown with the last version.

For each cell of the grid map the probabilities of its status (empty = EMP, occupied = OCC, or glass = GLS) are defined. The direct reflection model defines

- for an occupied cell

$$P^t_{OCC} = \frac{1}{1 + e^{(d_{real} - d_{range} - 5)}} \tag{5.30}$$

with

$$P_{OCC}(x, y) = \prod_{t=0}^{T} P^t_{OCC}(x, y), \tag{5.31}$$

- for a cell containing glass

$$P^t_{GLS} = 1 - P^t_{OCC} \tag{5.32}$$

with

$$P_{GLS}(x, y) = \prod_{t=0}^{T} P^t_{GLS}(x, y), \tag{5.33}$$

- and for an empty cell

$$P_{EMP}(x, y) = 1 - (P_{GLS}(x, y) + P_{OCC}(x, y)). \tag{5.34}$$

Following, the MRF is applied to minimize false positives and obtain a smooth result of the direct reflection model. Thus, the 2^{nd} order neighbourhood is used for the regularity function

$$f_{REG}(x, y, l) = \sum_{x'=\pm x} \Psi(l, f(x', y)) + \sum_{y'=\pm y} \Psi(l, f(x, y')) + \sum_{(x',y')=\pm(x,y)} \Psi(l, f(x', y')) \tag{5.35}$$

with $f(x, y)$ giving the label of site and the function Ψ is used to produce the penalty depending on the status (EMP, OCC, GLS) of the neighbouring cells. It is defined as

$$\Psi(l_1, l_2) = \begin{cases} 3, & \text{if} \quad l_1 = l_2 = GLS \wedge \Theta_{l_1} \neq \Theta_{l_2} \\ 0.6, & \text{if} \quad l_1 = GLS \wedge l_2 = EMP \\ 0, & otherwise \end{cases} . \tag{5.36}$$

The algorithm of Awais [2009] concerns only transparent objects influences. The approach is not limited to a predefined size of the obstacle, but falsely results in object detection in case of specular reflections. It also does not consider objects which are located behind the transparent surface.

5.2.8 Visible Angle Grid for Glass Environments (VisAGGE)

Also Foster et al. [2013] take advantage of the angle dependency of reflections to identify and eliminate transparent and specular reflective object influences. They modified a standard occupancy grid algorithm to map only objects which are visible in a certain angle range. Next, the laser scan is divided into two groups of points: “critical” and “certain”. The first group of measurements is located in an angle range in which distortions can be caused by transparent

or specular reflective objects. The second group of measurements is located in an angle range in which transparent or specular reflective objects do not have an influence.

To include the different types of points, the occupancy grid is split into three layers. The first layer immediately includes "certain" measurements. The second layer contains "critical" measurements. The algorithm checks if these measurements had been seen before from a "certain" angle. A third layer stores recent angles. This makes it possible to review them, relative to each cell, and allow motion detection based on a heuristic when traversing the environment the first time. For each cell, the probability is modeled by a visibility function $p(Z_n)$ with $VIS(\Phi) \approx 1$ for diffuse objects and $VIS(\Phi) \approx 0$ for glass. It is calculate by

$$p(Z_n) = VIS_n(\Phi) \cdot \prod_{i=0}^{n-1} (1 - VIS_i(\Phi)) \tag{5.37}$$

with the event Z_n from surface n.

Since the algorithm detects transparent and specular reflective objects based exclusively on the incident angle of the laser beam, it has no limitations concerning size, shape, or object type. To identify the object completely it is essential to bypass it. This assures that the object was seen from the "right perspective" (angle range) at least once. Unfortunately, the algorithm does not distinguish between transparent or specular reflective objects. That is why it is not possible to back-project mirrored measurements.

5.2.9 Glass Detection Based on the Incident Angle

The approach presented by Wang and J. [2017] is similar to the previously described approach of Foster et al. [2013]. In comparison, they do not calculate probabilities of glass objects but use simply three thresholds ($\epsilon_{glassIntensityDelta}, \epsilon_{glassTriggerInensity}, \epsilon_{glassProfilWidth}$) to determine a transparent obstacle. These thresholds are determined by a "calibration" procedure where the laser intensity is measured over the incident angle ($\pm 2.5°$), cf. Figure 5.6.

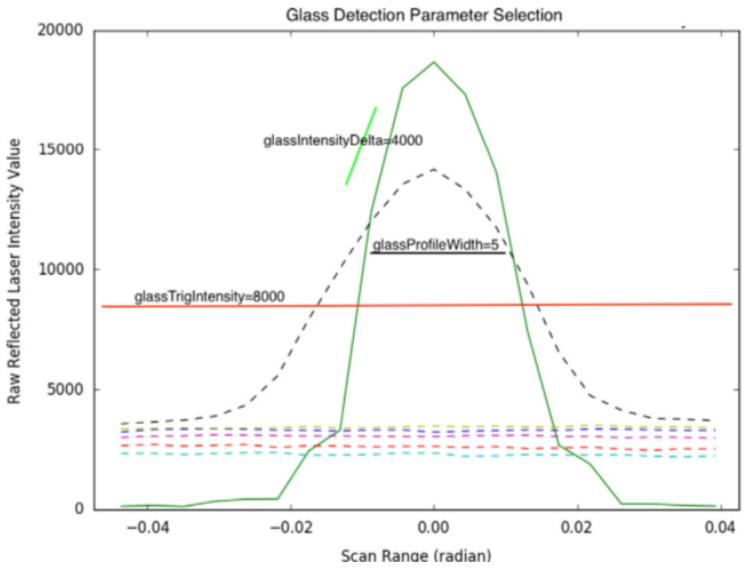


Figure 5.6: Laser "calibration" to determine threshold values [Wang and J., 2017].

The authors assume that the intensity dependency on the distance is negligible and apply the thresholds regardless of the measured distance. This assumption is wrong as shown by Tatoglu and Pochiraju [2012], Thrun et al. [2005] and in Section 7.1 of this work. As a result, this approach is only applicable for obstacles close to the distance used during calibration.

The determined thresholds are used to identify glass panels (highlighted in red). These panels are stored into a glass list $L_{glassTmp}$ with the distance to the robot d at the intensity peak, the corresponding scan angle α, the robot pose $\vec{P}_R = \{x, y, \theta\}$, and the time stamp t. Next, the trajectory of the robot is calculated based on a Rao-Blackwellized Particle Filter [Grisetti et al., 2005] (highlighted in green) and the map updated (highlighted in yellow). Afterwards, the raw glass panel list $L_{glassTmp}$ is updated with the best particle trajectory and stored in a final glass list L_{glass} (highlighted in blue), as illustrated in Figure 5.7

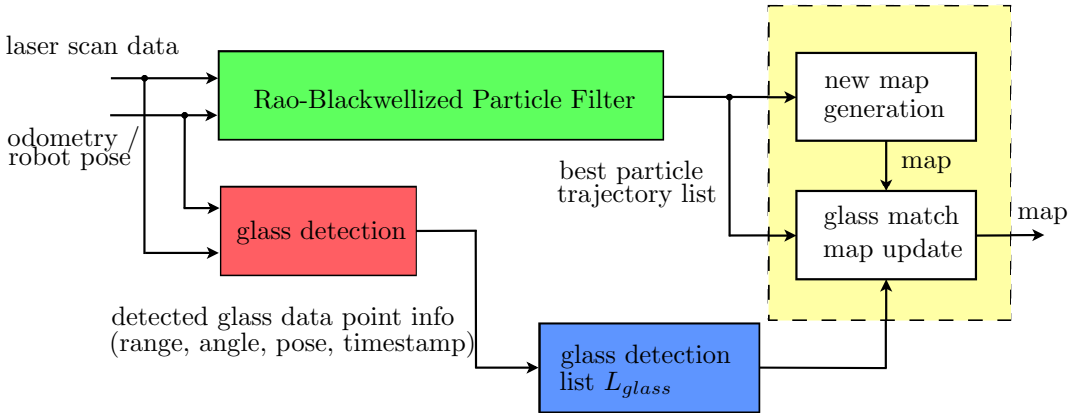


Figure 5.7: Modified SLAM algorithm with glass detection procedure, reproduced from [Wang and J., 2017]

The algorithm supplies an online identification for transparent objects. Due to its simplicity it is fast and requires less computing power, but it suffers from several drawbacks. First, the assumption that the intensity remains almost constant over the distance is wrong. Therefore, the method works correctly only at the distance which was used for calibration. Besides, objects located further away will result in less measurements on the same angle range as illustrated in Figure 4.3. The laser angular resolution results in less angular steps and therefore $\epsilon_{glassProfilWidth}$ is not gained. Similarily, incorrect intensity measurements will result in a wrong object classification because the gradient of intensity ∇int is simply calculated by Δint of two following intensity values $\Delta int = int_i - int_{i-1}$. This is why outliers have an huge impact. To overcome these problems, an adaptation of the threshold values w.r.t. the measured distance should be proceeded.

Another drawback is that there is no distinction between transparent and specular reflective objects. Specular reflective objects result in a more significant intensity curve than transparent objects. It can be assumed that the approach will cover them too. Nevertheless, the approach does not cover erroneous measurements (mirrored points) or distinguish between them. As a last drawback, only stationary transparent objects are taken into account. Hence, this approach is not applicable in a real world scenario as objects often are in motion.

5.2.10 Glass Detection by Respecting Different Scan Locations

Also Park et al. [2013] take advantage of the fact that transparent obstacles are only occasionally visible. That is why their algorithm takes scans of different locations into account to identify transparent obstacles. To do so, it is assumed that the first measurement always results from the transparent obstacle, while additional measurements result from obstacles behind or from reflected noise, cf. Figure 5.8 (marked by a black dashed line). While measurements from wall segments remain stationary, measurements caused by reflected noise move based on the location of the robot. This requires a stationary environment during the experiment. Hence, only a static map can be created. Besides, the authors assume that all transparent obstacles have a polygonal structure and a planar surface. Furthermore, obstacles should not be located directly behind the transparent surface. If so, the approach will lead to wrong measurements because the distance is smaller than $\epsilon_{obstacle}$ which is used to identify differences in the measured distances.

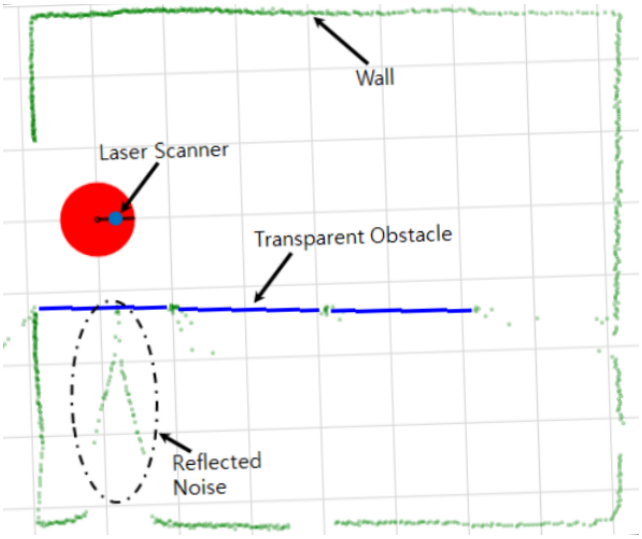


Figure 5.8: Laser scan data with noise (marked by a black dashed line) which result from a transparent object (blue line) [Park et al., 2013].

To identify transparent obstacles the algorithm, illustrated in Algorithm 8, uses the points between the robot ($cell_R(t)$) and the measured points ($cell_n(t)$) which are stored into a vector $\vec{L}$. These points in $\vec{L}$ result from the Map $DCMap(cell)$ after a movement of the robot was detected. The closest occupied cell to the robot is taken out of $\vec{L}$. This cell is assumed to result from the transparent object. If a threshold $\epsilon_{obstacle}$ is reached, the value of the cell $DCMap(cell_{L1})$ is increased.

This approach addresses solely planar, transparent obstacles. It fails if obstacles are located close behind the transparent surface. Besides, the approach does not respect dynamic processes (moving objects) which is necessary for mapping in real environments.

Algorithm 8 Mapping for transparent objects Park et al. [2013]

Input: $d_n(t), \alpha_n, \vec{P}_R(t), \theta_R(t)$: $d_n(t)$ is the distance of the n^{th} scan point at the time t, α_n the corresponding scan angle, $\vec{P}_R(t) = \{x, y\}$ is the robot position with x and y coordinates a the time t, and $\theta_R(t)$ is the orientation of the robot at the time t.

Output: $DCMap(cell)$: $DCMap(cell)$ is a grid map with counting number for each cell.

procedure MAPPING
  if $\vec{P}_R(t) \neq \vec{P}_R(t-1)$ **then** ▷ check, if robot has moved
    for each $d_n(t)$ **do**
      if $d_n(t) \in [D_{min}, D_{max}]$ **then** ▷ erase points out of measurement distance
        calculate $cell_n(t) := f(d_n(t), \alpha_n(t), \vec{P}_R(t), \theta_R(t))$
        create vector $\vec{L}$ of points between $cell_R(t)$ and $cell_n(t)$
        ▷ excluding $cell_R(t)$ and $cell_n(t)$
        loop
          find minimal distance from occupied cell $cell_{L1}$ to the robot $cell_R(t)$ in $\vec{L}$
          if $DCMap_{max} > DCMap(cell_{L1}) \geq \epsilon_{obstacle}$ **then**
            $DCMap(cell_{L1})$ += 1
            EXIT LOOP
          end if
          find next nearest cell $cell_{L2}$
          $cell_{L1} := cell_{L2}$
        end loop
        if $DCMap$ values for all cells in $\vec{L} < \epsilon_{obstacle}$ **then**
          $DCMap(cell_n(t))$ += 1
        end if
      end if
    end for
  end if
end procedure

5.2.11 3D Mirror Detection by Jumping Edge Detection in Panorama Images

Most approaches cover the 2D case. This is not sufficient when operating in a rescue scenario as it is exemplarily described in Chapter 1. Käshammer and Nüchter [2015] presented an online running 3D approach which transforms a 3D point cloud into a panorama image representation. Following, this panorama representation is searched for framed, square mirrors with a predefined size. Further, the points which are identified to be located behind the mirror plane are back-projected w.r.t. the mirror plane.

To result in a panorama image each point $\vec{p}_{3d,i} = \{\Theta, \Phi, r\}$ (in spherical coordinates) is converted into $\vec{p}_{Pano,i} = \{u, v\}$. Therefore, following projection rule is proceeded on each of the i scan points:

$$u = \Theta$$

$$v = \Phi$$

Applying the *farthest*-method, points $\vec{p}_{3d,i}$ projected on the same panorama point $\vec{p}_{Pano,i}$ are selected by their greater value of r. The distance r is represented by the gray value of the point $\vec{p}_{Pano,i}$. Undefined or missing panorama points are filled up in black ($r = 0$). After, the algorithm searches for jumping edges in the gray values of the panorama image by comparing to

a threshold ϵ_{jump}. The OpenCV-function *findContures()* [OpenCV, 2017] identifies the border of potential mirrors $M_{pot,j}$ (j is the amount of potential matches).

Several filter algorithms which erase wrong candidates of $M_{pot,j}$, are described below. The first algorithm checks for the mirror frames. Therefore, the contour of potential mirror $M_{pot,j}$ needs to be connected fully. Second, a range filter identifies if the contour points have less than a maximal variation in depth. Then, a PCA is applied to calculate the plane parameters. The plane is defined by its normal vector $\vec{n}_0$ and the distance r to the origin. After, the dimensions $x_{contour}$, $y_{contour}$, and $z_{contour}$ of the potential mirror border is determined. If each dimension conforms to the predefined sizes of the mirror, within an specified range, the positive identified mirror is stored in a list M_g where the amount of identified mirrors g is represented.

At the end, the algorithm checks point $\vec{p}_{3d,i}$ if it is located inside the mirror border. Positive matches are compared to the distance r of the mirror plane. Points located behind the mirror plane are assumed to be reflections in the mirror. Therefore, these points are back-projected w.r.t. the mirror plane.

Limitations of this algorithm occur for mirrors which are located far away from the scanner or beeing distorted. Furthermore, the algorithm respects solely square mirrors with a predefined size in current scans. This allows a pre-filtering of specular reflective influences, but fails for mirrors which are not seen completely in the current scan. Besides, it results in mismatches for transparent objects which are framed and have the predefined size.

5.2.12 Transparent Object Reconstruction in 3D

Also Albrecht [2017] presents a 3D approach to detect transparent and specular reflective objects as well as modulate their surface. Further, his work presents a method to automatically evaluate ICP-matching results which is not considered further. The approach applies point clouds from a laser scanner or a RGB-D-camera. Such point clouds are organized or unorganized based on the applied sensor. Organized point clouds are bigger and exist of a consistent size of measurements. The arrangement implies neighbouring points and erroneous measurements which are marked by a Not-a-Number-value (NaN-value). For unorganized point clouds these NaN-points are erased. This makes the identification of potential transparent and specular reflective objects more difficult. Based on the assumption that transparent or specular reflective objects are located on even ground, e.g. on a table, planes are identified by a RANSAC-based algorithm. Later, they are used to verify objects.

At unorganized point clouds, points which do not belong to a plane are clustered. They belong to objects. Regular objects stand on the plane and therefore the points are close to it. Clusters above or underneath the plane are assumed to be from a transparent object. Further, holes in the plane are identified. It is assumed that these holes are occlusions from an object. When taking the object clusters into account, the occlusion can be verified. To do so, a line is projected from the sensor to the center of the hole. If it does not pass any cluster this occlusion might be from a transparent object. If it does pass a cluster the result depends on the cluster. At organized point clouds occlusions of objects can be identified by searching for NaN-clusters. Therefore, a region-growing algorithm is applied.

By taking multiple point clouds from different view points into account, the surface of the transparent or specular reflective object is reconstructed. This is achieved by respecting the shape of the occlusion, the plane, and the measurements of the object surface, if they are available.

The main drawback of this algorithm is that the transparent and specular reflective surfaces need to be located on a surface, e.g. a table. Further, verification and object reconstruction requires multiple view points. Hence, it requires some time to identify transparent and specular

reflective objects. That is why the algorithm is not rated as online running in Table 5.1. The work of Albrecht [2017] is specialized to identify and reconstruct transparent and specular reflective objects in order to grab them. It is not applicable in a mapping scenario as it was described in Chapter 1.

5.3 Summary

Table 5.1 summarizes the state-of-the-art concerning recognition, identification, and separation of transparent and specular reflective influences. It points out important attributes, identified in Section 1, to operate a mobile robot in a rescue scenario. For each approach the satisfied criteria are highlighted in green.

As previously described, approaches designed for window detection in façades are not applicable due to the fact that transparent and specular reflective objects also occur elsewhere. Besides, it cannot be assured that façades will remain intact after an earthquake. Most of the other approaches which apply only a laser scanner, are capable to deal with unframed objects regardless or their size. Some of them are capable to detect transparent as well as specular reflective objects, but none of the approaches distinguishes between them. Hence, they do not back-project points located "behind" a specular reflective object (mirrored points) and leave points located behind a transparent object untouched at the same time. Only the approach of Käshammer and Nüchter [2015] back-projects points behind a mirror plane in 3D, but still has several drawbacks. First, it does not cover transparent objects. That is why such objects can cause false results. Framed objects with a predefined size are required. It is understood that this does not apply to a rescue scenario.

This is the reason why subjected work presents an approach to identify both, transparent and specular reflective objects, to distinguish between them, and to individually deal with points behind such objects. The following section describes an approach for 2D and 3D to detect transparent and specular reflective influences, distinguish between them and to eliminate errors caused by them. It covers pending issues being identified in this section. Another section discusses applied experiments to demonstrate the applicability and reliability of the approach.

Table 5.1: Overview state-of-the-art concerning this work.

approach of	**2D/3D**	**respect mirrors**	**respect glass**	**shape**	**size**	**framed**	**only laser**	**back-projection**	**classi-fication**	**online**	**comment**
Ali et al. [2008]	3D	indirect	yes	planar	variable	yes	no	no	no	yes	requires façade
Pu and Vosselman [2007]	3D	indirect	yes	planar	variable	yes	yes	no	no	yes	requires façade
Wang et al. [2010]	3D	no	yes	planar	variable	yes	yes	no	no	yes	requires façade
Yang and Wang [2008]	2D	yes	no	planar	variable	yes	no	no	no	partly	
Lai et al. [2005]	2D	yes	yes	planar	variable	no	no	no	no	yes	
Yang and Wang [2011]	2D	yes	no	planar	variable	yes	yes	no	no	yes	
Awais [2009]	2D	indirect	yes	planar	variable	no	yes	no	no	yes	
Foster et al. [2013]	2D	indirect	yes	variable	variable	no	yes	no	no	yes	
Wang and J. [2017]	2D	indirect	yes	variable	variable	no	yes	no	no	yes	
Park et al. [2013]	2D	no	yes	planar	variable	no	yes	no	no	yes	
Käshammer and Nüchter [2015]	3D	yes	no	planar	pre-defined	yes	yes	yes	no	yes	
Albrecht [2017]	3D	yes	yes	variable	variable	no	yes	no	yes	no	requires planar surface

Chapter 6

Reflection-Identification-Approach

The Reflection-Identification-Approach is a ROS-package containing multiple nodes for 2D and 3D detection, classification, as well as purging of transparent and specular reflective object influences. This chapter is split into two sections: the 2D case and the 3D case.

In each section first, an functional overview is given by a processing chain. It illustrates the basic structure, the modules of the approach (implemented as a ROS-nodes), and its relation with other modules. Following, the internal functionality and structure of each module are explained as well as the interconnections with other modules are outlined.

6.1 2D-Mirror-Identifier-Approach

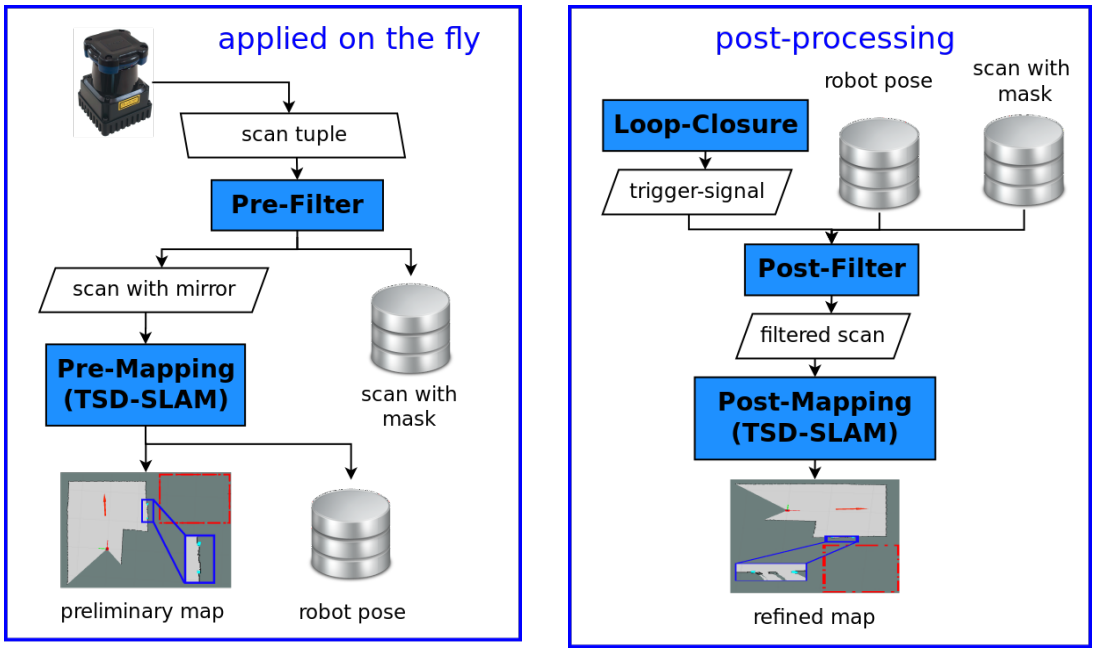
This section presents two versions of the 2D-Mirror-Identifier-Approach. Version 1, further called 2D-Mirror-Detector-Approach, applies two mapping modules and does not distinguish between transparent and specular reflective objects. It can be applied to any mapping approach without modifications. Version 2, further called 2D-Mirror-Identifier-Approach, requires only one mapping module and distinguishes between transparent and specular reflective objects. As it supports a post-update of the map, most mapping approaches need to be adapted. This procedure is described exemplarily in Section 6.1.5 for the TSD-SLAM (presented in Section 4.4). All modules are implemented as ROS-nodes and available at http://www.github.com/autonohm/ohm_mirror_detector.git.

6.1.1 Processing Chains of the 2D-Mirror-Identifier-Approach

The following processing chains give (cf. Figure 6.1 and Figure 6.2) an overview of the data flow between the different modules: Pre-Filter, Mapping, Loop-Closure, and Post-Filter. Each module, illustrated as a blue rectangle, is implemented as a ROS-node [ROS, 2015]. The data between each node are transmitted via ROS-messages and marked by white lozenges. Since all modules are implemented as ROS-nodes it is easy to exchange them with other nodes (e.g. TSD-SLAM with HECTOR-Mapping) as long as the required messages are supported.

Version 1: 2D-Mirror-Detector-Approach
The flow chart of the 2D-Mirror-Detector-Approach which is illustrated in Figure 6.1 consists of two chains. The first chain contains the 2D-Pre-Filter to erase transparent and specular reflective influences in current scans. It is applied on the fly, but cannot detect all influences due to angle dependencies. Therefore, the second chain is applying the 2D-Post-Filter to post-process the scans after a trigger signal.

The Pre-Filter receives a scan tuple S from the laser scanner. It contains distance and intensity data of Echo 1 and Echo 2. The 2D-Pre- and 2D-Post-Filter have an own mapping module attached. Both mapping modules receive a masked scan message from their filter modules and are building their own, individual map.



(a) Pre-Filter and Pre-Mapping.

(b) Post-Filter and Post-Mapping.

Figure 6.1: Processing chains of the 2D-Mirror-Detector-Approach: Pre-filtering removes affections on the fly. Post-Filtering refines the resulting map after a trigger signal was supported, e.g. from a Loop-Closure-module. The preliminary map still shows reflective influences (marked by a red dashed line rectangle) while the refined map is free of reflective influences. The location of the mirror is marked by a blue solid line rectangle and magnified.

The map compromises data from the Pre-Filter-module which excludes only effects of transparent and specular reflective objects which occur in current scans. It is built on the fly and is immediately available for navigation. The Post-Filter-module collects the scan messages from the Pre-Filter and builds up a history which also contains the related location and the pose of the robot. Location and pose are delivered by the Mapping-module. When it gets triggered by the Loop-Closure-module, the Post-Filter-module searches the entire history for transparent and specular reflective influences. As it processes the entire history, objects were seen from different perspectives. Therefore, it is possible to determine the location and size of the objects more precisely. The Loop-Closure-module checks if the robot returns to a previous location within a predefined amount of scans and distance. It compares the current location of the

robot with its path. It is assumed that a transparent or specular reflective object was bypassed in case of a closed loop. That is why the object was seen at least once. So, the laser beam hits the surface at least once from a visible angle. The TSD-SLAM module [Koch et al., 2015a] was the standard mapping module used for testing. Nevertheless, also other SLAM approaches were applied, for the experiments in Chapter 7, to demonstrate the usability and compatibility of the 2D-Mirror-Detector-Approach with other mapping modules.

Version 2: 2D-Mirror-Identifier-Approach
The illustrated 2D-Mirror-Identifier-Approach in Figure 6.2 is an extended version of the 2D-Mirror-Detector-Approach and is split into four modules: Pre-Filter, Mapping, Post-Filter, and Loop-Closure. In compare to the previous version, the Post-Filter module is capable to distinguish between transparent and specular reflective objects. Besides, there is no need for a second mapping module because the applied mapping approach (the TSD-SLAM) was customized. It should be mentioned that this version can be processed without two separate mapping nodes, equal to the 2D-Mirror-Detector-Approach (V1). The functional difference lies in the fact that the second version is capable to distinguish between transparent and specular reflective objects. If this feature is not required, Version 1 should be applied as it requires less computing power.

The Pre-Filter module of the 2D-Mirror-Approach remains unchanged. It receives a scan tuple S from the laser scanner which contains distance and intensity data of Echo 1 and Echo 2. The Pre-Filter runs on the fly and erases anomalies caused by transparent and specular reflective objects in the current scans. The preprocessed scan is then forwarded to the mapping module. The original scan, added with a mask for the identified anomalies from transparent and specular reflective objects, is forwarded to the Post-Filter.

The Post-Filter stores the masked data with their corresponding location and the robot pose in a history. Location and pose need to be supplied from the Mapping-module or any other Localization-module. As soon as an external trigger signal appears, e.g. from the Loop-Closure-module, the Post-Filter searches the entire history for transparent and specular reflective influences. In compare to the previous version (the Post-Filter of the 2D-Mirror-Detector-Approach), this version of the Post-Filter distinguishes between transparent and specular reflective objects by examining intensity values and considering back-projections.

In case of a positive identification, the Post-Filter sends out a filtered scan to update the mapping module. Regarding the identified type of object, the mapping module updates the map by erasing or adding scan points.

In case the applied mapping module does not support an update of the map it is possible to use two mapping modules as this was done in Version 1 - the 2D-Mirror-Detector-Approach.

The code of the individual modules of the 2D-Mirror-Detector and the 2D-Mirror-Identifier is described in detail below.

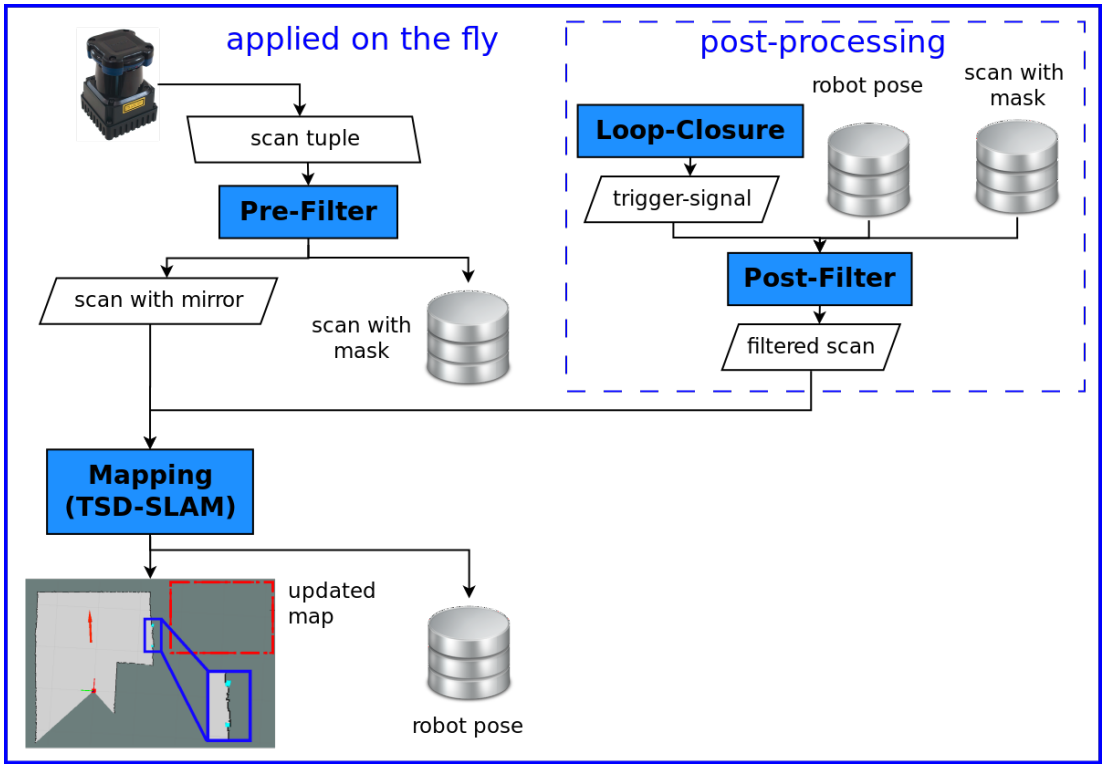


Figure 6.2: Processing chain of the 2D-Mirror-Identifier: Pre-filtering removes affections on the fly. Post-filtering refines the resulting map after a trigger signal, e.g. from a Loop-Closure-module. Because of that, the updated map is free of reflective influences (marked by a red dashed line rectangle). The detected mirror surface is magnified and marked by a blue solid line rectangle. At the magnification it can be seen, that the surface of the mirror was mapped completely. This can only be achieved with the modified SLAM module.

6.1.2 Code Description of the 2D-Pre-Filter

Figure 6.3 illustrates the simplified flow chart of the 2D-Pre-Filter in five steps: receive scan, clean-up scan, object recognition, filter scan, and publish scan. Algorithm 9 describes the Pre-Filter in detail and is highlighted according to the five steps of the flow chart.

Algorithm 9 2D-Pre-Filter

Input: S, α, I:
S includes the scan points of Echo 1 and Echo 2, α includes the corresponding angles, and I includes the corresponding intensities of Echo 1 and Echo 2.

Output: $G_{out,\text{valid,mirror,affected}}$:
$G_{out,\text{valid}}$ includes the valid scan points with their corresponding angles and intensities, $G_{out,\text{mirror}}$ includes the scan points, located on the surface of the transparent or specular reflective object, with their corresponding angles and intensities, and $G_{out,\text{affected}}$ includes the scan points, located behind the surface of the transparent or specular reflective object, with their corresponding angles and intensities.

1: **procedure** PREFILTER
2: $S, \alpha \leftarrow$ receiveScanTuple()
3: $D \leftarrow$ removeSparsePoints(S)
4: $G_{1,\{\text{valid,mirror,affected}\}} \leftarrow$ identifyReflection(D, I, α)
5: **if** $(n_{surface} \geq \epsilon_{n\_minSurface})$ **then**
6: $(\vec{c}_{object}) \leftarrow$ findLine($G_{1,\text{mirror}}$) ▷ get corner points of line
7: $(\alpha_{\vec{c_1}}, \alpha_{\vec{c_2}}) \leftarrow$ spanUpAngles($\vec{P}_R, \vec{c}_{object}$)
8: **for** $(i = 0, i < N_{Laser}, i{+}{+})$ **do** ▷ resort outliers, cf. Figure 6.5
9: $G_{2,\text{valid},i}, G_{2,\text{mirror},i}, G_{2,\text{affected},i} \leftarrow 0$
10: $d_{p,\text{mirror}} \leftarrow \|\vec{g}_i - \vec{c}_i\|$
11: **if** $((\alpha_i < \alpha_{\vec{c_1}}) \wedge (\alpha_i > \alpha_{\vec{c_2}}))$ **then**
12: $G_{2,\text{valid},i} \leftarrow \vec{g}_i, I_i, \alpha_i$
13: **else**
14: **if** $(d_{p,\text{mirror}} < -\epsilon_{plane})$ **then**
15: $G_{2,\text{valid},i} \leftarrow \vec{g}_i, I_i, \alpha_i$
16: **else if** $(d_{p,\text{mirror}} > \epsilon_{plane})$ **then**
17: $G_{2,\text{affected},i} \leftarrow \vec{g}_i, I_i, \alpha_i$
18: **else**
19: $G_{2,\text{mirror},i} \leftarrow \vec{g}_i, I_i, \alpha_i$
20: **end if**
21: **end if**
22: **end for**
23: $G_{out,\{\text{valid,mirror,affected}\}} \leftarrow G_{2,\{\text{valid,mirror,affected}\}}$
24: **else**
25: $G_{out,\{\text{valid,mirror,affected}\}} \leftarrow G_{1,\{\text{valid,mirror,affected}\}}$
26: **end if**
27: sendFilteredScans($G_{out,\{\text{valid,mirror,affected}\}}$)
28: **end procedure**

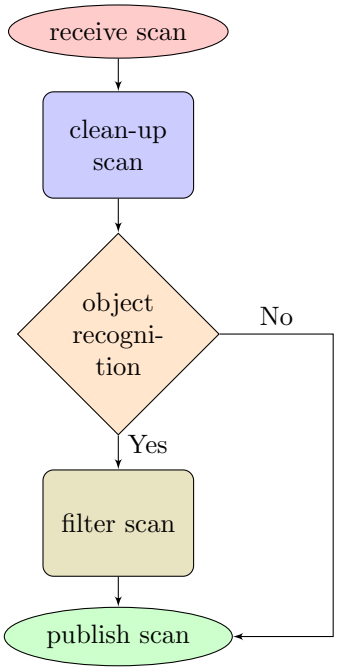


Figure 6.3: Flow chart of 2D-Pre-Filter-node.

A) Receive scan (highlighted in red):

In the first step, the 2D-Pre-Filter receives a scan tuple S with N_{Laser} scan points of Echo 1 and Echo 2

$$S = \{d_{1,i}, d_{2,i} \,|\, i = 1, \ldots, N_{Laser}\},$$

the corresponding angles

$$\alpha = \{\alpha_{1,i}, \alpha_{2,i} \,|\, i = 1, \ldots, N_{Laser}\},$$

and the corresponding intensities

$$I = \{I_{1,i}, I_{2,i} \,|\, i = 1, \ldots, N_{Laser}\}$$

from the laser scanner. Hence a laser scanner which supporting intensity and multi-echo values is required.

B) Clean-up scan (highlighted in blue):

In a second step, sparse points from the scan tuple S are removed and scan tuple

$$D = \{d_{1,i}, d_{2,i} \,|\, i = 1, \ldots, N_{Laser}\}$$

results. These isolated points, without other points nearby, are likely to be artefacts, for example from jumping edges. They appear when neighbouring measurements cross an object edge and provide discontinuity in depth, cf. Figure 6.4. To identify them, the distance between each point d_i and its neighbour d_{i+1} is compared to a threshold ϵ_{jump} by

$$f(d_i) = \begin{cases} d_i \leftarrow \text{valid} & \text{if} \quad |d_i - d_{i+1}| \leq \epsilon_{jump} \\ d_i \leftarrow \text{invalid} & \text{if} \quad |d_i - d_{i+1}| > \epsilon_{jump} \end{cases}. \tag{6.1}$$

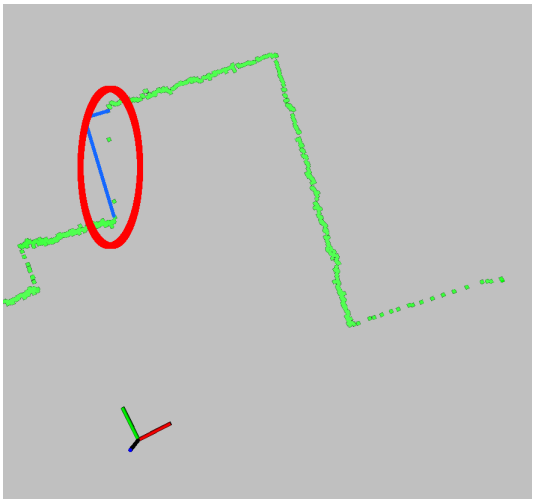


Figure 6.4: Erroneous measurements (highlighted with a red circle) caused by an edge. The real wall which is hidden for the robot, is illustrated as a blue line. The laser beam hits the corner of the wall and the measurement party results from the corner and partly from the background. That is why a jumping edge results.

C) Object recognition (highlighted in orange):

In the third step, the 2D-Pre-Filter searches for transparent and specular reflective influences. Therefore, the corresponding distance values of Echo 1 and Echo 2 in the scan tuple D are subtracted. A difference between the two echoes of a point indicates that the laser beam was influenced by a transparent or specular reflective object. The threshold ϵ_{echoes} is used to identify a mismatch. It is generally known that the distance of the first echo $d_{1,i}$ originates from the transparent or specular reflective object as it was hit first by the laser beam. It is also called "surface". The distance of the second echo $d_{2,i}$ results from a point farther away, henceforth called the "affected" point. As a result, the function *indentifyReflection()* returns three groups of points which are stored in the object type mask G_1 with m_1 and m_2 for Echo 1 and Echo 2:

$$f(m_{1,i}, m_{2,i}) = \begin{cases} m_{1,i} \leftarrow G_{\text{valid},i}, \quad m_{2,i} \leftarrow G_{\text{valid},i} & \text{if} \quad \Delta d_i \leq \epsilon_{echoes} \\ m_{1,i} \leftarrow G_{\text{surface},i}, \quad m_{2,i} \leftarrow G_{\text{affected},i} & \text{if} \quad \Delta d_i > \epsilon_{echoes} \end{cases} \tag{6.2}$$

with

$$\Delta d_i = d_{2,i} - d_{1,i}. \tag{6.3}$$

Glass fronts and mirrors are assumed to be planar. That is why a line with two end points is sufficient to model the object surface. A RANSAC-based-algorithm [Fischler and Bolles, 1981] determines these end points $\vec{c_1}$ and $\vec{c_2}$ out of the "surface" group and stores them into $\vec{c}_{object}(\vec{c}_1, \vec{c}_2)$.

This only applies if enough points are located on the surface. The number of points on the surface $n_{surface}$ is compared with a threshold $\epsilon_{n\_minSurface}$. For a positive result, the scan tuple G_1 is forwarded to proceed with step four – filter scan (highlighted in light green). If not, the points behind the surface ("affected") are simply erased as it is assumed that they result from erroneous measurements. Hence, points in the group "surface" are remasked as "valid", the object recognition for this scan finishes, and the algorithm continues with step five – publish scan (highlighted in green).

D) Filter scan (highlighted in olive):

In the fourth step which only takes place if the object recognition is successful, the scan tuple G_1 is filtered based on the determined corner points $\vec{c}_{object}$. Dirt on the object and roughness of its surface may result in an incorrect identification. For this reason, the scan is checked again to remove outliers. With robot position $\vec{P}_R$ end point $\vec{c_1}$ and the end point $\vec{c_2}$ a sector $\alpha_{(\vec{c_1},\vec{p},\vec{c_2})}$ is spanned up, cf. Figure 6.5.

$$f(\vec{g}_i, \vec{c}_i) = \begin{cases} \vec{g}_i \leftarrow \text{“in front”} & if \quad \|\vec{g}_i - \vec{c}_i\| < -\epsilon_{plane} \\ \vec{g}_i \leftarrow \text{“behind”} & if \quad \|\vec{g}_i - \vec{c}_i\| > \epsilon_{plane} \\ \vec{g}_i \leftarrow \text{“on surface”} & else \end{cases} \tag{6.4}$$

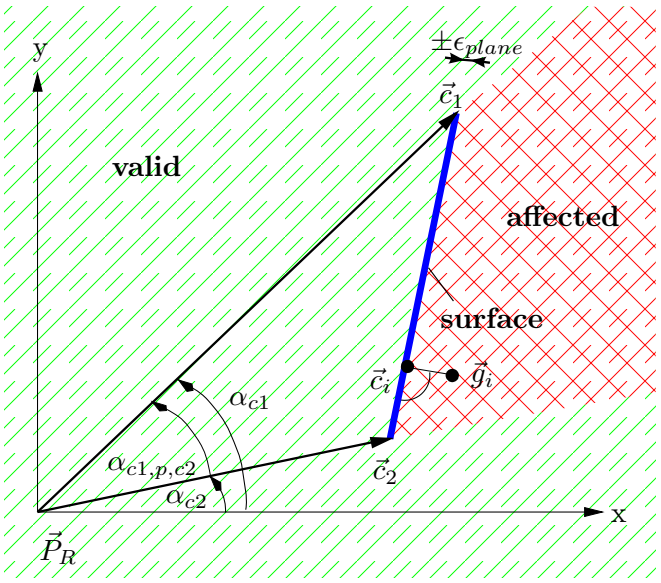


Figure 6.5: Classification of points based on the object line corners.

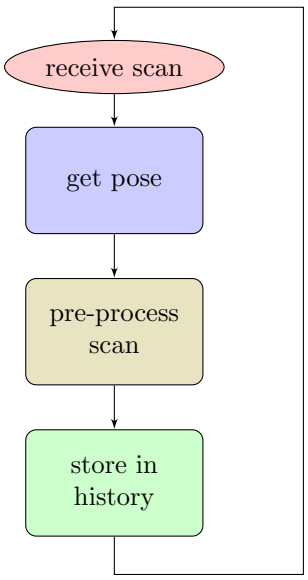
It should be mentioned that most time $\vec{P}_R$ is the origin because of the applied coordinate system. Each point $\vec{g}_i$ of the scan which is located between the corners $\vec{c_1}$ and $\vec{c_2}$, is projected onto the object plane to obtain the related perpendicular point $\vec{c}_i$ on the surface. The difference of $\vec{g}_i$ and $\vec{c}_i$ determines whether $\vec{g}_i$ is placed in front of the surface, on the surface, or behind the surface of the transparent or specular reflective object. In order to take disturbances in measurements into account, threshold ϵ_{plane} is designated to define an area (blue line in Figure 6.5) around the mirror plane. The value of ϵ_{plane} depends on the accuracy of the laser scanner. The points are remasked according to the three groups: valid, surface, and affected. “Valid” points are located in the green hatched area which is in front or next to the object. They are free from any transparent or specular reflective influences. The second group (“surface”) contains points on mirrors, window planes, and reflective metallic surfaces. They are found in the solid blue area. All remaining points can be found in the red crossed area which is behind the plane. They are assigned to the third group, henceforth called “affected” points. Resulting, the scan tuple G_{out} contains the filtered scan with the three groups.

E) Publish scan (highlighted in green):

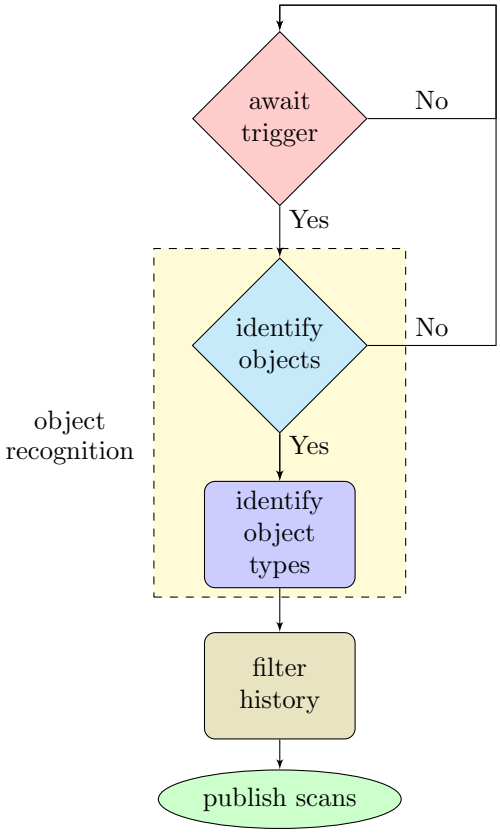
In the final step, all points in G_{out} are published to the mapping module and the Post-Filter module. In case objects are found, G_{out} contains the filtered scan tuple. Otherwise, the scan tuple resulting from the function *identifyReflections()* is stored in G_{out}.

6.1.3 Code Description of the 2D-Post-Filter

The Post-Filter which is illustrated in flow-chart 6.6 consists of two independent processed chains: storage chain and filter chain. The "storage chain" buffers the entire scan history, while the "filter chain" awaits a start signal to identify transparent and specular reflective objects. After its occurrence, the Post-Filter ceans up the entire history. As previously mentioned, two versions of the 2D-Mirror-Identifier-Approach exist. The difference lies in the object identification at the 2D-Post-Filter which is highlighted in purple at the filter chain, cf. Figure 6.6b. In compare to Version 2, V1 does not support a distinction between transparent and specular reflective objects. That is why the step "identify object types" is not included there.



(a) Flow chart of the 2D-Post-Filter Chain 1: storage chain.



(b) Flow chart of the 2D-Post-Filter Chain 2: filter chain.

Figure 6.6: Flow chart of 2D-Post-Filter-node with its storage and filter chain.

- **"Storage chain" of the 2D-Post-Filter:**

The "storage chain", see Flow-Chart 6.6a, is split into four successively processed steps: receive scan, get pose, pre-process scan, store in history. Algorithm 10 is colored according to the steps of the flow chart and explained below.

Algorithm 10 2D-Post-Filter - "storage chain"

Input: S:
S includes the scan points of Echo 1 and Echo 2, the corresponding intensities, angles, and the object mask according to the assignment of the Pre-Filter.
Output: G_1 is the history of the stored pre-processed scans.

1: **procedure** STORAGE CHAIN
2: $S_1 \leftarrow$ receiveMaskScan()
3: $\vec{P}_R \leftarrow$ requestTf($timestamp_{Scan}$)
4: $S_2 \leftarrow$ moveInWorldCoordinateSystem($S_1, \vec{P}_R$)
5: $A_{1,\{\text{mirror,affected}\}} \leftarrow$ extractCorrupted(S_2)
▷ points which are on an object surface or corrupted by the object
6: $G_{1,unchecked} \leftarrow$ storeHistories(S_2, A_1, M)
7: **end procedure**

A) Receive scan (highlighted in red):

In the first step, the Post-Filter receives a masked scan tuple

$$S_1 = \{d_{1,i}, d_{2,i}, \alpha_i, I_{1,i}, I_{2,i}, m_{1,i}, m_{2,i} \,|\, i = 1, ..., N_{Laser}\} \tag{6.5}$$

from the Pre-Filter. Where $d_{*,i}$ is the i^{th} the scan point, $*$ is the index for the corresponding Echo, α_i is the scanning angle, $I_{*,i}$ it the intensity, $m_{*,i}$ is the object type mask, and N_{Laser} is the amount of points per scan.

B) Get pose (highlighted in blue):

Following, the 2D-Post-Filter requests the pose $\vec{P}_R = (\vec{Tr}, \vec{R})$ with its current position $\vec{Tr} = (x, y, z)$ and orientation $\vec{R} = (\Phi, \Theta, \Psi)$. For the 2D case, the x and y position as well as the Θ orientation is sufficient to check as it is assumed that the robot operates in an even terrain. Therefore, z does not vary. For the 3D case also the z position and the Ψ and Φ orientation is required. The pose needs to be provided by the SLAM-node, a Localization-node, an IMU, or wheel encoders and is managed by the *ROS-TF-node*.

C) Pre-process scan (highlighted in olive):

In the third step, the robot pose $\vec{P}_R$ is used to convert the scan tuple S_1 (in robot coordinate system - RKS) into a scan tuple S_2 (in world coordinate system - WKS). Subsequently, all points which were masked as "surface" or "affected" are singled out into a respective group $A_{1,\text{surface}}$ and $A_{1,\text{affected}}$.

D) Store in history (highlighted in green):

In the final step, the scan tuples S_2, and $A_{1,*}$ are stored in a history bank G_1.

$$G_1 = \{S_{2,j}, A_{1,j,*} | \, j = 1, ..., L\}$$

where L is the length of the history. The storage chain, cf. Figure 6.6a, is repeated for every incoming scan.

- **"Filter chain" of the 2D-Post-Filter:**

The "filter chain" which is illustrated in Figure 6.6b is split into four steps: await trigger, object recognition, clean history, publish scans. To emphasize the difference between the 2D-Pre- and 2D-Post-Filter, the step "object recognition" is split into two sub-steps: identify objects and identify object types.

Algorithm 11 2D-Post-Filter - "filter chain"

Input: G_1:
G_1 is history of the stored pre-processed scans.

procedure FILTER CHAIN
  if externalTrigger() **then** ▷ e.g. external loop-closure detection
    $\vec{c}_{object} \leftarrow$ findLine($G_{A1,\text{errorSurface}}$) ▷ get corners of lines
    for $(j = 0, j < L, j$++$)$ **do**
      for $(i = 0, i < N_{Laser}, i$++$)$ **do** ▷ resort outliers, cf. Figure 6.5
        $G_{A2,\text{validPoint},j,i}, G_{A2,\text{errorSurface},j,i}, G_{A2,\text{behindErrorS},j,i} \leftarrow 0$
        $d_{p,\text{errorSurface}} \leftarrow \|\vec{c_{j,i}} - \vec{P}_R\| - g_i$
        if $((\alpha_{j,i} < \alpha_{\vec{c_1}}) \wedge (\alpha_{j,i} > \alpha_{\vec{c_2}}))$ **then**
          $G_{A2,\text{valid},j,i} \leftarrow g_{j,i}$
        else
          if $d_{p,errorSurface} < -\epsilon$ **then**
            $G_{A2,\text{validPoint},j,i} \leftarrow g_{j,i}$
          else if $d_{p,\text{errorSurface}} > \epsilon$ **then**
            $G_{A2,\text{behindErrorS},j,i} \leftarrow g_{j,i}$
          else
            $G_{A2,\text{errorSurface},j,i} \leftarrow g_{j,i}$
          end if
        end if
      end for
      $\vec{O}_{\text{type}} \leftarrow$ identifyReflectcionType(G_{A2}, $\vec{V}_{\text{distinction}}$, $\vec{c}_{\text{object}}$) ▷ see Algorithm 12
      $\vec{G}_{\text{out},*} \leftarrow$ filterHistory(G_{A2}, $\vec{O}_{\text{type}}$, $d_{\text{thres\_plane}}$, $d_{\text{thres\_visionCone}}$)
    ▷ * stands for validPoint, reflectiveSurface, transpSurface, behindReflective, behindTransparent
      sendFilteredScans($G_{\text{out},*}$)
    end for
  end if
end procedure

The "filter chain" is triggered after a reflective object was passed in order to reduce remaining errors, e.g. by a passing-algorithm or a loop-closure (see Section 6.1.4).

It should be mentioned that the Post-Filter processes the entire history bank G_1. Since the points are already transformed into the WKS there are scan points from different locations of the object. This leads to a greater amount of points on the object surface. Thus, this identification is more precise than the identification resulting from the Pre-Filter. Besides, it is possible to distinguish between the different types of object - transparent or specular reflective.

A) Await trigger (highlighted in red):

To start the "filter chain" which is illustrated in Algorithm 11, an external trigger signal from the Loop-Closure-module is required. It is assumed that the transparent or specular reflective object has been passed if the robot maneuvered a complete loop.

B) Object recognition - identify objects (highlighted in turquoise):

In the second step, transparent and specular reflective objects are identified. This step is similar to the object recognition used at the 2D-Pre-Filter (see Section 6.1.2). The function *findLine()* searches the entire history $G_{A1,\text{surface}}$. The function is based on the same RANSAC-algorithm which was already used in the Pre-Filter module. Since the entire history is used, there are more scan points placed on the object surface. The history contains scans from different positions as well. That is why the reflective object was seen from many different perspectives. It is assumed that each part of the object had been seen at least once. As a result, the RANSAC-algorithm of the 2D-Post-Filter creates a more accurate model in compare to the RANSAC-algorithm applied at the Pre-Filter. The resulting end points $\vec{c_1}$ and $\vec{c_2}$, of each object $\vec{c_{object}}$ are used together with the position of the robot $\vec{P_R}$ in order to span up a sector, similar to what was done at the 2D-Pre-Filter, cf. Figure 6.5. Subsequently, each scan point of the entire history is classified based on the "precise" model of the object surface. The points are masked as "valid", "surface", or "affected" and stored in G_2.

C) Object recognition - identify object types (highlighted in purple):

In the third step, the function *identifyReflectionType()* distinguishes between transparent and specular reflective objects, see Algorithm 12. Subsequently, three functions are used to independently determine the type of the object. Finally, the function *evaluateResults()* uses the three results to rate the final object type. Function *remaskScan()* now assigns a new object mask for each point according to its type:

* "unchecked": for points which are not checked yet
* "validPoint": for points which are not affected by transparent or specular reflective influences or been responsible for them
* "errorSurface": for points located on a transparent or specular reflective surface
* "behindErrorS": for points located behind a transparent or specular reflective surface
* "reflectiveSurface": for points located on a specular reflective surface
* "behindReflective": for points located behind a specular reflective surface
* "transpSurface": for points located on a transparent surface
* "behindTransparent": for points located behind a transparent surface
* "NaNPoint": for points which are invalid (e.g. caused by jumping edges)

Algorithm 12 2D-Pre-Filter - "filter-chain": identifyReflectionType()

Input: G_{A2}, $\vec{V}_{\text{distinction}}$, $\vec{c}_{\text{object}}$:
G_{A2} is a tuple of scans of the entire history, the vector $\vec{V}_{\text{distinction}}$ contains all necessary variables, and $\vec{c}_{\text{object}}$ contains the corner points of the lines.

Output: $\vec{O}_{\text{type}}$:
$\vec{O}_{\text{type}}$ is a vector containing the type of objects.

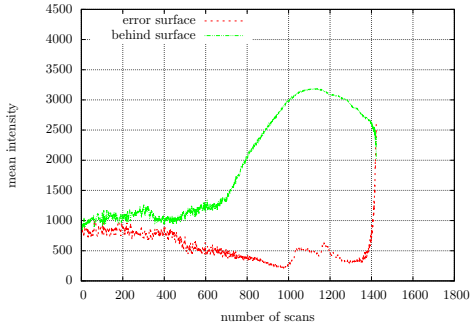
function IDENTIFYREFLECTIONTYPE()
$m_{\text{IntensFact}} \leftarrow \text{meanIntensFactorCheck}(G_{A2}, \vec{V}_{\text{distinction}})$
$m_{\text{Transf}} \leftarrow \text{transformationCheck}(G_{A2}, \vec{V}_{\text{distinction}}, \vec{c}_{\text{object}})$
$m_{\text{intensDiscon}} \leftarrow \text{checkDiscontinuity}(G_{A2}, \vec{V}_{\text{distinction}})$ ▷ see Algorithm 13
$\vec{O}_{\text{type}} \leftarrow \text{evaluateResults}(m_{\text{IntensFact}}, m_{\text{transf}}, m_{\text{discon}})$
end function

Determination of mean intensity factor:
The first function (*meanIntensFactorCheck()*), identifies the relation of intensity values on the object (Echo 1) and intensity values behind the surface (Echo 2). This is described by the material value $f_{material}$. The function calculates the mean intensity of both echoes by

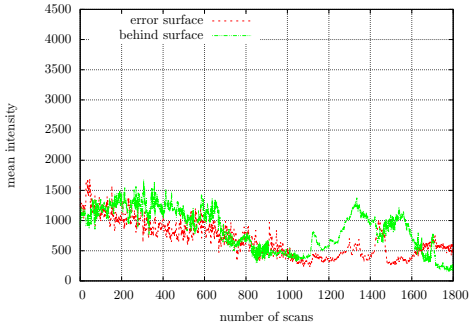
$$\hat{I}_* = \frac{\sum_{i=1}^{N_*} I_{*,i}}{N_*}. \tag{6.6}$$

Thus, $f_{material}$ follows as

$$f_{material} = \frac{\hat{I}_{behindErrorS}}{\hat{I}_{errorSurface}}. \tag{6.7}$$



(a) Median intensity values of a transparent object at a drive by scenario. Green are median intensity values of points behind the transparent plane ("behindTransparent") and red are median intensity values of points on the transparent plane ("transpSurface").

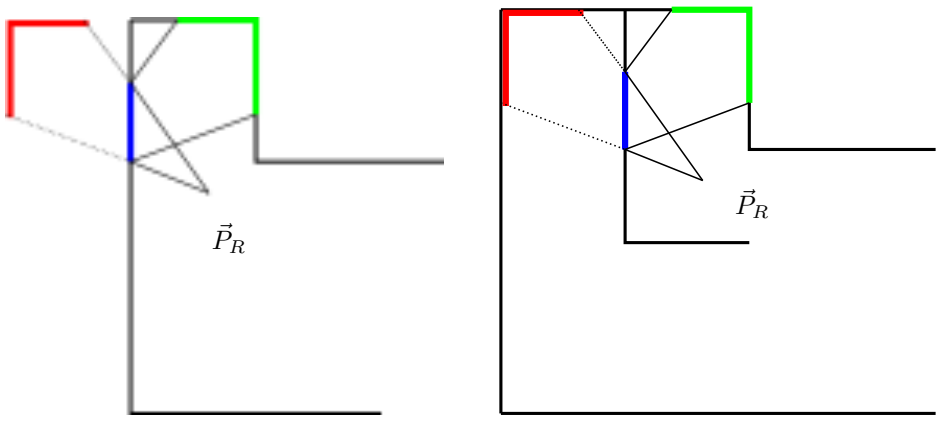


(b) Median intensity values of a specular reflecitve object at a drive by scenario. Green are median intensity values of points behind the transparent plane ("behindTransparent") and red are median intensity values of points on the transparent plane ("transpSurface").

Figure 6.7: Median intensities of each scan after a transparent object (a) and a specular reflective object (b) was bypassed.

Figure 6.7a illustrates the intensity values of a transparent object and Figure 6.7b illustrates the intensity values of a specular reflective object at a drive by scenario (see Section 7.2). For transparent objects the factor $f_{material} > 1$ because most of the intensity values of the points behind the object are greater than the intensity values of the points on the object surface. For specular reflective objects $f_{material} \leq 1$ because the intensity values on the object surface are greater than the ones behind the surface. The two charts illustrate the necessity to monitor the course of the two medians. To investigate the intensity, values of a single scan can lead to a wrong identification. The threshold $\epsilon_{intensFact}$ is used to rate the result and determine $m_{intensFact}$ ($m_{intensFact} = 0 \Rightarrow$ "reflective", and $m_{intensFact} = 1 \Rightarrow$ "transparent").

Evaluation of a valid transformation:
The second function (*transformationCheck()*) examines if there is a symmetry w.r.t. the identified object. It uses the points located behind the object and back-projects them w.r.t. to the identified object line (see Figure 6.8a). Following, these points $D = \{d_i | i = 1, \ldots, N_D\}$ and the points rated as "valid" $M = \{d_j | j = 1, \ldots, N_M\}$ are matched by an Iterative Closest Point algorithm (ICP)[AutonOHM, 2017]. In case of a positive result the translation $\vec{Tr}$ and the rotation $\vec{R}$ of the resulting transformation matrix $\vec{T} = (\vec{Tr}, \vec{R})$ is compared to a threshold $\epsilon_{trans} = (\epsilon_{transX}, \epsilon_{transY}, \epsilon_{trans\Phi})$. In case of small displacements, it is assumed that the plane has reflective properties, otherwise the object is rated to be transparent. The result is stored in m_{trans} ($m_{trans} = 0 \Rightarrow$ "reflective", and $m_{trans} = 1 \Rightarrow$ "transparent"). This method suffers from three drawbacks. First, locations with symmetry might result in wrong identification (cf. Figure 6.8b). Second, if the scan does not include the area where the back-projection is located there will be no valid transformation. Third, non-planar reflective surfaces will not result in a transformation. For them, the points have to be back-projected differently. Nevertheless, by fusing the result with the other two object identification methods, a more reliable distinction is achieved.

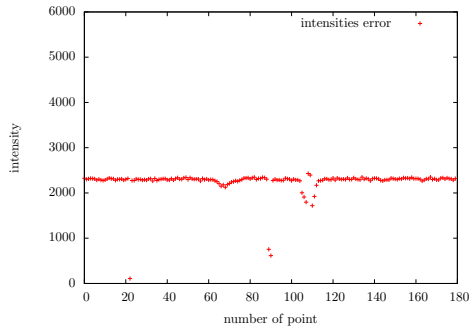


(a) Valid back-projection.

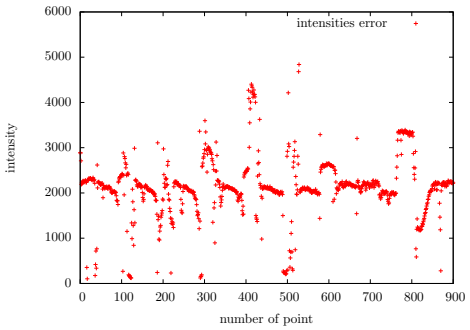
(b) Failed back-projection due to a symmetrical environment. The object (blue) can be a reflective, but also transparent.

Figure 6.8: Back-projection (green) of points behind a surface (red) w.r.t. the surface (blue) and the position of the robot $\vec{P}_R$.

Check for discontinuities:
The third function (*checkDiscontinuity()*) searches at intensity values of Echo 1 (points on the erroneous surface) if neighbouring scan points vary strongly, illustrated in Figure 6.9. To do so, it searches the entire history G_{A2} of scan points according to Algorithm 13. The algorithm compares each intensity value $I_{*,i}$ with its neighbour $I_{*,i+1}$. If the intensity value differs more than the threshold ϵ_{discon}, the point is rated as a mirror and the counter $cnt_{reflective}$ increases. Otherwise, the counter $cnt_{transparent}$ increases. Finally, the maximum value of $cnt_{reflective}$ and $cnt_{transparent}$ determines m_{discon} ($m_{discon} = 0 \Rightarrow$ "reflective", and $m_{discon} = 1 \Rightarrow$ "transparent").



(a) Intensity values of points behind a transparent object remain constant.



(b) Intensity values of points behind a specular reflective object vary strong. For a better illustration intensity values above 6.000 were cut off. The maximum intensity value of a mirrored point was 13.000.

Figure 6.9: Variation of intensity values behind a transparent object vs. intensity values behind a specular reflective object.

Algorithm 13 2D-Post-Filter - "filter chain": checkDiscontinuity

```
1:  procedure CHECKDISCONTINUITY
2:      for (n = 0, n < N, n++) do          ▷ N is amount of scan points in the history
3:          if (|I_{1,i} − I_{1,i+1}| ≥ ±ε_discon) then
4:              cnt_reflective++;
5:          else
6:              cnt_transparent++;
7:          end if
8:      end for
9:      if cnt_reflective < cnt_transparent then
10:         m_discon ← "transparent";
11:     else
12:         m_discon ← "reflective";
13:     end if
14: end procedure
```

Evaluate results:
In the end of Algorithm 12, the function *evaluateResults()* determines the final type of the object according to the most likelihood of $m_{intensFact}, m_{trans}$, and m_{discon} as

$$f(m_{final}) = \begin{cases} m_{final} = \text{“reflective”}, & \text{if} \quad \frac{\sum m_*}{3} < 2 \\ m_{final} = \text{“transparent”} & \text{if} \quad \frac{\sum m_*}{3} \geq 2 \end{cases} \tag{6.8}$$

with * stands for *intensFact*, *trans*, and *discon*, the value 0 stands for “reflective”, and the value 1 stands for “transparent”.

Next, the function *identifyReflectionType()* returns the results of each object $\vec{O}_{\text{type}}$ to the “filter chain“ (Algorithm 11).

D) Filter history (highlighted in olive):

In the following step, the 2D-Post-Filter (see Algorithm 11) continues to clean the entire history G_{A2} according to the determined object types $\vec{O}_{type}$. It uses the function *spanUpAngles()* which was previously described, cf. Figure 6.5. The resulting tuple $G_{\text{out},*}$ contains points with the final object mask: “validPoint”, “reflectiveSurface”, “transparentSurface”, “behindReflective”, “behindTransparent”, or “nanPoint”.

E) Publish scans (highlighted in green):

Finally, in the last step of the 2D-Post-Filter, the different scan messages are published. The first message is called “valid” and represents real objects. It contains points $G_{\text{out,valid}}$, but also points on the mirror plane $G_{\text{out,mirror}}$, points on the transparent plane $G_{A2,\text{transparent}}$, and points behind the transparent plane $G_{\text{out,transparent\_affected}}$. In addition, four messages (called “mirror”, “transparent”, “mirror_affected”, and “transparent_affected”) are published with their corresponding points. The points $G_{\text{out,mirror\_affected}}$ are not used yet. These points are caused by objects which are mirrored on the specular reflective surface. As shown in Figure 6.8a, they can be back-projected and therefore also used to update the map.

6.1.4 Code Description of the Loop-Closure-module

As described above, the visibility of transparent and specular reflective objects depends on the angle between the incoming laser beam and the surface of the object. It is assumed that the object was bypassed and therefore seen completely over time if the robot completes a loop. Therefore, the Post-Filter builds a history of all points. It should be mentioned that the Loop-Closure-node can be exchanged by any other algorithm which detects if a transparent or specular reflective object was bypassed. Nevertheless, this simple loop-closure detection is sufficient to demonstrate the usability of the 2D-Mirror-Identifier-Approach.

The flow chart of the Loop-Closure-node is illustrated in Figure 6.10. It can be simplified into four steps: receive pose, store in history, check previous locations, publish closed loop.

A) Receive position and orientation (highlighted in red):

In the first step, the node receives the current pose $\vec{P}_R = (\vec{Tr}, \vec{R})$ with its position $\vec{Tr} = (x, y, z)$ and orientation $\vec{R} = (\Phi, \Theta, \Psi)$. This needs to be provided e.g. by the SLAM-node, a Localization-node, an IMU, or wheel encoders. Based on the applied sensor (IMU, wheel encoders, etc.) or algorithm (SLAM) the accuracy varies. This Loop-Closure-node is applicable for 2D and 3D detection, as it respects all three directions and orientations.

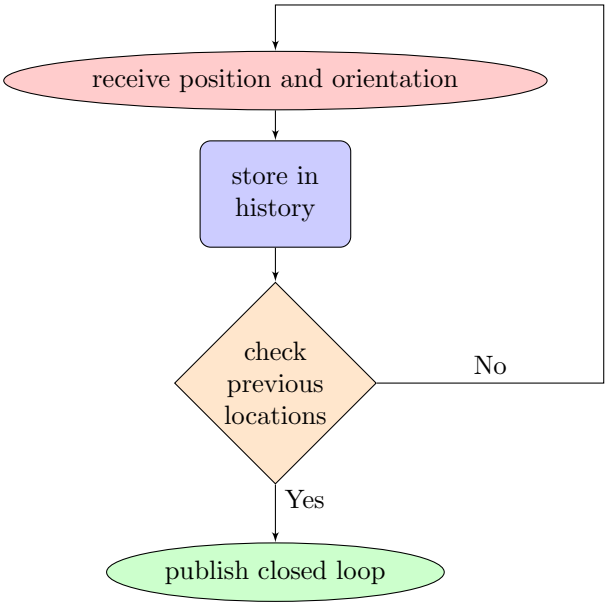


Figure 6.10: Flow chart of Loop-Closure-node.

B) Store in history (highlighted in blue):

In the second step, the robot pose $\vec{P}_R$ and the current time stamp t are stored into a history $\vec{H}_{pose}$.

C) Check previous locations (highlighted in orange):

Following, the entire history $\vec{H}_{pose}$ is searched for matches with the current pose $\vec{P}_R$. To prevent an immediate closed loop signal after the robot has moved, only poses are taken into account which are older than n_{pose} poses. To respect inaccuracies, a threshold $\epsilon_{nearby,*}$ (* stands for x, y, z) is used to rate a closed loop by:

$$f(closed\_loop) = \begin{cases} closed\_loop = \text{true} & \text{if} \quad \begin{aligned} &\epsilon_{nearby,x} < |x_{current} - x_{previous}| \wedge \\ &\epsilon_{nearby,y} < |y_{current} - y_{previous}| \wedge \\ &\epsilon_{nearby,z} < |z_{current} - z_{previous}| \end{aligned} \\ closed\_loop = \text{false} & \text{if} \quad \begin{aligned} &\epsilon_{nearby,x} \geq |x_{current} - x_{previous}| \vee \\ &\epsilon_{nearby,y} \geq |y_{current} - y_{previous}| \vee \\ &\epsilon_{nearby,z} \geq |z_{current} - z_{previous}| \end{aligned} \end{cases} \tag{6.9}$$

The orientation of the robot is not considered as it is not important for the scenarios in Section 7.

D) Publish closed loop (highlighted in green):

In case of a positive result, the fourth step is processed and the node publishes a closed loop signal. If there is no positive match with the previous position, the node returns to Step 1 (highlighted in red) and waits for the next pose.

6.1.5 Code Description of the Customized TSD-SLAM

The TSD-SLAM, as it was described in Section 4.4, is sufficient to use straight away for the 2D-Mirror-Detector-Approach (V1 of 2D-Mirror-Identifier-Approach). Since it is not possible to manipulate individual points in the map afterwards, the TSD-SLAM needed to be customized. This makes it possible to erase and insert points into the map which are supported by the 2D-Post-Filter-module.

As mentioned above, all points behind a transparent or specular reflective object are filtered out by the 2D-Pre-Filter. In case of a specular reflective object, this is correct. In case of a transparent object, the points should remain but they are erased precautionary. The reason for this is to make it possible to distinguish between transparent and specular reflective objects with the 2D-Pre-Filter. These precautionarily erased points should now be inserted after the 2D-Post-Filter identified the type of the object. In contrast, points behind the surface which are not identified with the 2D-Pre-Filter should now be erased from the map. This section describes the required changes in the TSD-SLAM in order to do so. This is exemplaryly implemented using TSD-SLAM, but also can be integrated into other mapping approaches to support post-filtering of transparent or specular reflective objects. To be able to fulfil these mentioned requirements, two steps need to be carried out: subscribe multiple scans and manipulate the existing map.

First, the TSD-SLAM needs to subscribe a new scan message. It should be able to receive messages from the 2D-Pre-Filter and from the 2D-Post-Filter. The messages from the 2D-Pre-Filter are used to build the preliminary map. This is already implemented in the standard TSD-SLAM. In contrast, the messages from the 2D-Post-Filter are used to update the map according to the identified object. Therefore, points need to be manipulated (inserted or erased) by overwriting the TSD-function values individual.

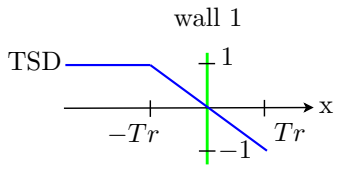
Second, the grid has to distinguish between the two messages. In case of a 2D-Pre-Filter message, the regular function *push()* (already existed) is processed. In case of a 2D-Post-Filter message, the function *pushForceIn()* (newly implemented) is processed. *pushForceIn()* distinguishes based on the value of the assigned object type. It erases the point from the existing map or inserts the point into it.

As mentioned before, a point in the TSD-SLAM map does also affect his surrounding area, called truncation radius $\pm Tr$, cf. Figure 6.11a. This area is represented by the Truncated-Signed-Distance-function (TSD-function) (blue line in Figure 6.11) which has a value between ± 1. The point of the wall appears at the location of the zero crossing (green line). Therefore, it is necessary to overwrite the TSD values of this area when inserting or erasing a point. Figure 6.11b illustrates an erased wall point (dashed green line). In contrast, Figure 6.11c illustrates an inserted point (green line "wall 1").

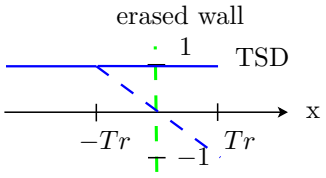
In this case, a drawback of the applied TSD-function in the TSD-SLAM occurs. The raycast will detect two walls next to each other because there are two zero crossings, cf. Figure 6.12. For "wall 1" the TSD values in front the wall shrinks and at "wall 2" a jump between $TSD = -1$ and $TSD = 1$ occurs. The effect in the map is illustrated in Figure 6.12 ("wall 1" and "wall 2").

This effect can also occur if a wall was seen from two sides, cf. Figure 6.11d. First, the surface of the wall (wall 1) was seen from the left side ($TSD > 1$). Second, the surface of the wall (wall 2) was seen from the right side ($TSD > 1$). Behind both surfaces the TSD value is smaller than zero. If the distance between the two surfaces is greater than $2 \cdot Tr$, the TSD value between is undefined (red hatched). This is the case for most walls in buildings.

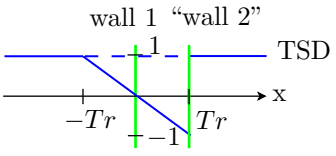
In contrast to *pushForceIn()* (the forced inserting or erasing of values), the function *push()* weighs the past TSD values and the new values. That is why the mapping of the TSD-SLAM is dynamic. It is understood that there will be a overlay of both functions if the distance between the two walls (one from the front, one from the back) is thinner than $2 \cdot Tr$. In such a case, a wall disappears for a short time and appears again which is another side-effect of the TSD-SLAM.



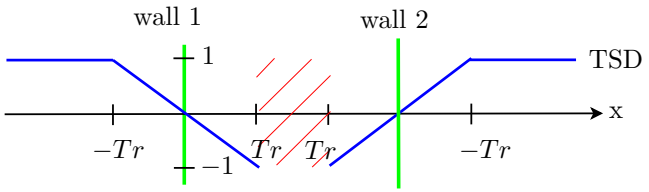
(a) In blue, the TSD-function after a point was inserted regularly and seen from one side. The wall is illustrated as a green line.



(b) In blue, the TSD-Function after a point was erased afterwards. The erased wall is illustrated as a green dashed line and the old TSD-Function values are illustrated as a blue dashed line.

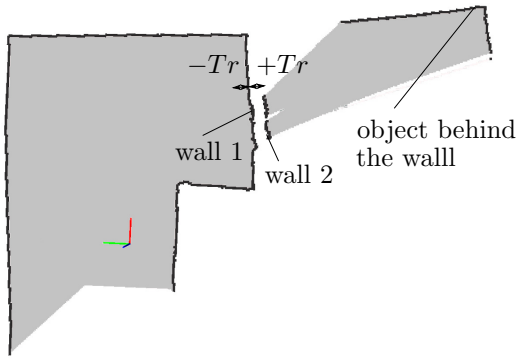


(c) In blue, the TSD-Function after a point was inserted afterwards. The two walls are illustrated as green lines and the old TSD-Function values as a dashed blue line.

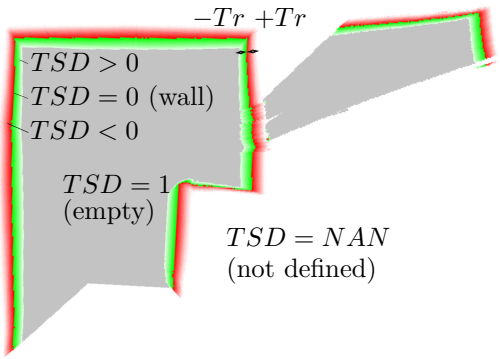


(d) In blue, the TSD-Function after a point was inserted regularly and seen from two sides. The two resulting walls are illustrated as a green line while the undefined TSD-region is marked by a red hatched area.

Figure 6.11: Various possibilities of wall occurance, based on the TSD-function value.



(a) Effect of a double wall, after a point (located on wall 1) was inserted afterwards.



(b) TSD-function values after a point was inserted retrospective.

Figure 6.12: Map and TSD-function values with a double wall caused by inserting a point retrospective.

6.2 3D-Mirror-Identifier-Approach

This section presents the 3D-Mirror-Identifier-Approach for 3D mapping. First, a processing chain illustrates the data flow between the four modules: 3D-Pre-Filter, Mapping, 3D-Post-Filter, and Loop-Closure. Following, the function of each module is described in detail. All modules are implemented as ROS-nodes and available at http://www.github.com/autonohm/ohm_mirror_detector_3D.git.

6.2.1 Processing Chain of 3D-Mirror-Identifier-Approach

The processing chain of the 3D-Mirror-Identifier-Approach is illustrated in Figure 6.13. The difference lies mainly in an additional axis – the z-axis. The four modules are marked by blue rectangles. Since they are implemented as ROS-nodes, an easy exchange to test with other implementations is possible. The required messages to interact with each other are marked by white lozenges.

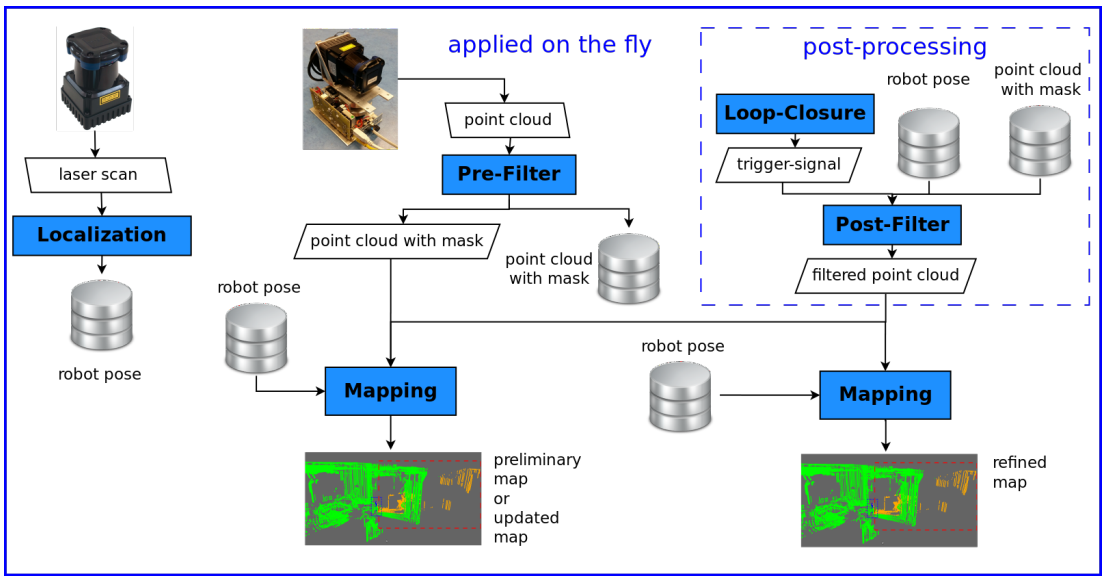


Figure 6.13: Processing chain of the 3D-Mirror-Identifier-Approach: Pre-filtering removes affections on the fly. Post-filtering refines the results. Based on the Mapping-module, a refined map or an updated map is built. The Localization-module is required if there is no localization is included in the Mapping-module. The Loop-Closure sends a trigger-signal to start the Post-Filter-process.

6.2.2 Code Description 3D-Pre-Filter

The 3D-Pre-Filter, cf. Figure 6.14, can be simplified in five steps: receive point cloud, clean-up point cloud, object recognition, filter point cloud, and publish point cloud. In compare to the 2D-Pre-Filter, the 3D-Pre-Filter is capable to distinguish between transparent and specular reflective objects because the point cloud contains more measurements on the surface. That is why the object recognition is split into two sub-steps (identify reflections and identify objects), similar to the object recognition of the 2D-Post-Filter. Algorithm 14 describes the 3D-Pre-Filter in detail and is highlighted according to the steps shown in the flow chart.

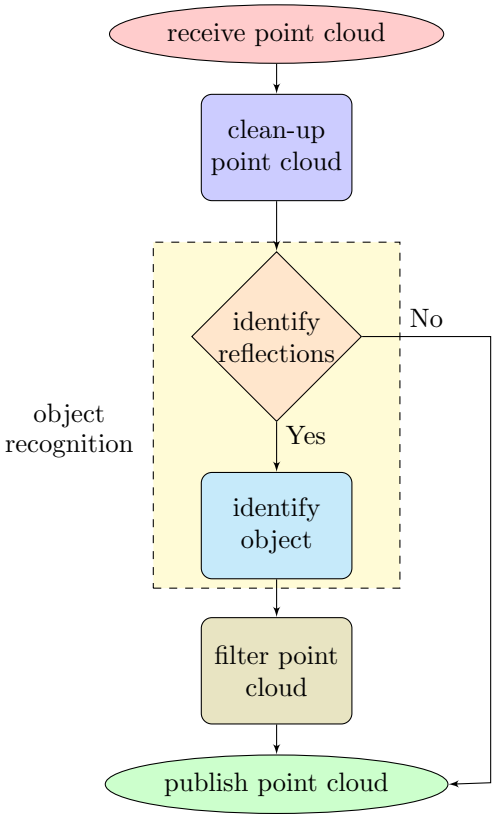


Figure 6.14: Flow chart of 3D-Pre-Filter.

Algorithm 14 3D-Pre-Filter

Input: $\vec{S}$:
$\vec{S}$ includes the point cloud with points of Echo 1 and Echo 2 as well as their corresponding values. Each point has the distance $d_{*,i}$, an angle α_i, xyz-coordinates $\vec{c}_*$, a normal-vector $\vec{n}_{0,*,i}$, an intensity $I_{*,i}$, and a masks $m_{*,i}$ with * is standing for Echo 1 and Echo 2.

Output: $\vec{G}_{out}$:
$\vec{G}_{out}$ contains multiple messages (valid, surface, affected). The message $\vec{G}_{\text{valid}}$ includes the valid scan points with their corresponding attributes and scan points on the surface (d, α, $\vec{c}$, $\vec{n}_0$, I, and m). $\vec{G}_{\text{surface}}$ includes the scan points, located on the surface of the transparent or specular reflective object, with their corresponding values. $\vec{G}_{\text{affected}}$ includes the scan points, located behind the surface of the transparent or specular reflective object, with their corresponding values.

1: **procedure** 3D-Pre-Filter
2: $\vec{G}_{\text{in}} \leftarrow$ receiveScanTuple($\vec{S}$)
3: $\vec{G}_1 \leftarrow$ distanceThresFilter($\vec{G}_{\text{in}}$, d_{min}, d_{max})
4: $\vec{G}_2 \leftarrow$ boxFilter($\vec{G}_1$, b)
5: $\vec{G}_3 \leftarrow$ outlierFilter($\vec{G}_2$, n_{inlier}, r, '*unchecked*')
6: $\vec{G}_4, n_{\text{affected}} \leftarrow$ identifyReflections($\vec{G}_3$, n_{inlier}, r, '*errorSurface*')
7: **if** ($n_{\text{affected}} \geq \epsilon_{\text{affected\_min}}$) **then**
8: $\vec{G}_5 \leftarrow$ outlierFilter($\vec{G}_4$, '*errorSurface*')
9: $\vec{o} \leftarrow$ separateObject($\vec{G}_5$, $\vec{V}_{\text{distinction}}$, n_{object}) ▷ see Algorithm 15
10: $\vec{G}_{\text{out}} \leftarrow$ cleanScanTuple($\vec{G}_5$, $\vec{o}$, $\epsilon_{\text{thres\_plane}}$, $\epsilon_{\text{thres\_visionCone}}$)
11: **else**
12: $\vec{G}_{\text{out}} \leftarrow \vec{G}_3$
13: **end if**
14: sendScanTuple($\vec{G}_{\text{out}}$)
15: **end procedure**

A) Receive point cloud (highlighted in red):

In the first step, the Pre-Filter receives a point cloud

$$\vec{S} = \{\vec{p}_{1,i}, \vec{p}_{2,i} | i = 1, \ldots, N_{cloud}\}$$

of Echo 1 and Echo 2 with N_{cloud} points. Each scan point $\vec{p}$ contains its corresponding coordinates

$$\vec{c} = \{\vec{c}_{1,i}, \vec{c}_{2,i} \,|\, i = 1, \ldots, N_{cloud}\},$$

its corresponding normals

$$\vec{n} = \{\vec{n}_{1,i}, \vec{n}_{2,i} \,|\, i = 1, \ldots, N_{cloud}\},$$

its corresponding distances to the origin (the laser scanner)

$$d = \{d_{1,i}, d_{2,i} \,|\, i = 1, \ldots, N_{cloud}\},$$

and its corresponding intensities

$$I = \{I_{1,i}, I_{2,i} \,|\, i = 1, \ldots, N_{cloud}\}.$$

The object type mask

$$m_{\text{type}} = \{m_{1,i}, m_{2,i} \,|\, i = 1, \ldots, N_{cloud}\}$$

is added and the value "unchecked" is assigned. It is used to classify the points as

* "unchecked": for points which are not checked yet
* "validPoint": for points which are not affected by transparent or specular reflective influences or be responsible for them
* "errorSurface": for points located on a transparent or specular reflective surface and the type of the surface is not distinguished yet
* "behindErrorS": for points located behind a transparent or specular reflective surface and the type of the surface is not distinguished yet
* "reflectiveSurface": for points located on a specular reflective surface
* "behindReflective": for points located behind a specular reflective surface
* "transpSurface": for points located on a transparent surface
* "behindTransparent": for points located behind a transparent surface
* "nanPoint": for points which are not valid

B) Clean-up point cloud (highlighted in blue):

In the second step, the point cloud is filtered by a threshold filter (function *distanceThresFilter()*), a box filter (function *boxFilter()*), and an outlier filter (function *outlierFilter()*). First, the threshold filter eliminates points which are out of the scanning range of the laser scanner.

$$f(d_1, d_2) = \begin{cases} d_{1,i} \leftarrow d_{1,i} & \text{if} \quad (d_{1,i} \geq \epsilon_{minDist}) \wedge (d_{1,i} \leq \epsilon_{maxDist}) \\ d_{1,i} \leftarrow NaN & \quad else \\ d_{2,i} \leftarrow d_{2,i} & \text{if} \quad (d_{2,i} \geq \epsilon_{minDist}) \wedge (d_{2,i} \leq \epsilon_{maxDist}) \\ d_{2,i} \leftarrow NaN & \quad \text{else} \end{cases} \tag{6.10}$$

Then, the box filter sets a box $b(x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max})$ around the robot location to eliminate points which represent the robot. This is illustrated in Figure 6.15.

$$f(p_1, p_2) = \begin{cases} p_{1,i} \leftarrow p_{1,i},\, p_{2,i} \leftarrow p_{2,i} & \text{if} \quad ((p_{1,x} \leq x_{min}) \vee (p_{1,x} \geq x_{max})) \\ & \quad \wedge ((p_{1,y} \leq y_{min}) \vee (p_{1,y} \geq y_{max})) \\ & \quad \wedge ((p_{1,z} \leq z_{min}) \vee (p_{1,z} \geq z_{max})) \\ p_{1,i} \leftarrow NaN,\, p_{2,i} \leftarrow NaN & \quad \text{else} \end{cases} \tag{6.11}$$

Last, the outlier filter searches for points without a predefined number of neighbours n_{inliner} in a radius r. It is based on the *RadiusOutlierRemoval*-function of the Point Cloud Library (PCL) [PCL, 2017]. It is assumed that such outliers result e.g. from jumping edges. It can be compared to the step "clean-scan" of the Pre-Filter in Section 6.1.2. The result of this process is illustrated in Figure 6.16.

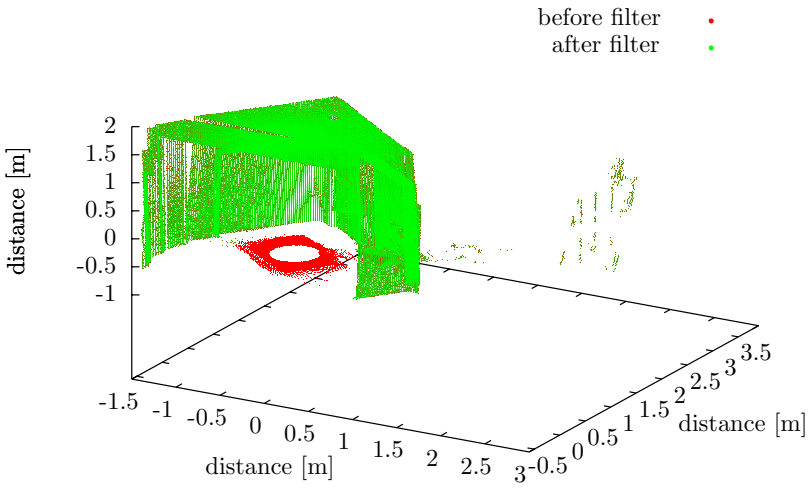


Figure 6.15: Result of the *boxFilter()*-function. The scanner was placed on a table which represents the robot. Red are the points before filtering and green are the points after the filter process. It can be seen that the points from the table are eliminated.

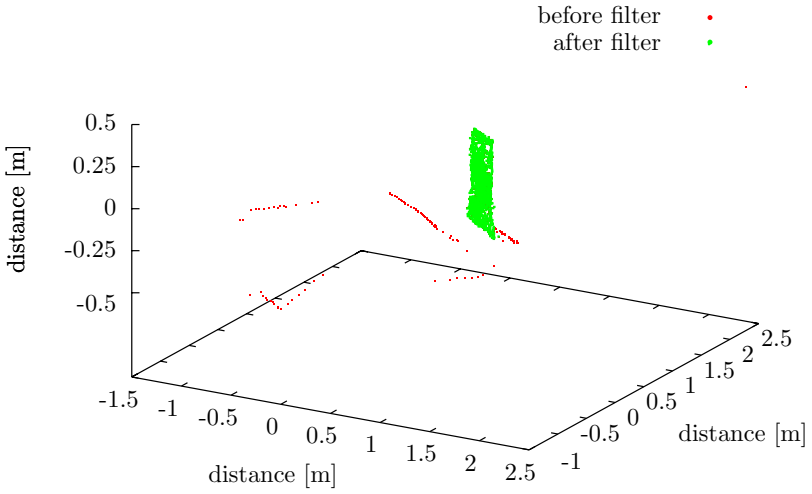


Figure 6.16: Result of the *outlierFilter()*-function exemplarily applied at the "surface points". Red are the points before filtering and green are the points after the filter process. It can be seen that the single points which resulted from angles in the environment are eliminated.

C) Object recognition (highlighted in yellow):

The object recognition (highlighted in yellow) which is the third step, contains two sub-steps to outline the difference to the 2D-Pre-Filter: identify reflections (highlighted in orange) and identify objects (highlighted in turquoise).

C.1) Object recognition - identify reflections (highlighted in orange):

In the first sub-step, the function *identifyReflections()*, searches the entire point cloud $\vec{G}_3$ for transparent and specular reflective influences and classifies the points. For this reason, the distance values of Echo 1 and Echo 2 are subtracted, compared to a threshold ϵ_{echoes}, and assigned to

$$f(m_1, m_2) = \begin{cases} m_{1,i} \leftarrow \text{“valid”}, m_{2,i} \leftarrow \text{“valid”} & \text{if} \quad \Delta d_i \leq \epsilon_{echoes} \\ m_{1,i} \leftarrow \text{“errorSurface”}, m_{2,i} \leftarrow \text{“behindErrorS”} & \text{if} \quad \Delta d_i > \epsilon_{echoes} \end{cases} \tag{6.12}$$

with

$$\Delta d_i = d_{2,i} - d_{1,i}. \tag{6.13}$$

The threshold ϵ_{echoes} is applied to take inaccuracies into account.

If the amount of affected points n_{affected} is smaller than a threshold $\epsilon_{\text{affected\_min}}$, the points are not used anymore. Then the 3D-Pre-Filter jumps to its last step (highlighted in green) and broadcasts the cleaned scan G_{out}. If there are enough points on the surface, the algorithm continues to search for objects.

C.2) Object recognition - identify object (highlighted in turquoise):

In the second sub-step, the points assigned to the transparent or specular reflective surface (“errorSurface”) are filtered by an outlier filter (*outlierFilter()*). This function was already applied in Step 2 (highlighted in blue) to all points. There, it eliminated errors, e.g. from jumping edges. This time, it is applied to eliminate single points in the group “errorSurface”. This was exemplarily illustrated in Figure 6.16. Only clusters with more than $\epsilon_{affected\_min}$ remain. Each cluster is rated as a potential object. The function *seperateObject()*, see Algorithm 15, identifies the object parameters of each cluster and determines the object type as long as the amount of “affected” points n_{affected} is greater than its threshold $\epsilon_{\text{affected\_min}}$.

To do so, the function *identifyPlanes()* determines the plane parameters and its corners based on the Singleton-Arc-Consistency-Segmentation-algorithm (SAC) of PCL [PCL, 2017]. First, the sub-function *EuclideanClusterExtraction()* searches the KD-tree representation of the point cloud $\vec{G}_5$ for clusters. It creates an empty list of clusters C and an empty queue Q. For each point $\vec{p}_i \in \vec{P}_5$ it adds it to the queue Q and searches for a set P_i^k of neighbours for $\vec{p}_i$ with $r_{p_i} \leq \epsilon_{dist\_thres}$. Here, r_{p_i} stands for the sphere radius around the point p_i and ϵ_{dist_thres} is the maximal distance between the point $\vec{p}_i$ and its neighbour. If the neighbours $\vec{p}_i^{\,k} \in \vec{P}_i^k$ are not processed yet, then they are added to Q. After all points of Q have been processed, Q is added to the cluster C_k with the current number of cluster k. The second sub-function uses the *SACSegmentation()* to determine planes in the list of clusters C. Then, the next sub-function which is based on a moment of inertia analysis of the PCL feature extractor, determines successively the corners $\vec{c}_1$, $\vec{c}_2$, $\vec{c}_3$, and $\vec{c}_4$, as well as the center point $\vec{p}_{center}$ of each identified plane. They are stored into a list of objects $\vec{o}$. It calculates the covariance matrix M_{moi} of the points $\vec{p}_i$ ($\vec{p}_i \in i, \ldots, N_{clusterK}$) of the current cluster C_k. $N_{clusterK}$ stands for the amount of points in this cluster. Then, its normalized Eigen values and vectors are extracted and processed in an iterative process. Each major Eigen vector is rotated and performed around

Algorithm 15 3D-Pre-Filter: separateObject()

Input: $\vec{G}_5$, $\vec{V}_{\text{distinction}}$, n_{object}:
$\vec{G}_5$ is the scan cloud tuple resulting from the function *outlierFilter()*. n_{object} is the minimal amount of points to identify an object. $\vec{V}_{\text{distinction}}$ is a vector containing all variables for the object distinction.

Output: $\vec{o}$:
$\vec{o}$ is a vector of objects with its properties (xyz-coordinates of corners, type of object, width, length, plane function parameters).

1: **function** SEPARATEOBJECT()
2: **while** ($n_{\text{affected}} \geq \epsilon_{\text{affected\_min}}$) **do**
3: $\vec{o} \leftarrow$ indentifyPlanes($\vec{G}_5$, $\epsilon_{\text{thres\_plane}}$, n_{object})
4: $\vec{m}_{\text{objectType}} \leftarrow$ analyzeObjectType($\vec{G}_5$, $\vec{V}_{\text{distinction}}$, $\vec{o}$) ▷ see Algorithm 16
5: **end while**
6: $\vec{o} \leftarrow \vec{m}_{\text{objectType}}$
7: **end function**

the others to provide the invariance to rotation of the point cloud. For every current axis the moment of inertia and the eccentricity is calculated. For this reason, the current vector is treated as the normal vector of the plane and the cluster C_k is projected onto it. After the eccentricity is calculated for this projection, the function determines the plane center p_{center}, the vectors $\vec{a}_x, \vec{a}_y,$ and $\vec{a}_z$ to span up a coordinate system related to the plane, cf. Figure 6.17. Further, it calculates, based on the outer bounding box (OBB), the coordinates of the maximal and minimal points of the plane $\vec{px}_{max}, \vec{px}_{min}, \vec{py}_{max}, \vec{py}_{min}, pz_{max},$ and pz_{min}. The OBB sets a bounding box, orientated according to the Eigen values, and takes the maximum values as a reference for the frame.

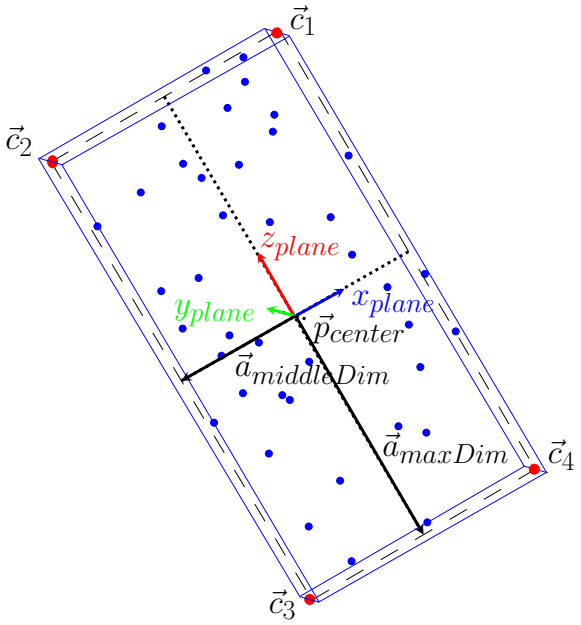


Figure 6.17: Corner determination for 3D objects.

Following, the dimensions

$$\begin{aligned} dim_x &= \|\vec{px}_{max} - \vec{px}_{min}\|, \\ dim_y &= \|\vec{py}_{max} - \vec{py}_{min}\|, \quad and \\ dim_z &= \|\vec{pz}_{max} - \vec{pz}_{min}\| \end{aligned} \tag{6.14}$$

are calculated and the four corners of the plane

$$\begin{aligned} \vec{c}_1 &= \vec{p}_{center} + 0.5 \cdot (\vec{a}_{maxDim} + 0.5 \cdot \vec{a}_{middleDim}), \\ \vec{c}_2 &= \vec{p}_{center} - 0.5 \cdot (\vec{a}_{maxDim} + 0.5 \cdot \vec{a}_{middleDim}), \\ \vec{c}_3 &= \vec{p}_{center} + 0.5 \cdot (\vec{a}_{maxDim} - 0.5 \cdot \vec{a}_{middleDim}), \quad and \\ \vec{c}_4 &= \vec{p}_{center} - 0.5 \cdot (\vec{a}_{maxDim} - 0.5 \cdot \vec{a}_{middleDim}) \end{aligned} \tag{6.15}$$

result. Here, $\vec{a}_{maxDim}$ is the axis vector $\vec{a}_*$ with the maximal dimension dim_* and $\vec{a}_{middleDim}$ is the axis vector $\vec{a}_*$ with the middle dimension dim_* ($*$ stands for the dimensions x, y, and z depending on the dimensions of the axis). The axis $\vec{a}_*$ with the minimal dimension dim_* is ignored as it represents the thickness of the plane (cf. Figure 6.17). The thickness results from the threshold ϵ_{dist_thres} which was used in the function *distanceThresFilter()* at Algorithm 14 line 3.

Then, each identified plane is compared to the rest of the planes which are stored in the list $\vec{o}$. New planes are added, while existing planes are fused together. Finally, function *identifyObjects()* returns the list of objects $\vec{o}$.

The second sub-function, of the function *separateObject()* (see Algorithm 15 line 4), is called *analyzeObjectType()* and illustrated in Algorithm 16. It uses three steps (*meanIntensFactorCheck()*, *transformationCheck()*, *intensVariationCheck()*) to identify the type of the object and a fourth step to evaluate the results (*evaluateResults()*).

Algorithm 16 3D-Pre-Filter: analyzeObjectType()

Input: $\vec{G}_5$, $\vec{V}_{\text{distinction}}$, $\vec{o}$:
$\vec{G}_5$ is the scan cloud tuple, $\vec{V}_{\text{distinction}}$, is a vector containing all variables for the object distinction, and $\vec{o}$ is the list of objects.

Output: $m_{\text{objectType}}$:
$m_{\text{objectType}}$ is vector containing the type of objects.

1: **function** IDENTIFYOBJECTTYPE()
2: $m_{\text{intensFact}} \leftarrow$ meanIntensFactorCheck($\vec{G}_5$, $\vec{V}_{\text{distinction}}$)
3: $m_{\text{transf}} \leftarrow$ transformationCheck($\vec{G}_5$, $\vec{V}_{\text{distinction}}$)
4: $m_{\text{intensVariation}} \leftarrow$ intensVariationCheck($\vec{G}_5$, $\vec{V}_{\text{distinction}}$) ▷ see Algorithm 17
5: $m_{\text{objectType}} \leftarrow$ evaluateResults($m_{\text{intensFact}}$, m_{transf}, $m_{\text{intensVariation}}$)
6: **end function**

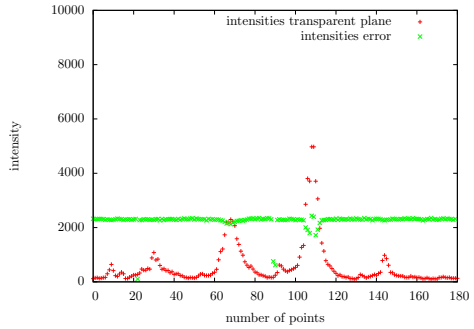
1^{st} step to analyse object type: determine mean intensity factor (*meanIntensFactorCheck()*):
As previously described in Section 6.1.3, the mean intensity factor $f_{material}$ stands for the relation of intensity values on the plane (m_{type} ="errorSurface") and intensity values behind the plane (m_{type} ="behindErrorS"). For transparent objects, most of the intensity values behind the plane are greater than the ones on the plane, cf. Figure 6.18a. For specular reflective objects

they are similar, cf. Figure 6.18b. Therefore, an arithmetic mean intensity $\hat{I}_{\text{“}behindErrorS\text{”}}$ and $\hat{I}_{\text{“}errorSurface\text{”}}$ of each group is calculated by

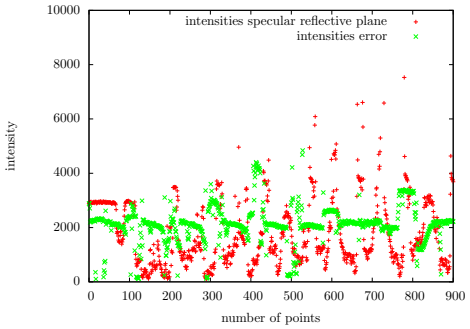
$$\hat{I}_* = \frac{\sum_{i=1}^{N_*} I_{*,i}}{N_*}. \tag{6.16}$$

Here, $I_{*,i}$ is the intensity value of each point i, N_* is the amount of intensity values, and * stands for “behindErrorS” and “errorSurface”. The factor $f_{material}$ is calculated by

$$f_{material} = \frac{\hat{I}_{\text{behindErrorS}}}{\hat{I}_{\text{errorSurface}}}. \tag{6.17}$$



(a) Intensity values of a transparent object: Green are intensity values of points behind the transparent plane (“behindTransparent”) and red are intensity values of points on the transparent plane (“transpSurface”).



(b) Intensity values of a specular reflective object: Green are intensity values of points behind the mirror plane (“behindReflective”) and red are intensity values of points on the mirror plane. For a better illustration of the differences between the intensity values on the plane (“reflectiveSurface”) and behind the plane (“behindReflective”), all values above 10.000 are cut. The maximum intensity value on the plane (“reflectiveSurface”) was 42.000.

Figure 6.18: Mean intensity factor of a transparent and specular reflective object.

Threshold $\epsilon_{intensFact}$ is used to rate the object as transparent or specular reflective. Finally, the function returns $m_{intensFact}$.

2^{nd} step to analyse object type: check for a valid transformation of back-projected points (*transformationCheck()*):
A reflective surface can be determined if the object behind has a symmetry w.r.t. the surface to an object in front of the surface. Therefore, the function *transformationCheck()* uses the points located behind the identified surface (“errorSurface”) and back-projects them, w.r.t. the identified plane. Now, the back-projected points $D = \{d_i | i = 1, \ldots, N_D\}$ and the points in front of the plane $M = \{d_j | j = 1, \ldots, N_M\}$ (“valid”) are matched by ICP.

In case of a positive match, cf. Figure 6.19, translation $\vec{Tr} = \{\Delta x, \Delta y, \Delta z\}$ and the rotation $\vec{R} = \{\Phi, \Theta, \Psi\}$ of the transformation matrix results $\vec{T} = (\vec{Tr}, \vec{R})$ with its angles and distances is compared to thresholds $\epsilon_{trans} = \{\epsilon_x, \epsilon_y, \epsilon_z, \epsilon_\Phi, \epsilon_\Theta, \epsilon_\Psi\}$. In case of small displacements it is assumed that the plane has reflective properties. In case of a huge displacement it is assumed that the ICP had matched the back-projected point cloud to an equal area. But in fact, this area was not causing the reflection. This could happen e.g. when a wall segment is matched to another wall segment. Finally, the function returns the result m_{Transf} and stores the Transformation matrix with the back-projected points.

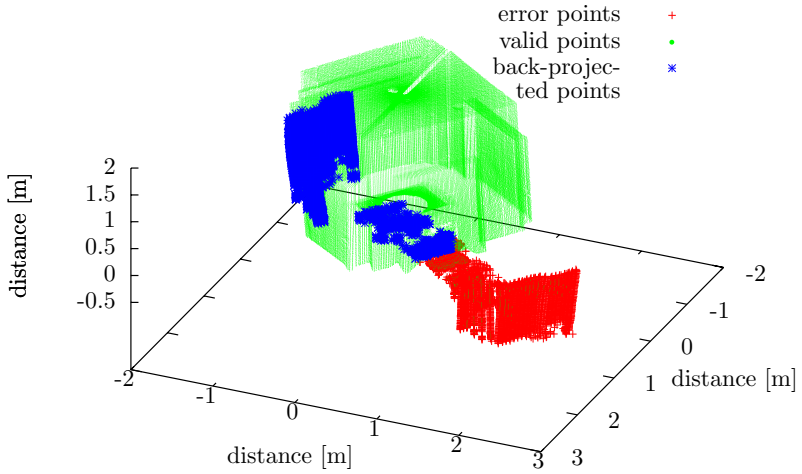
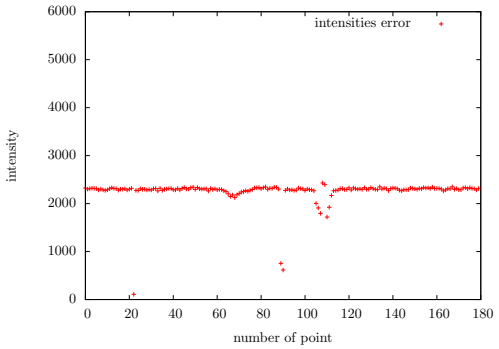


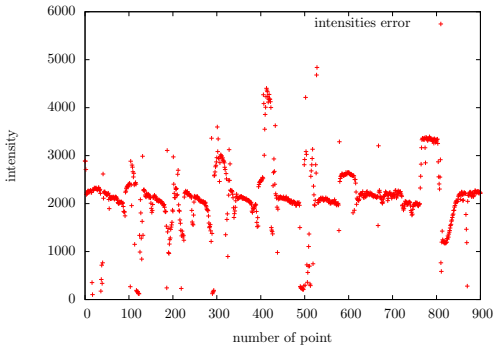
Figure 6.19: Result of back-projected and ICP fitted points (blue stars). The mirrored corner of the room (red crosses) is successfully back-transformed to its true position. It can be mapped there to improve the map. There are still "valid" points (green dots) in the area of the "behindErrorS" points. That is because the function *cleanScanTuple()*, see Algorithm 14, was not processed yet(red crosses).

It is understood that this method suffers from two drawbacks. First, locations with symmetry might result in a wrong identification. Second, it is assumed that the mirrored object is included in the "valid" point cloud as well. If only the mirrored object is seen, the function will not get an ICP result. Because of this, a verification based only on back-projection is not sufficient.

3^{rd} step to analyse object type: verification of consistent intensity values (*intensVariationCheck()*):
The intensity values of points behind transparent and specular reflective objects vary differently. Figure 6.20a illustrates the intensity values behind a transparent object which remain almost constant. In contrast, the intensity values behind a mirrored object vary strongly, as shown in Figure 6.20b. That is why the function *intensVariation()* analyses the behaviour of the intensity values of points located behind the surface ("behindErrorS").



(a) Intensity values of points behind a transparent object remain constant.



(b) Intensity values of points behind a specular reflective object vary strongly. For a better illustration intensity values above 6.000 were cut off. The maximum intensity value of a mirrored point was 13.000.

Figure 6.20: Variation of intensity values behind a transparent object vs. intensity values behind a specular reflective object.

To analyse the intensities the function calculates the median intensity $\tilde{I}_{median}$. Based on their size, it sorts the intensity values into I_{sort} and determines the median by:

$$f(\tilde{I}_{median}) = \begin{cases} I_{sort, \frac{N_{\text{“behindErrorS”}}+1}{2}} & \text{for } N_{\text{“behindErrorS”}} \text{ is uneven} \\ \frac{1}{2}(I_{sort, \frac{N_{\text{“behindErrorS”}}}{2}} + I_{sort, \frac{N_{\text{“behindErrorS”}}+1}{2}}), & \text{for } N_{\text{“behindErrorS”}} \text{ is even} \end{cases} \tag{6.18}$$

Here, $N_{\text{“}behindErrorS\text{''}}$ is the amount of points behind the surface.

Further, it identifies the object type based on the amount of intensity values n_{close} which are close to the median intensity $\tilde{I}_{median}$. It uses the threshold factor $\epsilon_{thres\_similar}$ to identify if the point is close, as illustrated in Algorithm 17.

If less than ϵ_{thres_vary} percent of the points are close to the median, the surface is rated to be a reflective and vice versa. In the end, the function returns the object type $m_{intensVariation}$.

Algorithm 17 intensVariationCheck()

Input: $\vec{G}_5$, $\vec{V}_{\text{distinction}}$:
$\vec{G}_5$ is the scan cloud tuple, $\vec{V}_{\text{distinction}}$, is a vector containing all variables for the object distinction.
Output: $m_{intensVariation}$:
$m_{intensVariation}$ is the resulting object type after the intensity variation check.

1: **for** $(i = 0; i < N_{\text{“behindErrorS”}}; i$++$)$ **do**
2: **if** $(|I_i - \tilde{I}_{median}| < \epsilon_{thres\_similar} \cdot \tilde{I}_{median})$ **then**
3: n_{close}++;
4: **end if**
5: **end for**
6: **if** $((\epsilon_{thres\_vary}) < (n_{close}/N_{\text{“behindErrorS”}}))$ **then**
7: $m_{intensVariation}$ = “reflectiveSurface”
8: **else**
9: $m_{intensVariation}$ = “transpSurface”
10: **end if**
11: **return** $m_{intensVariation}$

4^{th} **step: Evaluate results:**
In the final Step to identify the object type, the function *evaluateResults()* determines the final type of the object according to the greatest probability of $m_{intensFact}, m_{transf}$, and $m_{intensVariation}$ as

$$f(m_{objectType}) = \begin{cases} m_{objectType} \leftarrow \text{“reflectiveSurface”}, & \text{if} \quad \frac{\sum m_*}{3} < 2 \\ m_{objectType} \leftarrow \text{“transpSurface”} & \text{if} \quad \frac{\sum m_*}{3} \geq 2 \end{cases} \tag{6.19}$$

Here, $*$ stands for $intensFact$, $transf$, and $intensVariation$. The result is returned to the object list $\vec{m}_{object}$ in Algorithm 15 line 4.

As soon as there are not enough points left ($n_{affected} < \epsilon_{affected\_min}$), the list of object types $\vec{m}_{objectType}$ is added to the list of objects $\vec{o}$. The list is returned to Algorithm 14 by the function *separateObject()*.

D) Filter point cloud (highlighted in olive):

Now, the 3D-Mirror-Pre-Filter (see Algorithm 14), continues to clean scan tuple $\vec{G}_5$. Function *cleanScanTuple()* creates a vision cone for each object $\vec{o}$ and masks all points. Figure 6.21 illustrates the vision cone for one object. Blue points are located on the object surface and masked as “transpSurface ” or “reflectiveSurface” – based on type of the object. Points marked in red are located behind the object and therefore masked as “behindTransparent” or “behindReflective” – based on the type of the object. All other points are masked as “valid”. The resulting point cloud is stored into G_{out}.

The thickness d_{plane} results from threshold ϵ_{dist_thres} which was described during the step “object recognition - identify object”.

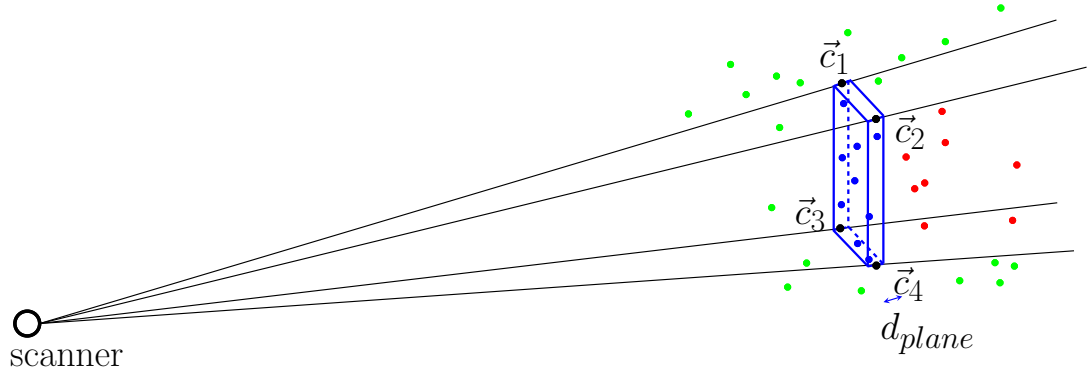


Figure 6.21: Vision cone to mask points according to their belonging. Blue points are located on the object surface and masked as "transpSurface " or "reflectiveSurface" based on type of the object. Points marked in red are located behind the object and therefore masked as "behindTransparent" or "behindReflective" based on the type of the object. All other points are masked as "valid".

E) Publish point cloud (highlighted in green):

In the last step, the 3D-Mirror-Pre-Filter publishes the masked point cloud $\vec{G}_{out}$ to the Mapping-module and the Post-Filter-module. It uses six messages according to the type of the object ("errorSurface", "reflectiveSurface", "transpSurface", "behindErrorS", "behindReflective", and "behindTransparent"). Also, a message with all "valid" points is broadcast. This message also includes points on the surfaces of transparent and specular reflective objects, as well as points behind transparent surfaces. These points represent measurements from real obstacles and therefore should remain. Points behind reflective objects result from reflections and therefore are being erased.

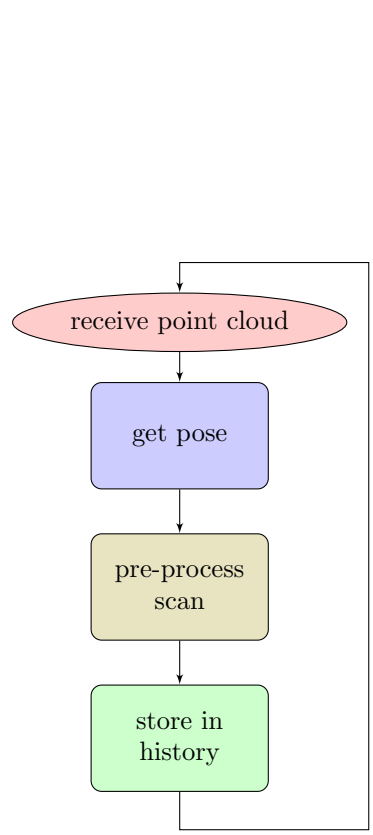
6.2.3 Code Description 3D-Post-Filter

The 3D-Post-Filter operates similar to the 2D-Post-Filter. This can be seen in the flow chat in Figure 6.22. The difference lies in the structure of the incoming data. In compare to the 2D-Post-Filter it does not receive a 2D-laser scan, but a 3D-point cloud.

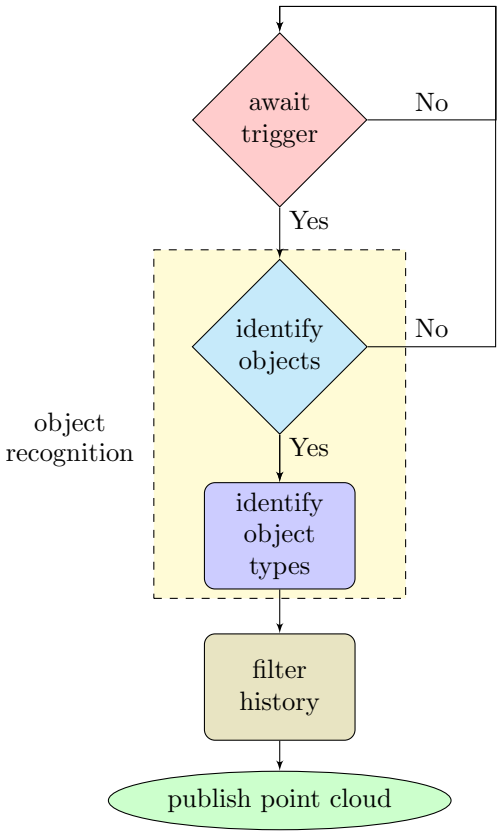
The Post-Filter is split into two independent processed chains: a storage chain and a filter chain. The "storage chain" buffers the entire point cloud history and the corresponding pose of the robot. In contrast, the "filter chain" awaits a trigger signal, e.g. from the Loop-Closure-module, to start processing the history. After a signal was received, the entire history is applied to identify and classify transparent and specular reflective objects. Here, the Post-Filter differs from the Pre-Filter as it considers the information of all point clouds. The Pre-Filter only observes the current point cloud delivered from the scanner. That is why the Post-Filter results in refined information of the objects. Hence, its influences on the laser scanner measurements can be determined more precisely.

- **“Storage chain” of the 3D-Post-Filter:**

Figure 6.22a illustrates the “storage chain” which is subdivided into four steps: receive point cloud, get pose, pre-process scan, store in history. The steps are coloured according to the flow chart in Algorithm 18 and processed one after the other.



(a) Flow chart of the 3D-Post-Filter Chain 1: storage chain.



(b) Flow chart of the 3D-Post-Filter Chain 2: filter chain.

Figure 6.22: Flow chart of 3D-Pre-Filter-node with its storage and filter chain.

Algorithm 18 3D-Post-Filter - "storage chain"

Input: S:
S includes the point cloud of Echo 1 and Echo 2, the corresponding intensities, angles, and the object mask according to the assignment of the Pre-Filter.
Output: G_H is the history of the stored pre-processed scans.

1: **procedure** STORAGE CHAIN
2: $G_1 \leftarrow$ receiveMaskCloud(S)
3: $\vec{P} \leftarrow$ requestTf($timestamp_{Scan}$)
4: $G_2 \leftarrow$ moveInWorldCoordinateSystem(G_1, P_R)
5: $A_{1,\{\text{"errorSurface"},\text{"behindErrorS"}\}} \leftarrow$ extractCorrupted(G_2)
▷ points which are on an object surface or corrupted by the object
6: $G_H \leftarrow$ storeHistories(G_2, A_1, P_R)
7: **end procedure**

A) Receive scan (highlighted in red):

The "storage chain" of the Post-Filter awaits a point cloud

$$S_1 = \{d_{1,i}, d_{2,i}, \vec{c}_1, \vec{c}_2, \vec{n}_{0,1}, \vec{n}_{0,2}, \alpha_i, I_{1,i}, I_{2,i}, m_{1,i}, m_{2,i} \,|\, i = 1, \ldots, N_{cloud}\} \tag{6.20}$$

from the Pre-Filter. Here, $d_{*,i}$ is the distance of the i^{th} scan point, c_* is the corresponding coordinates, $n_{0,*}$ is the corresponding normal vector, α_i is the corresponding scanning angle, $I_{*,i}$ the intensity of this point, and $m_{*,i}$ is the object type mask assigned from the Pre-Filter. The place holder $*$ stands for Echo 1 and Echo 2 and N_{cloud} is the amount of scan points.

B) Get pose (highlighted in blue):

In the next step the Post-Filter requests the corresponding pose $\vec{P}_R = (\vec{Tr}, \vec{R})$ with the position $\vec{Tr} = (x, y, z)$ and orientation $\vec{R} = (\Phi, \Theta, \Psi)$, to this point cloud. This is provided by the *ROS-TF-node* which manages the transformations supported by the SLAM-node, any other Localization-node, IMU, GPS, etc..

C) Pre-process scan (highlighted in olive):

In the third step, the scan is converted from the RKS into the WKS. Hence, point cloud S_2 results. Points located on transparent or specular reflective surfaces (mask value: "reflectiveSurface" and "transpSurface") and behind them (mask value: "behindReflective" and "behindTransparent") are stored into separate tuples $A_{1,\{\text{"}errorSurface\text{"},\text{"}behindErrorS\text{"}\}}$. The Post-Filter classifies each object without any previous knowledge. Since it considers multiple scans, this is more accurate and previous misassignments are eliminated.

D) Store in history (highlighted in green):

Next, in the final step of the "storage chain", the point cloud S_2, the extracted points $A_{1,\{\text{"}errorSurface\text{"},\text{"}behindErrorS\text{"}\}}$, as well as the corresponding pose $\vec{P}_R$ are stored into the tuple G_H.

- **"Filter chain" of the 3D-Post-Filter:**

The "filter chain" which is illustrated in Figure 6.22 operates similar to the 3D-Pre-Filter. It consists of four steps: await trigger, object recognition, filter points cloud, and publish point cloud. Algorithm 19 is coloured according to these steps.

In compare to the Pre-Filter, the point clouds do not require the "clean-up point cloud"-step as it has been previously proceeded by the Pre-Filter. Further, it does not receive point clouds, but awaits a trigger signal to work through the stored history G_H. As soon as the signal appears, the entire history is employed to identify and classify transparent and specular reflective objects. Then, it publishes the refined clouds, as well as messages containing the objects ("reflectiveSurface" and "transpSurface") and the affected points ("behindReflective" and "behindTransparent"). These messages can be used to build a refined map. They can also be used to update a map as described in Section 6.1.5 and demonstrated in Section 7.3.

Algorithm 19 2D-Post-Filter - "filter chain"

Input: G_H:
G_H is history of the stored pre-processed scans.

1: **procedure** FILTER CHAIN
2: **if** externalTrigger() **then** ▷ e.g. external loop-closure detection
3: **for** $(j = 0, j < L, j$++$)$ **do**
4: $G_2 \leftarrow$ outlierFilter(G_H, "errorSurface")
5: $\vec{o} \leftarrow$ seperateObject(G_2, $\vec{V}_{distinction}$, n_{object})
6: $G_{out} \leftarrow$ cleanScanTuple(G_2, $\vec{o}$, ϵ_{thres_plane}, $\epsilon_{thres\_visionCone}$)
7: sendFilteredScans(G_{out})
8: **end for**
9: **end if**
10: **end procedure**

A) Await trigger (highlighted in red):

To start the "filter-chain", the Post-Filter lurks for an external trigger signal in the first step. This can be supported by a Loop-Closure-node, by a Bypassing-node, or manually.

B) Object recognition(highlighted in yellow / black):

After the signal occurred, the object detection and identification starts which is highlighted in yellow on the flow chart and in black in Algorithm 19 (for better readability). The algorithm searches the entire history bank $A_{1,\{"errorSurface","behindErrorS"\}}$ for transparent and specular reflective objects. Since all points are converted into the WKS it is possible to fuse them. The determined planes result from a greater amount of points. That is why they are more accurate in compare to the determined planes of the Pre-Filter.

Now these objects are classified according to the sub-steps described at the Pre-Filter at Algorithm 16 in Section 6.2.1. This process also achieves refined information as there are more points located on the surface and behind the surface. It results in a list of classified objects $\vec{o}$.

C) Filter history (highlighted in olive):

In the third step, the entire history is cleaned according to the classified surfaces. Each point cloud in G_2 is searched if points are located behind an object $\vec{o}$ or on its surface. To do so, a

vision cone (cf. Figure 6.21) is spanned up and points are remasked according to their location and object type ("validPoint", "reflectiveSurface", "transpSurface", "behindReflective", and "behindTransparent").

D) Publish scans (highlighted in green):

Finally, these points in G_{out} are published. Five different messages are available for further usage:

- cleandScan: This message contains all points with their object mask. They can be used to build a refined map when using two mapping stages.
- points2Adapt: This message includes all points which are located on a transparent or specular reflective object as well as points located behind them. This message is intended to update a map according to the assigned mask of the point.
- surfacePoints: This message carries points which are located on a transparent or specular reflective surface. They result from real objects and therefore immediately can be added by a mapping module.
- behindReflective: Points behind a reflective surface are points which are mirrored. Their real location is in front of the surface. This message contains the points which need to be eliminated in the existing map. Further, they can be back-projected and included on their real location.
- behindTransparent: The points are located behind a reflective surface. Therefore, they can be inserted into the map.

6.2.4 Description Loop-Closure-module

The Loop-Closure-algorithm presented at Section 6.1.4 applies already a 3D-representation $\vec{P}_R = (\vec{Tr}, \vec{R})$ with its position $\vec{Tr} = (x, y, z)$ and orientation $\vec{R} = (\Phi, \Theta, \Psi)$. It is possible to use the same ROS-node to operate it together with the 3D-Mirror-Identifier-Approach.

The difference to the 2D case is due to the fact that the z–axis will not remain zero. Except when the robot drives through an environment with an even ground.

6.2.5 Description Localization-module

The applied mapping module described in the next subsection does not support localization. Hence, the pose $\vec{P}_R$ needs to be supplied by an additional node. The HECTOR-SLAM-node which was already applied for the 2D experiments supports this localization. It should be noted that this is only a 2D localization because the required sensors (e.g. IMU) for the 3D-pose estimation are not available. Because of the fact that the robot operates in even terrain, this is sufficient.

To provide a reliable pose, the HECTOR-mapping-node receives a laser scan message S from an additional 2D laser scanner. This can also be supplied by a cut through the 3D point cloud which is in parallel to the ground. Because of the higher availability of scan data with an additional 2D scanner this method was chosen. The HECTOR-SLAM-node builds up a 2D map based on an occupancy grid (see Section A.7.2). As a result of the matching which is achieved by merging multiple sup-grid resolutions, the robot pose is available. This pose is then broadcast to the TF-node which manages the poses. Now the mapping node as well as the Post-Filter request these poses.

6.2.6 Description Mapping-module

3D-Mapping is achieved by OctoMap-Mapping-node. It requests the robot pose from the TF-node as it does not include a localization. OctoMap is a simple mapping node, but not a SLAM node. Furthermore, it does not support post-manipulation of the map as it was achieved by the customized TSD-SLAM in Section 6.1.5.

Nevertheless, the first mapping node (cf. Figure 6.13) subscribes the messages from the Pre-Filter, while the second mapping node subscribes the messages from the 3D-Post-Filter. Two individual maps are built which are not fused. The first map still contains influences from transparent or specular reflective objects which are not identified in the first step. In compare, the second map, the refined map, does not contain such influences. This is due the fact that the object surface was modeled precisely by considering the entire history of scans.

Chapter 7

Experiments and Results

This chapter describes the experiments which are applied to: identify parameters of transparent and specular reflective objects; evaluate the Reflection-Identification-Approach in 2D and 3D; prove the reliability and assets of an object classification; and demonstrate the potential of the customized TSD-SLAM. Each section describes the experimental setup, explains the purpose of the experiment, and discusses the results.

All experiments are carried out with an Hokuyo UTM-30LX-EW multi-echo laser scanner. Its specification as well as the applied parameters for the modules are listed in Section A.

7.1 Static Scene Experiment to Identify the Parameters of the Reflection Model of Different Surfaces

The first experiment was conducted to develop intensity models to identify and distinguish between transparent and specular reflective objects. Therefore, the parameters of the reflection model for different surfaces are determined. Figure 7.1c illustrates the samples which consist of diffuse reflective (blue paper, red paper, white paper, green paper, and yellow paper), specular reflective (mirror and shiny aluminium), and transparent objects(glass and transparent plastic).

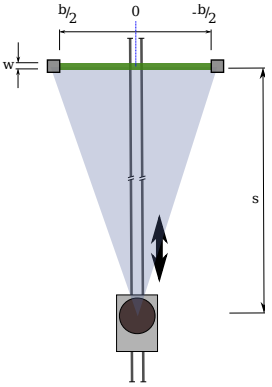
Figure 7.1a illustrates the schematic for the static scene experiment. The samples ($b_{object} = 52\ cm$) as well as the laser scanner are mounted on a track as shown in Figure 7.1b. The distance $s = [0.5; 6.5]\ m$ of the sample was extended stepwise by $0.5\ m$. For each data take only the section of the sample was measured. The angle around the perpendicular incident angle of the laser beam can be determined by

$$\alpha(i) = 0.25° \cdot i - \frac{0.25° \cdot (i_{max} - 1)}{2} \tag{7.1}$$

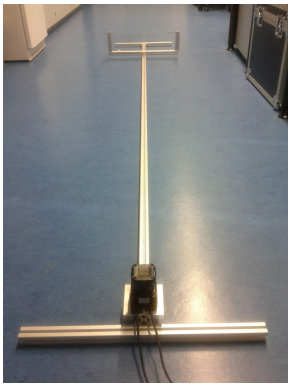
where i_{max} is the amount of measurement points on the sample of the current step (position of the sample) and i is the number of the scan points ($i \in 0, \ldots, i_{max}$). Since the angular resolution of the scanner is constant (0.25°), there are less measurement points for samples farther away. For each step, the arithmetic mean intensity $\hat{I}_{\alpha(i)}$ of each angle $\alpha(i)$ is calculated over $N = 5000$ scans:

$$\hat{I}_{\alpha(i)} = \frac{\sum_{n=0}^{N} I_{i,n}}{N} \tag{7.2}$$

Here, $I_{i,n}$ is the intensity value at the angle $\alpha(i)$ of the n^{th} scan.



(a) Experimental setup schematic for static scene.



(b) Hokuyo with varying sample mount for experiment with static scene.

(c) Photo of samples for experiment of static scene.

Figure 7.1: Experimental setup with sampels to identify the parameters of the reflection model for different surfaces (blue paper, red paper, white paper, green paper, yellow paper, mirror, shiny aluminium, glass, and transparent plastic).

In contrast, systematic errors caused by roughness, dirt and mechanical misalignments remain. This can be noticed in Figure 7.3b. Since the sample was not aligned perfectly with the laser scanner, the peak value of each step is shifted slightly around the perpendicular incident angle ($\alpha = 0°$).

Figures 7.3a – 7.3d illustrate four samples: white paper, aluminium, glass (thickness of 6 mm), and a mirror. For white paper (cf. Figure 7.3a), a diffuse reflective surface, the intensity curve shows a minimal dependency on the incident angle $\alpha(i)$. In compare, the curve of aluminium (cf. Figure 7.3b) shows a clear peak on a perpendicular incident angle. Also the curve of the mirror (cf. Figure 7.3d) shows a similar behaviour as the curve of aluminium, but it contains more disturbances. Besides, the maximum intensity values differ.

The curve can be described by the Phong-reflection model [Phong, 1975] where the measured intensity comprises ambient, diffuse, and specular properties:

$$I_{measured} = I_{ambient} + I_{diffuse} + I_{specular} \tag{7.3}$$

Neglecting the ambient and the diffuse reflection, the intensity can be modeled by

$$I_{specular} = I_{in} \cdot k_{specular} \cdot \cos^{n}\alpha \tag{7.4}$$

with I_{in} defines the intensity arriving at the surface, $k_{specular}$ is an empirically determined reflection factor, and α is the angle between the surface and the laser beam. For diffuse surfaces n remain small, higher values belong to specular reflective surfaces, and $n = \infty$ defines an ideal mirror, cf. Figure 7.2.

For glass (cf. Figure 7.3c) the intensity curve is hardly recognisable. The reason lies in the reflective and refractive property of glass. Nevertheless, for transparent and specular reflective objects a strong angle dependency is visible. That is why an object needs to be seen from the right position. Only if the required angle range was investigated, the surface can be mapped. This can be assured if an object was bypassed completely. With the assumption that this is achieved after the robot completed a loop, a Loop-Closure-algorithm was implemented. Nevertheless, a bypass-algorithm which is specialized to lurk for a passed transparent or specular reflective object would be more elegant.

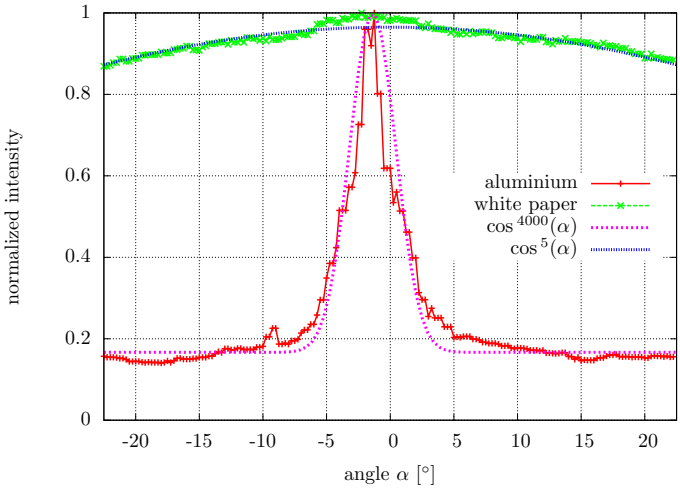
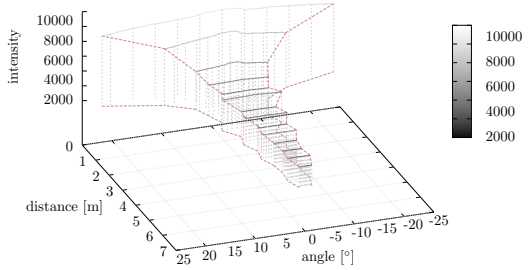
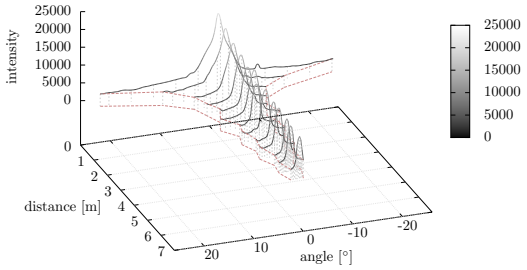


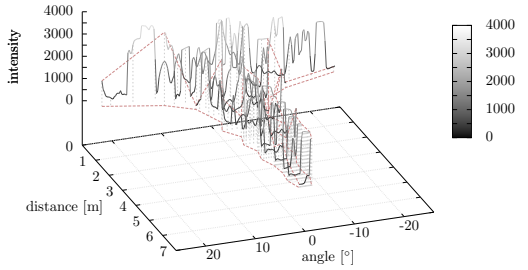
Figure 7.2: Fitting $\cos^n \alpha$ to white paper and aluminium at $d = 0.5\ m$.



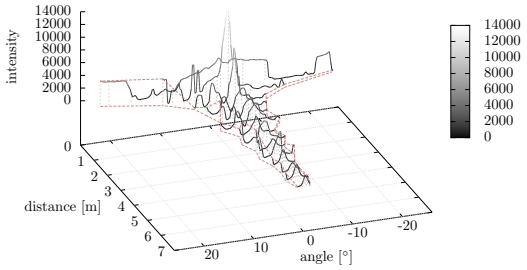
(a) Intensity curve results from white paper in dependency of distance and incident angle of the laser beam.



(b) Intensity curve results from aluminium in dependency of distance and incident angle of the laser beam.



(c) Intensity curve results from glass in dependency of distance and incident angle of the laser beam.



(d) Intensity curve results from a mirror in dependency of distance and incident angle of the laser beam.

Figure 7.3: Intensity curves result from white paper, aluminium, glass, and a mirror in dependency of the measurement distance and the incident angle of the laser beam.

Figure 7.4 illustrates intensity curves for surfaces in dependency on their distance. The dependency of the curve can be described by

$$I_{received} \approx \frac{I_{send}}{d^2}. \tag{7.5}$$

This confirms the research of Tatoglu and Pochiraju [2012] as well as the research of **?**. It verifies that the assumption of Wang and J. [2017] is wrong which claimed that there is no distance dependency of the intensity.

The experiment was conducted to verify the intensity dependency related to measurement distance and incident angle of a multi-echo laser scanner (Hokuyo UTM-30LX-EW). The results for transparent plastic and glass are similar. Also, the results for aluminium and the mirror are alike. However, the intensity curves of transparent plastic as well as of mirror are more noisy than the curves of glass and aluminium. This is caused by the surface and the density. Transparent plastic is not as clear as glass. Thus, this causes more variation in the curve. For the mirror this can be explained by the setup. A mirror consists of a glass plate with a shiny foil attached on the back. Two effects take place: refraction (glass) and reflection (shiny foil) Aluminium does not show both effects. However, a mirror should be rated as a reflective object and the transparent properties can be neglected.

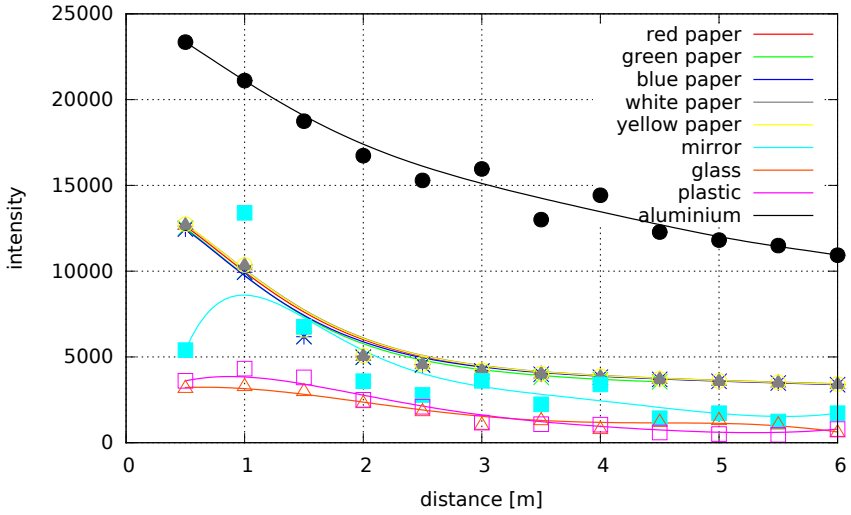


Figure 7.4: Intensity curves result from different surfaces in dependency of measurement distance. It is assumed that one value (mirror at $0.5m$) results from a wrong measurement caused by surface contamination.

The experiment confirms that it is necessary to bypass the entire object to see it at least once from the right perspective. The approach presented in this work was designed with a 2D-Pre-Filter, a 2D-Post-Filter, and a Loop-Closure-module. Further, it verifies that the intensity curves of transparent and specular reflective objects differ from each other. Because of that, a classification is feasible.

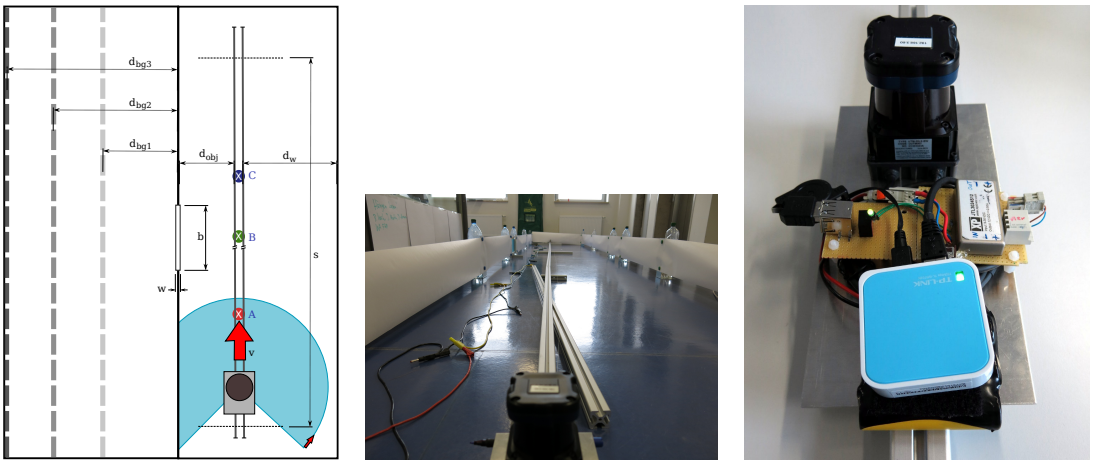
7.2 Drive by Experiment to Verify Behaviour of Intensities

This experiment was conducted to verify the intensity values regarding the incident angle and its echoes. In addition, models to classify transparent and specular reflective objects should be determined.

To do so, the laser scanner was mounted on a track which was aligned parallel in two aisles of white paper. Hence, the diffuse reflection is assured. One aisle consists of a sample (glass, transparent plastic, aluminium, mirror) which was bypassed by the scanner (start was at Pos. A and end at Pos. C in Figure 7.5a) with a constant velocity of

$$v_{laser} = \frac{0.278\ m}{77.36\ s} = 0.0036\ {}^{m}/_{s}. \tag{7.6}$$

For transparent samples (glass and transparent plastic) another white aisle was located behind the sample with a distance d_{bg} as illustrated in Figure 7.5a. The distance was changed during the experiment to investigate the influence of the background and on the intensity when scanning a transparent object (scenario 1: $d_{bg} = 50\ cm$, scenario 2: $d_{bg} = 110\ cm$, scenario 3: $d_{bg} = 160\ cm$). Figure 7.5b shows the experimental setup and Figure 7.5c the laser scanner equipped with a power supply and a wireless router on its aisle. The 2D-Post-Filter was modified so that only the points located on the surface are taken into account. They were saved into a separate file for evaluation.



(a) Experimental setup schematic for the drive-by scenario.

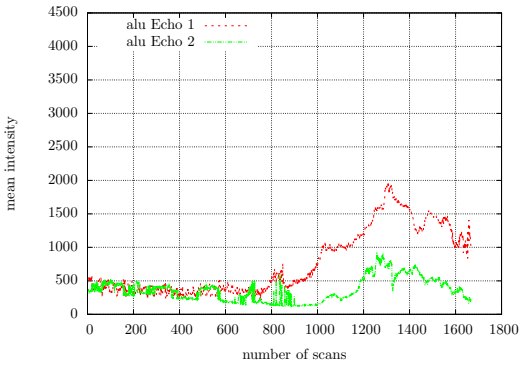
(b) Experimental setup for the drive-by scenario.

(c) Hardware setup of Hokuyo with indepent power supply and wireless connection to run it on the track.

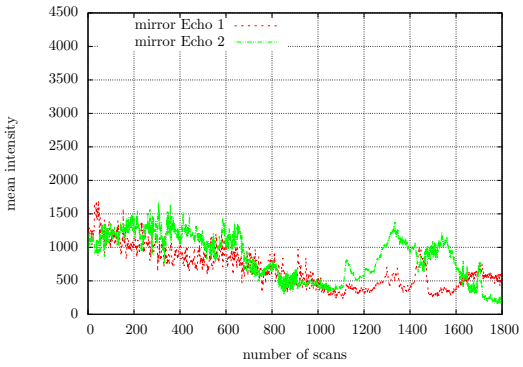
Figure 7.5: Experimental setup to verify the intensity values regarding the incident angle and its echoes in a drive-by scenario.

Figure 7.6a - Figure 7.6d illustrate the mean intensity (see Equation 6.6) of Echo 1 and Echo 2 for different scanner positions on the track. Echo 1 measurements result from the sample surface, while Echo 2 measurements result from the object behind (for a transparent sample) or from a mirrored object (for a reflective sample). For transparent objects, the mean intensity value of Echo 2 is greater than the mean intensity value of Echo 1. In contrast, for

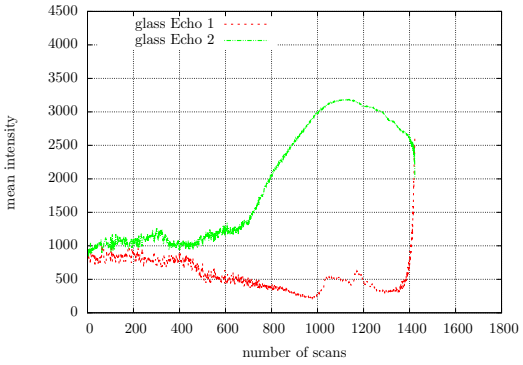
specular reflective objects the mean intensity value of Echo 2 is equal to the mean intensity value of Echo 1. At the maximum mean intensity value, the laser has reached a perpendicular incident angle to the surface center (Pos. B in Figure 7.5a). This corresponds to the results of Experiment 7.1.



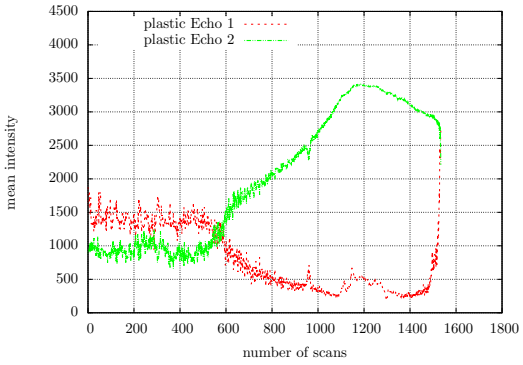
(a) Intensity curve results from aluminium in dependency of distance and incident angle of the laser beam.



(b) Intensity curve results from a mirror in dependency of distance and incident angle of the laser beam.



(c) Intensity curve results from glass in dependency of distance and incident angle of the laser beam.



(d) Intensity curve results from a transparent plastic in dependency of distance and incident angle of the laser beam.

Figure 7.6: Intensity curves result from aluminium, a mirror, glass, and a transparent plastic in dependency of the measurement distance and the incident angle of the laser beam.

Figure 7.7 illustrates the factor between the two mean intensity values $\hat{I}_1$ and $\hat{I}_2$. It changes with the location of the scanner for transparent objects and remains constant for specular reflective objects.

The experiment demonstrates that a tracking of the intensity values results in a better understanding of the object type and location. Objects can be identified by comparing the mean intensity value of Echo 1 and Echo 2. For transparent objects, measurements of Echo 2 have a higher intensity value than measurements of Echo 1.

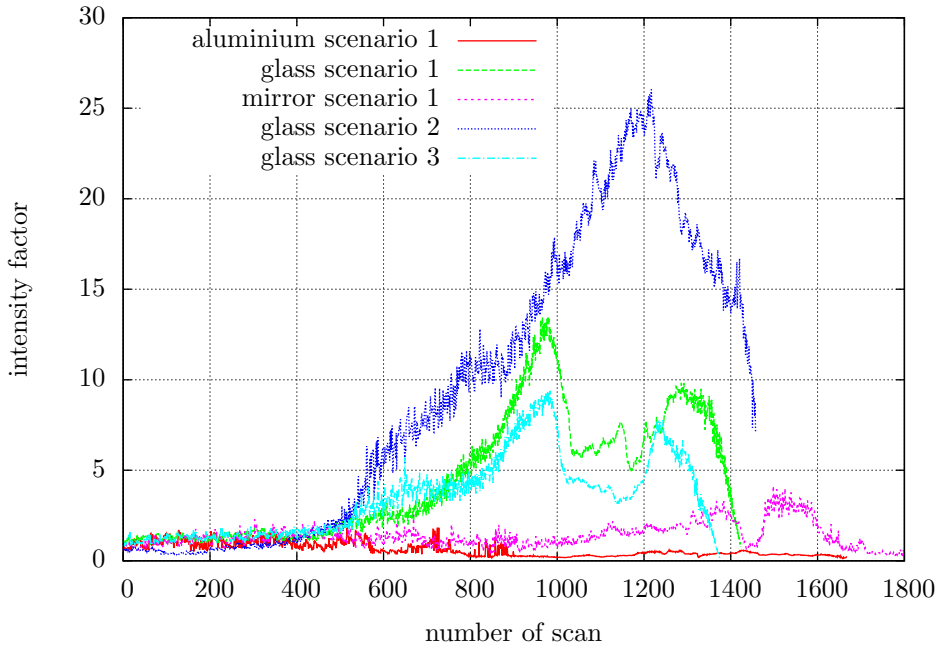


Figure 7.7: Intensity factor curve of various samples (aluminium, mirror, and glass). For glass, three distances of the background are illustrated (scenario 1: $d_{bg} = 50\ cm$, scenario 2: $d_{bg} = 110\ cm$, scenario 3: $d_{bg} = 160\ cm$).

7.3 SLAM Evaluation with 2D-Mirror-Detector-Approach

This experiment applies an Hokuyo UTM-30LX-ELW mounted on a Kobuki [Kobuki, 2017] (cf. Figure 7.8) to evaluate the 2D-Mirror-Detector-Approach, Version 1 without an object classification (cf. Figure 7.10). Further, three SLAM-approaches were applied to compare the resulting maps. To ensure equal test conditions, a rosbag-file was recorded.

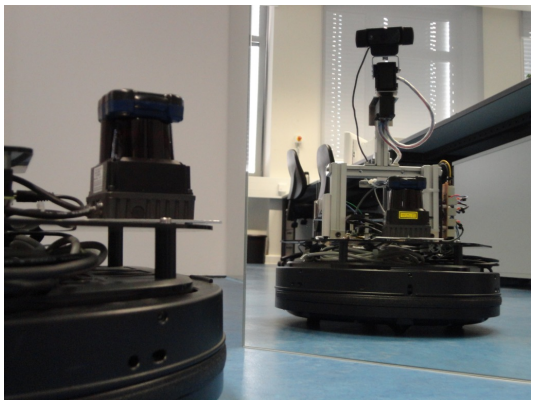


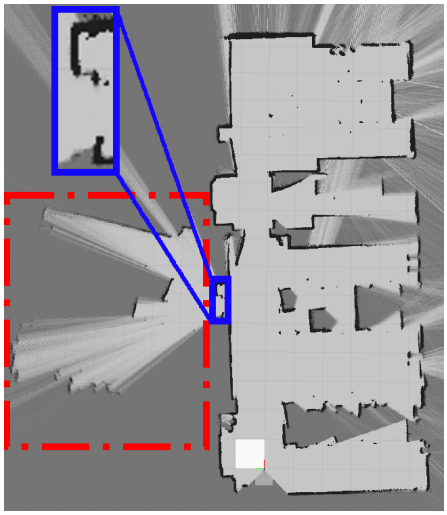
Figure 7.8: Kobuki equipped with an Hokuyo UTM-30LX-ELW for 2D mapping. It stands in front of a mirror and faces itself.

Figure 7.9 illustrates the test environment, an office room which had a frameless mirror (length $b = 52\ cm$). It is marked by a blue rectangle. The mirror was placed with a gap off the wall and aligned in an angle to it. This can be seen on the magnified map at Figure 7.10d. It demonstrates that no frame- or wall-detection is necessary to locate the object (free standing). The mirror was arranged so that the beam of the laser scanner (which measures parallel to the ground) hits the surface in the center. That is why the curvy shape of the mirror does not affect the measurements.

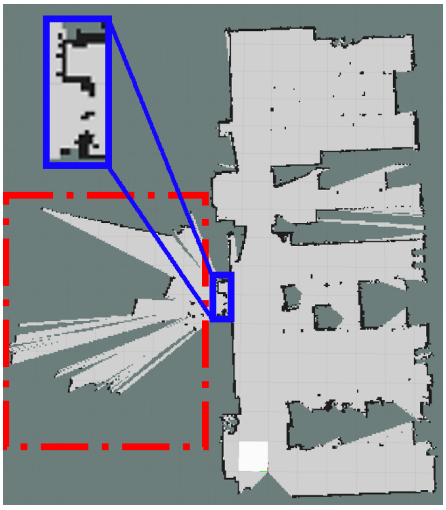
Figure 7.9: Frameless mirror (marked by a blue rectangle) placed next to the wall in an office room.

Figures 7.10a, 7.10b, and 7.10c illustrate the maps created with SLAM approaches using unfiltered scans (Echo 1 of the Hokuyo). Figure 7.10d illustrates the refined map of the applied 2D-Mirror-Detector-Approach with TSD-SLAM. The nodes were connected according to the processing chain in Figure 6.1.

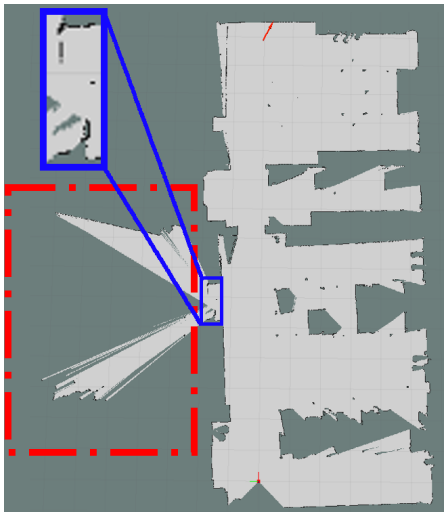
In Figure 7.10a – 7.10d the location of the mirror is marked by a blue rectangle and magnified to illustrate the mapped surface of the mirror. The identified end points of the mirror are marked by a turquoise dot in Figure 7.10d. The object size was determined to be $b = 50\ cm$ which is pretty close to the real size ($b = 52\ cm$). Only with the 2D-Mirror-Identifier-Approach the surface of the mirror is fully visible. Otherwise, the surface is not or only partly visible. This is due the fact that the reflective surface is only occasionally visible for the laser scanner. Hence, a dynamic mapping algorithm will overwrite the wall points (surface) if the wall is not seen anymore. The danger in not seeing the surface, but seeing points behind (marked by a red broken line), lies in the fact that the exploration node of the robot tries to navigate through the surface. In its point of view, there is an open gap and some area behind it which needs to be investigated. As a result, the robot crashes into the mirror. It is obvious that this can be prevented by applying the Approach. Furthermore, the distortions resulting from the mirror (marked by a red broken line) are completely eliminated by the 2D-Mirror-Detector-Approach (cf. Figure 7.10d). As illustrated in Figure 6.1 any mapping approach can be combined with the 2D-Mirror-Detector-Approach and the 2D-Mirror-Identifier-Approach. Therefore, no manipulation of the mapping approach is required as long as no map update should be proceeded. As a drawback, two separate maps are created and only the refined version assures a complete mapped surface and elimination of transparent and specular reflective influences.



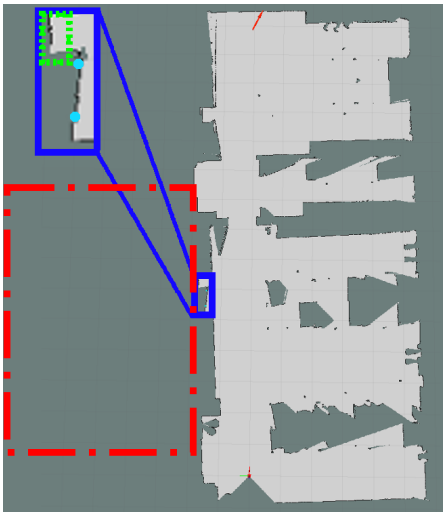
(a) Map of CRSM-SLAM created from Echo 1 of the laser scanner. It contains all influences (reflections) of the mirror. Besides, the surface of the mirror is not mapped.



(b) Map of HECTOR-SLAM created from Echo 1 of the laser scanner. It contains all influences (reflections) of the mirror. Besides, the surface of the mirror is not mapped.



(c) Map of TSD-SLAM created from Echo 1 of the laser scanner. It contains all influences (reflections) of the mirror. Besides, the surface of the mirror is not mapped.



(d) Map of TSD-SLAM with 2D-Mirror-Identifer-Approach. It contains no influences (reflections) of the mirror and the surface of the mirror is mapped.

Figure 7.10: Maps registered from the same dataset with different SLAM approaches in an environment containing a mirror. The mirror is marked by a blue solid line, reflections are marked by a red broken line. The visible wall segment behind the mirror is marked by a green dotted line in (d). It illustrates that the mirror was free standing.

7.4 Object Classification with 2D-Mirror-Identifier-Approach

This experiment considers two aspects: classification of influencing transparent and specular reflective objects as well as verification of a modified TSD-SLAM to automatically update the map. It is understood that it is necessary to know which object appears in the scan to correctly update the map. For a transparent object (e.g. glass), the surface as well as the object behind the surface should be mapped. It represents a solid object which cannot be passed by the robot. Hence, it also should be mapped. Since it is possible to see through glass, the object points behind represent a real object (e.g. a wall). For a reflective object (e.g. mirror) only the points on the surface should be mapped. The points behind illustrate a mirrored object which is located in front of the mirror. The real location of the mirrored object depends on the incident angle of the laser beam when scanning. These points need to be back-projected to their original location. After, they can be entered to the map.

The experiment was conducted with the same robot setup as the previous experiment in Section 7.3. This time the environment consists of a corridor and a laboratory room. The corridor contains a glass façade (cf. Figure 7.11a) and a 52 *cm* long mirror (marked by a blue rectangle in Figure 7.11a) was leaning on a cabinet in the laboratory.

(a) Picture of the corridor with glass surface.

(b) Picture of the laboratory with mirror which is marked by a blue solid rectangle.

Figure 7.11: Mapped environment with a glass surface and a mirror leaning on the cabinet.

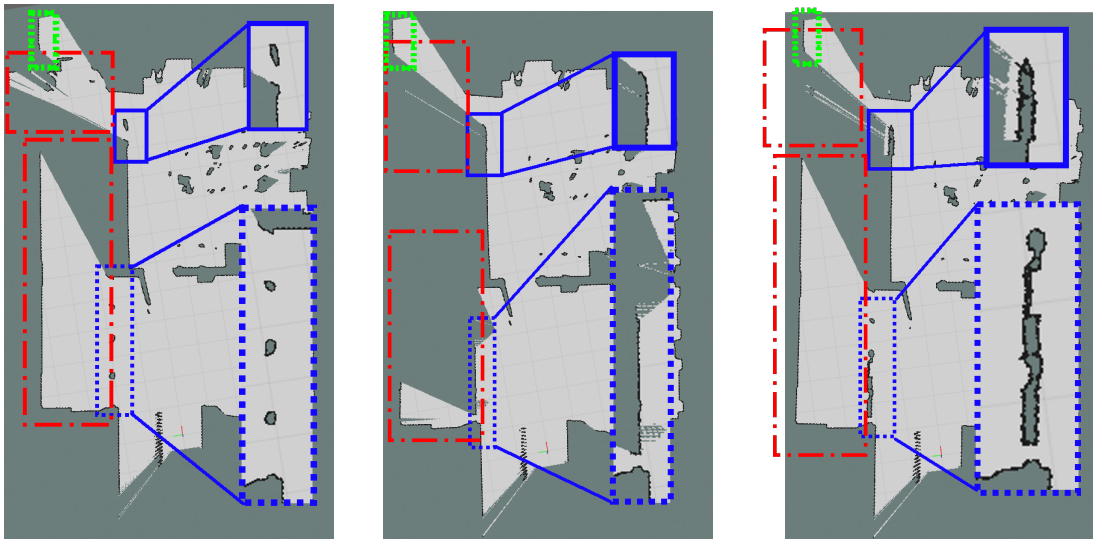
Figures 7.12a – 7.12b illustrate the resulting maps from the 2D-Mirror-Detector-Approach (V1) and Figure 7.12c from the 2D-Mirror-Identifier-Approach (V2 with classification).

The resulting map from the 2D-Pre-Filter (cf. Figure 7.12a) contains all points behind the glass façade (marked by a blue dotted line rectangle) as well as most of the points behind the mirror (marked by a blue solid line rectangle). In the magnification of the façade one can see that the surface is not mapped at all. Only the frame elements (gray structure in Figure 7.11a are visible. Also, the mirror is only partly mapped. This is due the fact that the TSD-SLAM creates a dynamic map. This eliminates objects which are occasionally visible or move. Due to the weighting of the TSD-function the objects clear away faster or slower. Since reflective objects are longer visible (due to a bigger visible angle range), some surface points of the mirror remain in the map.

In compare, the resulting map from the 2D-Post-Filter (cf. Figure 7.12b) contains the surfaces of the glass and the mirror. On the top corner of the glass façade some part of the surface is missing. The reason for this is that the surface was not seen from the required angle range as an open door to the laboratory blocked the laser beams. The objects behind the surfaces are erased by the 2D-Post-Filter. Because there is no distinction between transparent and specular reflective objects in V1 it is obvious to eliminate all points which are behind a surface. Only a small error remains at the lower part of the glass façade. This results from an imprecise defined surface end point. At the top left corner of the maps, a green dotted line rectangle highlights a wall segment. This is an existing wall which was seen through the open door at the time the robot arrived at the top end of the laboratory. In compare to the pre-filtered map, the wall segment is clearly visible and no erroneous measurements remain.

Figure 7.12c illustrates an updated map from the 2D-Mirror-Identifier-Approach. After the mirror was passed, a trigger signal was sent manually to start the 2D-Post-Filter-Stage. The 2D-Mirror-Detector-Approach was configured according to the processing chain in Figure 6.1 and the 2D-Mirror-Identifier-Approach according to the processing chain in Figure 6.2. As previously mentioned, this signal can be sent by an Loop-Closure-module as well. The updated map contains both surfaces, the mirror and the glass façade. This is similar to the resulting map of the 2D-Mirror-Detector-Approach, cf. Figure 7.12b. The difference can be seen in the magnifications of the surfaces. In Figure 7.12c, the updated map contains two walls next to each other. This results from the update of the TSD-function as described in detail in Section 6.1.5. A second difference between the two maps is due the fact that the updated map contains object points behind the glass surface. In compare, object points behind the mirror surface are erased. This is achieved by the object classification which is implemented in the 2D-Mirror-Identifier-Approach. Hence, the updated map (cf. Figure 7.12c) contains additional information in compare to the post-filtered map (cf. Figure 7.12b) – the existence of a corridor behind the glass façade. Thinking back to the scenario which was described in the introduction, a rescue team will know that there is an additional room which cannot be reached by the robot due to a closed glass front. This was not obvious on the map without an object classification (cf. Figure 7.12b).

The experiment outlines that a distinction between transparent and specular reflective objects is possible and required to improve the map. It demonstrates that the 2D-Mirror-Identifier-Approach successfully identifies multiple objects and distinguishes between transparent and specular reflective influences. It eliminates erroneous data behind specular reflective objects and maintains data behind transparent objects. Further, the experiment demonstrates that an updated map can be generated by modifying the SLAM-node. This simplifies usability and reduces memory requirements because only one map exists. Further, it allows navigation on the best current available map.



(a) Map created from the data of the 2D-Mirror-Detector-Approach 2D-Pre-Filter with TSD-SLAM.

(b) Map created from the data of the 2D-Mirror-Detector-Approach 2D-Post-Filter with TSD-SLAM.

(c) Updated map created from the 2D-Mirror-Identifer-Approach.

Figure 7.12: Maps created with the 2D-Mirror-Detector-Approach and the 2D-Mirror-Identifier-Approach. The glass area is marked by a blue dotted line rectangle. The mirror is marked by a blue solid line rectangle. The points behind the surfaces are marked by a red broken line rectangle. A wall segment which results from the corridor next to the two rooms is marked by a green dottet line rectangle.

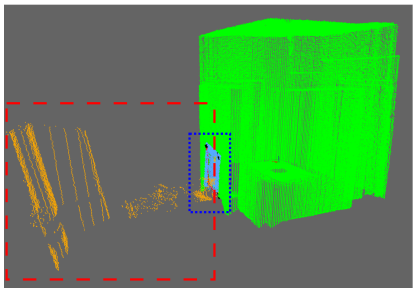
7.5 Object Classification with 3D-Reflection-Identifier

Aim of the first 3D-experiment was the applicability test of the 3D-Pre-Filter. It is applied to two environments to demonstrate that an identification as well as classification of transparent and specular reflective objects with a rotating Hokuyo UTM-30LX-EW laser scanner is possible. Figures 7.13a – 7.13f illustrate pictures of the chosen environments and the purified point clouds of the 3D-Pre-Filter module. Points which are rated as “valid” are coloured in green. Points which are located on the transparent or specular reflective surface are coloured in turquoise and marked by a blue rectangle. Likewise, the objects are marked by a blue rectangle in the environment as well. The points behind the detected object which are rated as “affected” are coloured in orange. The applied parameters for the 3D-Pre-Filter and the 3D-Hokuyo-node are listed in Section A.6.

Empty room with a mirror:
The first scene (cf. Figure 7.13a and Figure 7.13b) consists of an unframed mirror leaning on a door. Due to the angle between the scanner and the door, the mirror can be rated as free standing. The room was empty so that no other influences of transparent or specular reflective objects disturb the measurements.

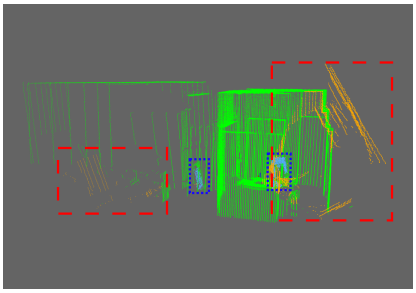
Table 7.1 shows the results of the 3D-Pre-Filter which classifies the object correctly to be reflective.

(a) Picture of scenery of an empty room containing a framless, free standing mirror. The mirror is marked by a blue rectangle.



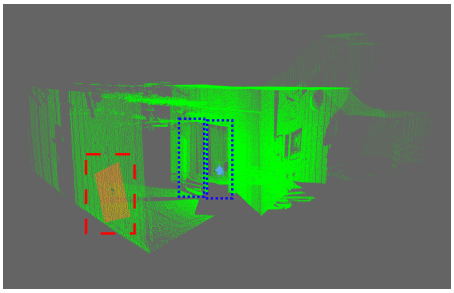
(b) Pre-filtered point cloud of the 3D-Reflection-Idenifier. Points located on the object are coloured in turqouise and marked by a blue rectangle. Points behind the mirror are coloured in orange and marked by a red broken line rectangle. The rest of the points are coloured in green.

(c) Picture of scenery of a room containing two framless, free standing mirrors. The mirrors are marked by blue rectangles.



(d) Pre-filtered point cloud of the 3D-Reflection-Idenifier. Points located on the objects are coloured in turqouise and marked by blue rectangles. Points behind the mirrors are coloured in orange and marked by red broken line rectangles. The rest of the points are coloured in green.

(e) Picture of scenery of a stairway containing a glass façade. The glass area is marked by a blue rectangle.



(f) Pre-filtered point cloud of the 3D-Reflection-Idenifier. Points located on the doors are coloured in turqouise and marked by a blue rectangle. Points behind the glass doors are coloured in orange and marked by a red broken line rectangle. The rest of the points are coloured in green.

Figure 7.13: Sceneries to verify the applicability of the 3D-Reflection-Idenifier-Approach. The first scenery contains a frameless, free standing mirror in an empty room (Figure 7.13a – Figure 7.13b). Then another mirror was added (Figure 7.13c – Figure 7.13d). The third scenery contains a glass façade in a stairway (Figure 7.13e – Figure 7.13f).

Description	Result
real size [cm]	60×40
measured size [cm]	58.4×36.1
$f_{material}$	1.78
meanIntensFactorCheck()	$\pm 20\%$
transformationCheck()	$\vec{T} = (4.8°, 0.4°, -0.4°, 0.01\ m, 0.01\ m, 0.10\ m)$
intensVariation()	38%
objectType()	"reflective"

Table 7.1: 3D-Pre-Filter results of an empty room with a mirror.

In addition, a second mirror was added to the scene (cf. Figure 7.13c). The result of the 3D-Pre-Filter is illustrated in Figure 7.13d. It demonstrates that multiple mirrors are also correctly identifiable.

Stairway with a glass façade:
The second scene (cf. Figure 7.13e and Figure 7.13f) consists of a stairway with two glass doors. Due to the strong dependency of the laser beams incident angle only a small area is detectable (see measured size in Table 7.2). The identified area is illustrated as turquoise points in Figure 7.13f and the resulting points behind the surface are coloured in orange. The detected area is around the laser beam which hits the surface perpendicularly. This point is located in front of the robot. This fact is due to the orientation of the glass door and the robot. Hence, the robot will not crash into the glass door, even the area is not completely detected. Nevertheless, it is necessary to fuse multiple scans (from different locations) to determine the entire glass area. Therefore, a 3D-Post-Filter-module is required. Regarding the object classification, the results of all three methods are correct and the glass door is rated to be a transparent object.

Description	Result
real size [cm]	2 doors á 88×198
measured size [cm]	24.2×20.9
$f_{material}$	0.00073
meanIntensFactorCheck()	$\pm 20\%$
transformationCheck()	$\vec{T} = (-15.0°, -13.7°, 19.4°, -0.86\ m, 0.62\ m, -0.07\ m)$.
intensVariation()	51%
objectType()	"transparent"

Table 7.2: 3D-Pre-Filter results of a stairway with a glass façade.

In summary, it can be said that multiple unframed transparent and specular reflective objects are detectable and classifiable by the 3D-Pre-Filter. In contrast to the 2D-Approach, the greater amount of measurements on the surface already allow a classification with the 3D-Pre-Filter-node. The stairway-scene points out that applying a 3D-Post-Filter is still advisable. This is due the fact that the bypassing of the object assures that a surface is seen completely from the required angle view.

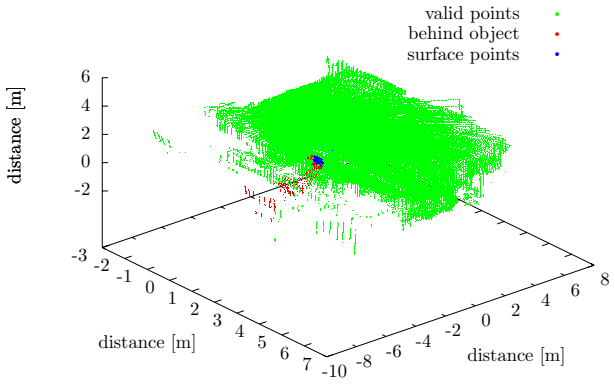
7.6 Mapping with 3D-Reflection-Identifier-Approach

The final experiment is conducted to demonstrate the reliability of the 3D-Reflection-Identifier-Approach. The 3D-Pre-Filter as well as the 3D-Post-Filter are connected to an individual mapping stage. Two maps are created: a preliminary map (Figure 7.14a) and a refined map (Figure 7.14b). It should be noted that it is possible to build an updated map as well. The applied mapping module needs to support a post-update. This was demonstrated exemplarily for the 2D-case at Section 7.4. The mapping module (OctoMap) which is applied in this experiment does not support any localization. Therefore, HECTOR-Mapping-node is utilized with a Sick Tim551-2050001 laser scanner to support the robot pose.

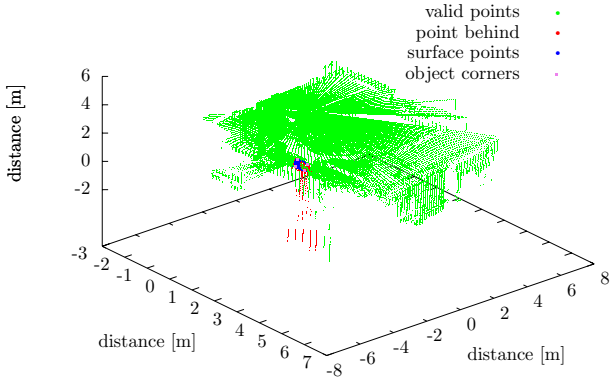
Figure 7.15a-7.15b illustrates the resulting pre-filtered point clouds, and Figure 7.14a illustrates the preliminary map. The reflective surface is only visible in point cloud 2 (Figure 7.15b). In compare to the pre-filtered point clouds, the refined map (Figure 7.14b) does contain almost no influences of the reflective object. Due to the post-processing the reflective surface can be recognised better. Further, point cloud 1 and point cloud 2 are filtered from influences. This result is similar to the result of the 2D-Post-Filter (see Section 7.3). Table 7.3 lists the detected mirror planes. For the Pre-Filter, the object dimensions of each scan including an object are listed. For the Post-Filter, only the final object size is listed. This confirms that a post-processing improves the detection and identification of transparent and specular reflective objects. The refined map of the Post-Filter is illustrated in Figure 7.14b.

Description	**Result Pre-Filter**	**Result Post-Filter**
real size of mirror [cm]	60×40	
mirror size at scan 1 [cm]	no detection	48.8×28.2
mirror size at scan 2 [cm]	28.9×24.7	
mirror size at scan 3 [cm]	no detection	

Table 7.3: Pre-Filter results vs. Post-Filter results of the 3D-Reflection-Identification-Approach.

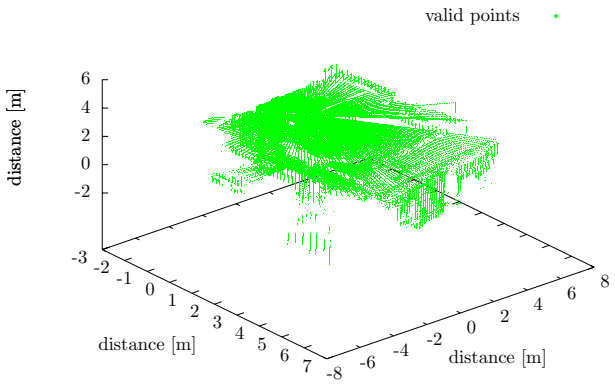


(a) Mapped point clouds after the 3D-Pre-Filter.

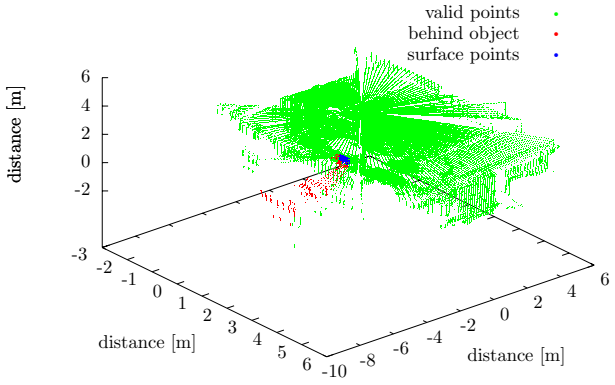


(b) Mapped point clouds after the 3D-Post-Filter.

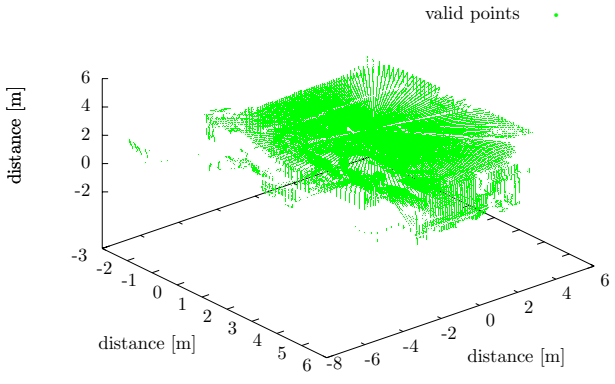
Figure 7.14: Mapped point clouds resulting from the Pre-Filter and the Post-Filter. Due to the Post-Filter, also influenced points in point cloud 2 and 3 are filtered out.



(a) Point cloud 1 after the 3D-Pre-Filter. No specular reflective influences are detected.



(b) Point cloud 2 after the 3D-Pre-Filter. A reflective surface has been detected.



(c) Point cloud 3 after the 3D-Pre-Filter. No specular reflective influences are detected.

Figure 7.15: Point clouds filtered by the 3D-Pre-Filter and refined map. The reflective object was only seen at point cloud 2.

Chapter 8

Summary and Outlook

Subject thesis investigates influences of transparent and specular reflective objects and potentials to improve laser scanner based mapping for mobile rescue robots. In regions devastated by disasters like earthquake, flooding, hurricane, etc. robots are a valuable support for rescue teams. The robots are made to inspect narrow holes, search for victims, and protect rescue teams as they are the first ones to enter dangerous areas. To master this challenge and support the rescue troops with reliable information, a reliable mapping is required. It is understood that in such a scenario it is not possible to customize the environment before inspection. Hence, erroneous measurements caused by transparent and specular reflective objects must be detected and eliminated during missions.

Bases for subject thesis were the three questions defined in Chapter 1:

- Is it possible to detect transparent and specular reflective objects and their influences solely with a laser scanner?
- How can a distinction be made between a transparent and a specular reflective object?
- If the influences can be recognised and characterised, is this possible on the fly, during the mission, or does it require a post-processing?

To answer these questions, as a first step, state-of-the-art approaches are reviewed. They are summarized in Table 5.1 which provides for an rough overview of their assets and drawbacks. It points out that

- in many cases manipulations of the environment (covering objects) are still required. This is particularly the case for stationary measuring systems.
- most approaches apply an additional sensor principle to detect transparent and specular reflective influences.
- only a few approaches are running on the fly.
- often, it is necessary to predefine the size and shape of the objects to be able to recognise them.
- up till now, no classification of transparent and specular reflective objects has been provided. Current algorithms are optimized to one or the other type of object.
- 3D mapping is rarely employed, even when required for navigation and manipulation in real world.

The review evinces that there is no approach available which is capable to satisfy all requirements: identification of influences, classification of objects, mapping surfaces of these objects, handle influences according to the object type, running on the fly, and applying only a laser scanner. It also covers the potential and the need of such an algorithm out.

The state of the art already confirms that it is possible to detect transparent and specular reflective objects and their influences solely with a laser scanner. However, the strong dependency of the laser beam on the incident angle w.r.t. the surface leads to various results. Based on the angle and the type of the surface, the measurements can produce the following results: from the surface, from an object behind the surface, or from an object in front of the surface (a mirrored object). As one cannot be sure that the beams always hit the surface within the required angle range, objects are occasionally visible. That is why a Pre-Filter and a Post-Filter stage is required, especially for 2D mapping. Experiments demonstrated that the Pre-Filter is capable to eliminate most influences. By applying a Post-Filter it is possible to improve the results. Hence, all influences are detected and eliminated. Further, the object surface is modeled more accurately. For 3D mapping, a scan results in more measurements. That is why it is easier to gain values from the required angle range. Nevertheless, the experiments demonstrate that a Post-Filter refines the results as well, especially for transparent objects. Reason being the fact that the angle range is smaller than the angle range of specular reflective objects.

This leads directly to the second question which deals with the distinction between transparent and specular reflective objects. The implemented Mirror-Identifier-Approach which is the core of this thesis, demonstrates that a distinction is possible. Objects are classified based on three independent methods: determine the mean intensity factor, check for a valid back-projection, analyse the intensity variation. In 3D, this is immediately possible with the Pre-Filter. In 2D, a Post-Filter is required as the amount of data is too small to achieve reliable results. The drawback of the post-processing can be overcome by customizing the SLAM node as proceeded in Experiment 7.4. Hence, the map is updated during mission.

This corresponds to the third question. For 3D, it is possible to recognise and distinguish between transparent and specular reflective objects on the fly. In contrast, in 2D a Pre- and Post-Filter is required, because of the fewer amount of measurements. As verified by the experiments, the Pre-Filter identifies most influences on the fly, but only the Post-Filter is capable to classify the objects. To avoid a separate mapping stage (pre- and post-mapping) a solution was presented in Section 6.1.5. The customized TSD-SLAM demonstrates that it is possible to rely on one map and update it as soon as precise data are available. To do so, surface points as well as points behind a transparent object are added and erroneous points behind a reflective object are deleted on demand.

To start the Post-Filter, a Loop-Closure-module was implemented. It is assumed that the mirror was passed after the robot returns to a previous location. Because of that, the object was seen at least once from the right perspective.

Summarizing, it is clear that an identification of transparent and specular reflective objects can be achieved solely with a laser scanner on the fly. In compare to standard mapping, surfaces of such objects are mapped completely. The points behind the surfaces are mapped based on the type of the object. In addition, a back-projection of mirrored points can be used for mapping as well. A post-processing leads to improved determination of the surface dimensions. Hence, also the detection of affected points is better. The applied surface model describes a planar, square object. Other shapes as well as uneven surfaces have not yet been considered. Nevertheless, the experiments demonstrate that a distinction between transparent and specular reflective objects is possible and reliable with the presented Reflection-Identification-Approach. Last but not least, multiple objects regardless of the size are detectable.

8.1 Future Work

It should be mentioned, that the experiments also emphasize some drawbacks and generate impetus for future work. They can be summarized as follows:

- implement models to respect other shapes of objects (round, multi corner, etc.)
- implement models to respect other surface structures (unplanar)
- detect immediately the bypass of an object
- optimize storage of the Post-Filter
- implement parallel processing

These are briefly addressed to define their potential and list the required changes.

The implemented Reflection-Identification-Approach assumes planar and square objects. Hence, the object model is a line in 2D and a square plane in 3D. It is understood that the model represents only a limited group of objects. For the 2D case, the shape is irrelevant as any shape refers to a line. Just the length needs to be variable which has already been respected. That is why round, square, as well as multi-corner objects are already detectable. This is different for the 3D case. Here, the implemented algorithm fits a square plane by adapting the length according to the outer points of the object. Fitting also other shapes will improve the precision of the 3D-Mirror-Identifier-Approach. By doing so, the vision cone (cf. Figure 6.21) needs to be modified as well. This is the case because the span up of the vision cone is based on the frame of the object.

Uneven objects should be handled differently. This is true for the 2D and 3D-case. On the one hand, it results in a precise mapped surface, on the other hand, round reflective objects show a different mirrored object than previously described in Section 2.1. That is why the implemented back-projection will fail. The chance of a wrong classification rises. It is understood that this also affects the vision cone.

A third drawback results from the applied Loop-Closure-module. It lurks for the robot to return to its previous position. This is based on the assumption that the object was bypassed and seen from the "right" perspective. It would be smarter if the robot automatically detects that it has bypassed a transparent or specular reflective object. Moreover, the robot only considers the necessary set of scans which contain the object. The rest of the history can remain untouched. This would save resources for other processes and speed up the Post-Filter. This leads directly to the next drawback.

Currently, the entire history is simply saved and processed as soon as the Post-Filter starts. A smart storage which minimizes data, helps to fasten the node and minor storage requirement.

Last but not least, a parallel processing improves the processing time of all modules. For example, the three functions identifying the type of object can be processed in parallel. This results from the fact that they do not depend on each other. Also, some filter-functions can be calculated in parallel.

The thesis outpoints that influences of transparent and specular reflective objects are detectable and it is possible to distinguish between them by solely applying a laser scanner. This improves mapping and helps robots to enter unknown areas without covering such objects. Nevertheless, there is still potential to improve this approach which opens opportunities for future work.

Appendix A

Appendix

A.1 Parameters of Hokuyo UTM-30LX-EW

The Hokuyo UTM-30LX-EW laser scanner [Hokuyo, 2017], applied for the experiments in Section 7, consists of following parameters:

Parameter:	**Value:**
Scan points (N_{Laser}):	1081
Scan frequency (f_{scan}):	50 Hz
Scan angle (α_{Laser}):	270°
Angular resolution (α_{step}):	0.25°
Wave length (λ):	905 nm
Distance range of Echo 1 and Echo 2 (d_1, d_2):	$0.1 - 60$ m
Intensity range (I_1, I_2):	$0 - 2^{24}$ (*)

(*) the measured intensity during the experiments was < 22000

A.2 Parameters for Rotating 3D-Hokuyo-node

The rotating 3D-Hokuyo-node applies an Hokuyo UTM-30LX-EW with its parameters (see Section A.1). The node parameters were set to:

Parameter:	**Value:**
calibrate_time	false
publish_intensity	true
publish_multiecho	true
error_limit	4
speed	10.0 rpm
fragmentSize	0.0

A.3 Parameters of Sick

The Sick TIM551-2050001 laser scanner SICK [2017], applied for the experiments in Section 7, consists of following parameters:

Parameter:	**Value:**
Scan points (N_{Laser}):	271
Scan frequency (f_{scan}):	15 Hz
Scan angle (α_{Laser}):	270°
Angular resolution (α_{step}):	1°
Wave length (λ):	850 nm
Distance range (d_1)	$0.05 - 10$ m

A.4 Parameters for Loop-Closure-node

The parameters for the Loop-Closure-node were set to:

Parameter:	**Value:**
thres_x	0.5 m
thres_y	0.5 m
thres_z	0.5 m
checkAfterPoses	500

A.5 Parameters for 2D-Mirror-Identifier-Approach

During the experiments, the variables of the 2D-Mirror-Identifier-Approach were set to following values.

A.5.1 Parameters for 2D-Pre-Filter-node

For the 2D-Pre-Filter parameters were set to:

Parameter:	**Value:**
substract_threshold_dist	0.05 m
multiline	true
particlefilter_threshold_dist_mirror	0.05 m
particlefilter_threshold_dist_affected	0.08 m
particlefilter_threshold_angle	10 steps $\hat{=}$ 2.5°
ransac_threshold	0.04 m
ransac_iterations	100
ransac_points2fit	30
minMeasureDistance	300 mm
maxMeasureDistance	60000 mm

A.5.2 Parameters for 2D-Post-Filter-node V1

The 2D-Post-Filter parameters were set to:

Parameter:	**Value:**
thres_mirrorcorner	0.15 m
thres_mirrorline	0.05 m
thres_openingAnglePrefilter	0.5
thres_angleThreshold	10 steps $\hat{=}$ 2.5°
min_range	0.0 m
max_range	30.0 m
low_reflectivity_range	3.0

A.5.3 Parameters for 2D-Post-Filter-node V2

The 2D-Post-Filter parameters were set to:

Parameter:	**Value:**
thres_mirrorcorner	0.15 m
thres_mirrorline	0.05 m
thres_openingAnglePrefilter	0.5
thres_angleThreshold	10 steps $\hat{=}$ 2.5°
min_range	0.0 m
max_range	30.0 m
low_reflectivity_range	3.0
thres_MinPointsICP	100
thres_MaxDistICPTrans	0.1 m
thres_MaxAngleICPTrans	10
ransac_threshold	0.04 m
ransac_iterations	100
ransac_points2fit	20

A.6 Parameters for 3D-Mirror-Identifier-Approach

For the 3D-Mirror-Identifier-Approach following parameters used during the experiments.

A.6.1 Parameters for 3D-Pre-Filter-node

For the 3D-Pre-Filter parameters were set to:

Parameter:	**Value:**
pub_Results	false
min_measure_distance	0.3 m
max_measure_distance	30.0 m
box_min_x_distance	−0.3 m
box_min_y_distance	−0.3 m
box_min_z_distance	−0.1 m
box_max_x_distance	0.3 m
box_max_y_distance	0.3 m
box_max_z_distance	0.3 m
outlierFilter_minNeighbours	30
outlierFilter_searchRadius	0.1 m
substract_threshold_distance	0.05 m
planeDetection_threshold_distance	0.05 m
planeDetection_minAmountPoints	100
visionCone_threshold	0.05 m
max_Dist_ICP	0.6 m
max_Rot_ICP	0.175
fitness_Fct_ICP	0.1
thres_mean_intens_factor	1.0
sizeVarFactor	0.2
threshold_variation_intensity	0.4

A.6.2 Parameters for 3D-Post-Filter-node

For the 3D-Post-Filter parameters were set to:

Parameter:	**Value:**
pub_Results	false
activateFilters	false
min_measure_distance	0.3 m
max_measure_distance	30.0 m
box_min_x_distance	−0.3 m
box_min_y_distance	−0.3 m
box_min_z_distance	−0.1 m
box_max_x_distance	0.3 m
box_max_y_distance	0.3 m
box_max_z_distance	0.3 m
outlierFilter_minNeighbours	30
outlierFilter_searchRadius	0.1 m
substract_threshold_distance	0.05 m
planeDetection_threshold_distance	0.05 m
planeDetection_minAmountPoints	100
visionCone_threshold	0.05 m
max_Dist_ICP	0.6 m
max_Rot_ICP	0.175
fitness_Fct_ICP	0.1
thres_mean_intens_factor	1.0
sizeVarFactor	0.2
threshold_variation_intensity	0.4

If the Filter-functions should be applied ($activateFilters = true$), following variafbles can be set:

Parameter:	**Value:**
min_measure_distance	0.3 m
max_measure_distance	30.0 m
box_min_x_distance	−0.3 m
box_min_y_distance	−0.3 m
box_min_z_distance	−0.1 m
box_max_x_distance	0.3 m
box_max_y_distance	0.3 m
box_max_z_distance	0.3 m
outlierFilter_minNeighbours	30
outlierFilter_searchRadius	0.1 m

A.7 Parameters for Mapping-Approaches

The applied mapping approaches utilized their standard parameters.

A.7.1 CRSM-SLAM

For the CRSM-SLAM the standard parameters are:

Parameter:	**Value:**
hill_climbing_disparity	40
slam_container_size	500
slam_occupancy_grid_dimentionality	0.02 m
map_update_density	40
map_update_obstacle_density	3.0
scan_density_lower_boundary	0.3
max_hill_climbing_iterations	1000
desired_number_of_picked_rays	30
occupancy_grid_map_freq	1.0 Hz
robot_pose_tf_freq	5.0 Hz
trajectory_freq	1.0 Hz
robot_width	0.6 m
robot_length	0.75 m

A.7.2 HECTOR-SLAM

For the HECTOR-SLAM the standard parameters are:

Parameter:	**Value:**
use_tf_scan_transformation	false
use_tf_pose_start_estimate	false
pub_map_odom_transform	true
map_resolution	0.05 m
map_size	2048
map_start_x	0.5
map_start_y	0.5
map_multi_res_levels	2
update_factor_free	0.4
update_factor_occupied	0.9
map_update_distance_thresh	0.4 m
map_update_angle_thresh	0.06
laser_z_min_value	−1.0 m
laser_z_max_value	1.0 m
advertise_map_service	true
scan_subscriber_queue_size	5
output_timing	false
pub_drawings	true
pub_debug_output	true

A.7.3 TSD-SLAM

For the TSD-SLAM the standard parameters are:

Parameter:	**Value:**
x_offset	0.0 m
y_offset	0.0 m
yaw_offset	0.0°
map_size	11
cellsize	0.015 m
truncation_radius	10
min_range	0.0 m
max_range	30.0 m
object_inflation_factor	1
registration_mode	0
icp_iterations	80
reg_trs_max	1.0 m
reg_sin_rot_max	0.5
trials	30
sizeControlSet	360
use_odom_rescue	false
max_velocity_rot	6.28 $^{m}/_{s}$
max_velocity_lin	1.0 m
zrand	0.05 m

A.7.4 OctoMap

For the OctoMap the standard parameters are:

Parameter:	**Value:**
resolution	0.025 m
sensor_model/range	4.0 m
sensor_model/hit	0.8
sensor_model/miss	0.31
sensor_model/min	0.12 m
sensor_model/max	0.95 m
min_z_range	0.05 m
max_z_range	1.8 m
min_x_size	25.0 m
min_y_size	25.0 m
pointcloud_min_z	0.10 m
pointcloud_max_z	1.80 m
filter_ground	true
occupancy_min_z	0.05 m
occupancy_max_z	2.0 m
sensor_model/max_range	30.0 m

Abbreviations

General abbreviations:

cf. confer to figure

e.g. for example

et al. et alia

etc. et cetera

PhD Doctor rerum naturalium

Pos. position

StGB German criminal code

V1 Version 1

V2 Version 2

vs. versus

w.r.t. with respect to

Mathematical & physical related abbreviations:

1D 1 dimensional

2D 2 dimensional

2.5D 2.5 dimensional

3D 3 dimensional

AM amplitude modulation

DoF Degree-of-Freedom

FM frequency modulation

FOV field of view

IR infrared

NIR near-infrared

RKS robot coordinate system

ToF Time-of-Flight

WKS world coordinate system

Hardware related abbreviations:

CCD carbonate compensation depth

CMOS complementary metal-oxid-semiconductor

CPU central processing unit

FPS frames per second

GPS Global-Positioning-System

GPU graphical processing unit

IMU Inertial-Measurement-Unit

LED light emitting diode

LIDAR Light Detection and Ranging

RGB Red-Green-Blue

RGB-D Red-Green-Blue-Distance

Algorithms & Software related abbreviations:

CIE-LAB Commission internationale de l'èclairage- LAB
(L: lightness, A: color opponent green-red, B: color opponent blue-yellow)

CRSM Critical-Ray-Scan-Match

EKF Extended-Kalman-Filter

HECTOR Heterogeneous Cooperating Team of Robots

ICP Iterative Closest Point

KD K-Dimensional

KinFu KinectFusion

MRF Markov Random Field

NaN Not-a-Number

NDT Normal Distribution Transform

OBB outer bounding box

OctoMap Octograph based Mapping

OpenCV Open Source Computer Vision Library

PCA Principal Component Analysis

PCL Point Cloud Library

RANSAC Random-Sample-Consensus

ROS Robot Operating System

RRHC Random-Restart-Hill-Climbing

SAC Singleton-Arc-Consistency

SLAM Simultaneous-Localization-and-Mapping

SMG Scan-Matching-Genetic

TF Transformation

TIN triangulated irregular network

TSD Truncated-Signed-Distance

VisAGGE Visible Angle Grid for Glass Environments

List of Figures

References

20minuten.ch. Erdbebenserie in Italien fordert ein Todesopfer. URL http://www.20min.ch/panorama/news/story/Kirchturm-von-Amatrice-nach-Beben-eingestuerzt-28524993, January 2017. Online; accessed 11/03/2017. 1, 138

S. Albrecht. *Transparent Object Reconstruction and Registration Confidence Measures for 3D Point Clouds Based on Data Inconsistency and Viewpoint Analysis*. PhD thesis, Universität Osnabrück, Institute for Computer Science Knowledge Based Systems, 2017. 65, 66, 67

H. Ali, Ch. Seifert, N. Jindal, L. Paletta, and G. Paar. Window Detection in Facades. In Rita Cucchiara, editor, *ICIAP*, pages 837–842. IEEE Computer Society, 2007. ISBN 0-7695-2877-5. URL http://dblp.uni-trier.de/db/conf/iciap/iciap2007.html#AliSJPP07. 51

H. Ali, B. Ahmed, and G. Paar. Robust Window Detection from 3D Laser Scanner Data. *Congress on Image and Signal Processing*, 1:115–118, 2008. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4566279. 51, 67

M. Andriluka, S. Roth, and B. Schiele. Pictorial Structures Revisited: People Detection and Articulated Pose Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1014–1021, June 2009. DOI 10.1109/CVPR.2009.5206754. 2

Arduino. Arduino. URL http://www.arduino.cc, April 2017. Online; accessed 11/01/2017. 20

AutonOHM. Obviously Library from TEAM AutonOHM of TH Nuremberg. URL https://github.com/autonohm/obviously, April 2017. Online; accessed 11/02/2017. 81

M. Awais. Improved Laser-Based Navigation for Mobile Robots. In *2009 International Conference on Advanced Robotics (ICAR)*, pages 1–6, 2009. 58, 59, 60, 67, 139

P.J. Besl and N.D. McKay. A Method for Registration of 3D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:239–256, 1992. ISSN 0162-8828. DOI http://doi.ieeecomputersociety.org/10.1109/34.121791. 44

P. Biber and W. Strasser. The Normal Distributions Transform: A New Approach to Laser Scan Matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2743–2748, October 2003. DOI 10.1109/IROS.2003.1249285. 46

M.A. Fischler and R.C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*, 24(6): 381–395, June 1981. ISSN 0001-0782. DOI 10.1145/358669.358692. 74

FLIR. FLIR Bumblebee2. URL https://www.ptgrey.com/bumblebee2-firewire-stereo-vision-camera-systems, April 2017. Online; accessed 11/01/2017. 23, 138

P. Foster, Z. Sun, J.J. Park, and B. Kuipers. VisAGGE: Visible Angle Grid for Glass Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2213–2220, 2013. DOI 10.1109/ICRA.2013.6630875. 60, 61, 67

K. Fujinoki. *Redundant Multiscale Haar Wavelet Transforms*, pages 443–449. Springer International Publishing, Cham, 2015. ISBN 978-3-319-12577-0. DOI 10.1007/978-3-319-12577-0_49. 51

G. Grisetti, C. Stachniss, and W. Burgard. Improving Grid-Based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2432–2437. IEEE, 2005. URL http://dblp.uni-trier.de/db/conf/icra/icra2005.html#GrisettiSB05. 62

A. Hammer, H. Hammer, and K. Hammer. *Taschenbuch der Physik*. Lindauer, 2012. ISBN 9783874880954. URL https://books.google.de/books?id=O3KAMwEACAAJ. 7, 13

Hokuyo. Hokuyo UTM-30LX-EW. URL https://www.hokuyo-aut.jp, April 2017. Online; accessed 11/02/2017. 22, 125, 138

D. Holz, D. Droeschel, S. Behnke, St. May, and H. Surmann. Fast 3D Perception for Collision Avoidance and SLAM in Domestic Environments. In Alejandra Barrera, editor, *Mobile Robots Navigation*, pages 53–84. IN-TECH Education and Publishing, Vienna, Austria, March 2010. ISBN 978-953-307-076-6. 2

A. Hornung, K.M. Wurm, and M Bennewitz. Humanoid Robot Localization in Complex Indoor Environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1690–1695, October 2010. DOI 10.1109/IROS.2010.5649751. 40

A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomeous Robots*, 34(3):189–206, April 2013. ISSN 0929-5593. DOI 10.1007/s10514-012-9321-0. 40, 43, 139

I. Ihrke, K. Kutulakos, H. Lensch, M. Magnor, and W. Heidrich. Transparent and Specular Object Reconstruction. *Computer Graphics Forum*, 29(8):2400–2426, December 2010. 48, 49, 50, 139

Xiaoyi Jiang and Horst Bunke. *Dreidimensionales Computersehen: Gewinnung und Analyse von Tiefenbildern*. Springer-Verlag, Berlin und Heidelberg, 1997. 14, 15, 18

P.-F. Käshammer and A. Nüchter. Mirror Identification and Correction of 3D Point Clouds. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W4:109–114, 2015. DOI 10.5194/isprsarchives-XL-5-W4-109-2015. URL http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-5-W4/109/2015/. 64, 66, 67

Kobuki. Kobuki Robots, Janurary 2017. URL http://kobuki.yujinrobot.com/. Online; accessed 11/01/2017. 111

P. Koch, St. May, and M. Kühn. ROS TSD-SLAM Package. URL http://wiki.ros.org/ohm_tsd_slam, November 2015a. Online; accessed 11/25/2015. 44, 70

Ph. Koch, St. May, M. Schmidpeter, M. Kühn, J. Martin, Ch. Pfitzner, Ch. Merkl, M. Fees, R. Koch, and A. Nüchter. Multi-Robot Localization and Mapping Based on Signed Distance Functions. In *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, https://www.waset.org/conference/2015/03/istanbul/ICARSC, March 2015b. 44

Ph. Koch, St. May, M. Schmidpeter, M. Kühn, Ch. Pfitzner, Ch. Merkl, R. Koch, M. Fees, J. Martin, D. Ammon, and A. Nüchter. Multi-Robot Localization and Mapping Based on Signed Distance Functions. *Journal of Intelligent & Robotic Systems*, pages 1–20, 2016. ISSN 1573-0409. DOI 10.1007/s10846-016-0375-7. Koch2016a. 44

R. Koch, St. May, L. Böttcher, M. Jahrsdörfer, J. Maier, M. Trommer, and A. Nüchter. Out of Lab Calibration of a Rotating 2D Scanner for 3D Mapping. In *Proc. SPIE 10332, Videometrics, Range Imaging, and Applications XIV, 1033207*, June 2017a. DOI 10.1117/12.2270298. 27

R. Koch, St. May, P. Murmann, and A. Nüchter. Identification of Transparent and Specular Reflective Material in Laser Scans to Discriminate Affected Measurements for Faultless Robotic SLAM. *Robotics and Autonomous Systems*, 87:296–312, 2017b. ISSN 0921-8890. DOI 10.1016/j.robot.2016.10.014. URL http://www.sciencedirect.com/science/article/pii/S0921889015302736. 47

St. Kohlbrecher and J. Meyer. ROS Hector-SLAM Package. URL http://wiki.ros.org/hector_slam, November 2015. Online; accessed 11/01/2017. 36

St. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klinga. A Flexible and Scalable SLAM System with Full 3D Motion Estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160, November 2011. DOI 10.1109/SSRR.2011.6106777. 37, 38, 139

St. Kohlbrecher, J. Meyer, Th. Graber, K. Petersen, U. Klingauf, and O. von Stryk. Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots. In Sven Behnke, Manuela M. Veloso, Arnoud Visser, and Rong Xiong, editors, *RoboCup*, volume 8371 of *Lecture Notes in Computer Science*, pages 624–631. Springer, 2013. ISBN 978-3-662-44467-2. URL http://dblp.uni-trier.de/db/conf/robocup/robocup2013.html#KohlbrecherMGPKS13. 37, 139

J.B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, Princeton, NJ, 1999. ISBN 0691058725 9780691058726. URL http://www.worldcat.org/title/quaternions-and-rotation-sequences-a-primer-with-applications-to-orbits-aerospace-and-virtual-reality/oclc/246446345. 40

X.-C. Lai, C.-Y. Kong, S.S. Ge, and A.A. Mamun. Online Map Building for Autonomous Mobile Robots by Fusing Laser and Sonar Data. In *IEEE International Conference Mechatronics and Automation*, volume 2, pages 993–998 Vol. 2, July 2005. DOI 10.1109/ICMA.2005.1626687. 56, 67

St.Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer Publishing Company, Incorporated, 3rd edition, 2009. ISBN 9781848002784. 60

B.D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision (DARPA). In *Proceedings of the 1981 DARPA Image Understanding Workshop*, pages 121–130, April 1981. 39

M. Magnusson. *The Three-Dimensional Normal-Distributions Transform : An Efficient Representation for Registration, Surface Analysis, and Loop Detection*. PhD thesis, Örebro University, School of Science and Technology, 2009. 46

St. May, R. Koch, A. Nüchter, Ch. Pfitzner, Ph. Koch, and Ch Merkl. A Generalized 2D and 3D Multi-Sensor Data Integration Approach Based on Signed Distance Functions for Multi-Modal Robotic Mapping. In *Vision, Modeling and Visualization*, URL http://www.vmv2014.de/, October 2014. 44, 45, 46, 47

MESA. HEPTAGON. URL http://hptg.com/industrial, April 2017. Online; accessed 11/01/2017. 25

Microsoft. Microsoft Kinect Sensor. URL https://msdn.microsoft.com/en-us/library/hh438998.aspx, April 2017. Online; accessed 11/03/2017. 24, 25, 138

G. Mingas, E. Tsardoulias, and L. Petrou. An FPGA Iimplementation of the SMG-SLAM Algorithm. *Microprocessors and Microsystems*, 36(3):190–204, 2012. ISSN 0141-9331. DOI http://dx.doi.org/10.1016/j.micpro.2011.12.002. URL http://www.sciencedirect.com/science/article/pii/S0141933111001244. 28, 31

R.A. Newcombe, A.J. Davison, Sh. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, St. Hodges, D. Kim, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. *10th IEEE International Symposium on Mixed and Augmented Reality*, 7(10): 127–136, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6092378. 44

H. Nobach. *Optische Messtechnik*. Borsdorf : Ed. Winterwork, 1 edition, July 2012. URL http://d-nb.info/1024256995. 8, 12, 13, 138

A. Nüchter. *3D Robotic Mapping: The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*. Springer Publishing Company, Incorporated, 1st edition, 2009. ISBN 3540898832, 9783540898832. 17

OpenCV. OpenCV-Library. URL http://opencv.org, April 2017. Online; accessed 11/01/2017. 65

J. Park, E.T. Matson, and J.-W. Jung. A Method to Localize Transparent Glass Obstacle using Laser Range Finder in Mobile Robot Indoor Navigation. In Jong-Hwan Kim, Eric T. Matson, Hyun Myung, Peter Xu, and Fakhri Karray, editors, *RiTA*, volume 274 of *Advances in Intelligent Systems and Computing*, pages 29–35. Springer, 2013. ISBN 978-3-319-05581-7. URL http://dblp.uni-trier.de/db/conf/rita/rita2013.html#ParkMJ13. 63, 64, 67, 139

PCL. PCL. URĹ http://pointclouds.org, April 2017. Online; accessed 11/01/2017. 90, 92

Pepperl+Fuchs. Pepperl+Fuchs. URL https://www.pepperl-fuchs.com, April 2017. Online; accessed 11/01/2017. 20

B.T. Phong. Illumination for Computer Generated Pictures. *Commun. ACM*, 18(6):311–317, June 1975. ISSN 0001-0782. DOI 10.1145/360825.360839. URL http://doi.acm.org/10.1145/360825.360839. 106

S. Pu and G. Vosselman. Extracting Windows from Terrestrial Laser Scanning. In *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 3–52, 2007. 51, 52, 67

S. Pu, G. Vosselman, and Commission Vi. Automatic Extraction of Building Features from Terrestrial Laser Scanning. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 33–39, 2006. 51

M. Recky and F. Leberl. Windows Detection using K-Means in CIE-LAB Color Space. *20th International Conference on Pattern Recognition*, 0:356–359, 2010. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5597805. 51

ROS. ROS. URĹ http://www.ros.org, July 2015. Online; accessed 11/03/2017. 68

St. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Pearson, 3 edition, December 2009. ISBN 0136042597. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0136042597. 34

SICK. SICK TIM551-2050001. URL https://www.sick.com, July 2017. Online; accessed 11/02/2017. 126

SSRR, Summer, and School. RoboCupRescue Robot League Rules for 2013, May 2013. URL http://wiki.ssrrsummerschool.org/doku.php?id=rrl-rules-2013. Online; accessed 11/04/2017. 2

H. Surmann, A. Nüchter, and J. Hertzberg. An Autonomous Mobile Robot with a 3D Laser Range Finder for 3D Exploration and Digitalization of Indoor Environments. *Robotics and Autonomous Systems*, 45:181–198, 2003. DOI 10.1016/j.robot.2003.09.004. 26

A. Tatoglu and K. Pochiraju. Point Cloud Segmentation with LIDAR Reflection Intensity Behavior. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 786–790, 2012. DOI 10.1109/ICRA.2012.6225224. 62, 108

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005. 58, 62

E. Tsardoulias and P. Loukas. Critical Rays Scan Match SLAM. *Journal of Intelligent & Robotic Systems*, 72(3):441–462, 2013. ISSN 1573-0409. DOI 10.1007/s10846-012-9811-5. 28, 29, 33, 34, 35, 139

M. Tsardoulias. ROS CRSM-SLAM Package. URL http://wiki.ros.org/crsm_slam, November 2015. Online; accessed 11/04/2017. 28

Velodyne. Velodyne LiDAR. URL http://www.velodynelidar.com/hdl\discretionary{-}{}{}32e.html, April 2017. Online; accessed 11/04/2017. 25, 26

R. Wang, J. Bach, W.R. Street, and F.P. Ferrie. *Window Detection from Mobile LIDAR Data*, pages 58–65. IEEE, 2010. URL http://www.cim.mcgill.ca/~ruisheng/software/WindowDetectionFromMobileLiDARData_WACV_CameraReady.pdf. 52, 53, 54, 67, 139

X. Wang and Wang J. Detecting Glass in Simultaneous Localisation and Mapping. *Robotics and Autonomous Systems*, 88:97–103, 2017. URL http://dblp.uni-trier.de/db/journals/ras/ras88.html#WangW17. 61, 62, 67, 108, 139

Wikipedia. Erdbeben in Mittelitalien 2016. URL https://de.wikipedia.org/wiki/Erdbeben_in_Mittelitalien_2016, February 2017. Online; accessed 11/05/2017. 1

O. Wulf and B. Wagner. Fast 3D-Scanning Methods for Laser Measurement Sstems. In *In the 14th International Conference on Control Systems and Computer Science*, 2003. 27, 139

K.M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, 2010. URL http://octomap.sourceforge.net/. 40, 44

K.M. Wurm, D. Hennes, D. Holz, R.B. Rusu, C. Stachniss, K. Konolige, and W. Burgard. Hierarchies of Octrees for Efficient 3D Mapping. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4249–4255, September 2011. DOI 10.1109/IROS.2011.6094571. 40

S.-W. Yang and C.-C. Wang. Dealing with Laser Scanner Failure: Mirrors and Windows. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3009–3015, Pasadena, California, May 2008. 54, 56, 67

S.-W. Yang and C.-C. Wang. On Solving Mirror Reflection in LIDAR Sensing. *IEEE/ASME Transactions on Mechatronics*, 16(2):255–265, 2011. ISSN 1083-4435. DOI 10.1109/TMECH.2010.2040113. 56, 57, 58, 67, 139

Die Schriftenreihe

wird vom Lehrstuhl für Informatik VII: Robotik und Telematik der Universität Würzburg herausgegeben und präsentiert innovative Forschung aus den Bereichen der Robotik und der Telematik.

Die Kombination fortgeschrittener Informationsverarbeitungsmethoden mit Verfahren der Regelungstechnik eröffnet hier interessante Forschungs- und Anwendungsperspektiven. Es werden dabei folgende interdisziplinäre Aufgabenschwerpunkte bearbeitet:

- Robotik und Mechatronik: Kombination von Informatik, Elektronik, Mechanik, Sensorik, Regelungs- und Steuerungstechnik, um Roboter adaptiv und flexibel ihrer Arbeitsumgebung anzupassen.

- Telematik: Integration von Telekommunikation, Informatik und Steuerungstechnik, um Dienstleistungen an entfernten Standorten zu erbringen.

Anwendungsschwerpunkte sind u.a. mobile Roboter, Tele-Robotik, Raumfahrtsysteme und Medizin-Robotik.

Lehrstuhl Informatik VII
Robotik und Telematik
Am Hubland
D-97074 Wuerzburg

Tel.: +49 (0) 931 - 31 - 86678
Fax: +49 (0) 931 - 31 - 86679

schi@informatik.uni-wuerzburg.de
http://www7.informatik.uni-wuerzburg.de

ISSN: 1868-7474 (online)
ISSN: 1868-7466 (print)
ISBN: 978-3-945459-25-6 (online)

Zitation dieser Publikation

KOCH, R. (2018). Sensor Fusion for Precise Mapping of Transparent and Specular Reflective Objects. Schriftenreihe Würzburger Forschungsberichte in Robotik und Telematik, Band 16. Würzburg: Universität Würzburg.
URN: urn:nbn:de:bvb:20-opus-163462