FH Salzburg

BACHELOR THESIS 2

Detecting Obstacles in 3D Point Clouds for Collaborative Robot Environments

carried out at the
Information Technology and Systems Management
University of Applied Sciences Salzburg

submitted by
Armin Niedermüller

Course Director: FH-Prof. DI Dr. Gerhard Jöchtl
Supervisor: FH-Prof. DI (FH) DI Dr. Andreas Unterweger

Salzburg, June 2020

Declaration on Oath

I hereby declare on oath that I have written this bachelor's thesis on my own and without external assistance and have used no other source materials or aids than those listed. Furthermore, I hereby affirm that I have identified any text or content taken from the source materials used accordingly.

To date, this paper has neither been submitted to any other examination board, in this country or abroad, nor published in an identical or similar form.

14th June 2020
Date

Signature

With the advent of inexpensive 3D sensors and open source robotics software, low-cost Human-Robot Collaboration (HRC) systems are becoming more competitive against expensive and proprietary systems. For such systems to be safe, the Cobot must sense obstacles before touching them. Strict technical requirements exist to realize a safe and efficient system to work with, such as low latency, high accuracy and low memory consumption. This bachelor thesis proposes a low-cost and open source prototype to enable collision detection for an industrial Collaborative Robot (Cobot) inside a HRC workspace.

The 3D images of two ceiling-mounted cameras are converted into a occupancy voxel grid and the latency together with the memory consumption is analyzed during conversion. Besides, different comparisons between real-world objects and their voxel grid counterpart are conducted to find out the accuracy. Since the voxel grid includes voxels from the Cobot itself, the Cobot can be misinterpreted as an obstacle and can thus be inoperable. Those voxels are removed by a Robot-Self Filter (RS-Filter), whose parameters are first optimized before measuring its efficiency. At last, the voxel grid together with a Cobot 3D model are visualized inside a virtual workspace.

A low latency is reached, and the memory consumption decreases sharply in the tests after optimizing the OctoMap parameters. Altough the measured accuracy is not suited for fine-grain tasks like the gripping of objects with the Cobot itself, it is good enough for the detection of obstacles, which is the main goal of this work. Considering those results, together with the RS-Filter removing all Cobot voxels, the prototype is suited for collision aware path planning. All in all, the goals of this thesis are reached, and the prototype represents a solid foundation for a HRC system in which humans can safely work together with Cobots in close proximity.

Contents

Acronyms

Cobot Collaborative Robot

CPU Central Processing Unit

DoF Degrees Of Freedom

GPU Graphics Processing Unit

GUI Graphical User Interface

HRC Human-Robot Collaboration

IR Infrared

LIDAR Light Detection And Ranging

SOR-Filter Statistical Outlier Removal Filter

PT-Filter Pass-Through Filter

PCL Point Cloud Library

RAM Random-Access Memory

ROS Robot Operating System

SDK Software Development Kit

RS-Filter Robot-Self Filter

SIFT Scale-Invariant Feature Transform

SLAM Simultaneous Location And Mapping

SL Structured Light

URDF Unified Robot Description Format

List of Figures

List of Tables

1 Introduction

In the introduction, Section 1.1 first describes the problem and the motivation. Then, Section 1.2 presents the objectives of the work and Section 1.3 shows the resulting structure.

1.1 Motivation

With the growing availability of Collaborative Robots (Cobots), humans can work together with robots in close proximity to execute specific tasks without the problem of being hurt. Cobots possess sensors (e.g. torque sensors) and use them to stop their movement when contacting obstacles. In addition, they move at a limited speed and are lightweight [1].

Cobots themselves can only sense obstacles when they already have touched them and thus realize a post-collision safety strategy. Even with rounded edges, humans can be hurt, especially when the collisions occur around the facial region of the human. One way of improving this situation is the use of 3D sensors (e.g. stereo cameras), which ultimately should enable the Cobot to sense obstacles before touching them [1]–[3]. This pre-collision safety strategy can provide a higher degree of safety and improve workflow between the human and the Cobot [4], [5].

1.2 Goals

In the scope of this work, a prototype virtual environment for a Human-Robot Collaboration (HRC) workspace is implemented. The prototype uses the 3D point clouds of 3D cameras to build a 3D geometric representation of the surrounding environment of a HRC workspace including a Cobot. The ultimate goal is an efficient and fast 3D representation of free, occupied and unmapped areas around the Cobot, often referred to as *occupancy maps*. In this thesis, we focus on the use of 3D voxel (volumetric pixel) grids to map the environment. The prototype should include a 3D visualization and simulation of a 3D Cobot model inside the workspace together with a live 3D voxel grid. The Cobot model should later be used for collision-aware path planning.

This work examines how the proposed application can be implemented and used to enable collision detection for a Cobot using two Intel D435 low-cost 3D cameras. Due to the involvement of people in an HRC, the frame rate of updating a voxel grid is an important factor. A lower frame rate causes a higher latency that can cause unwanted collisions between the Cobot and its environment or humans.

Further important factors for the prototype to consider are the accuracy and resolution of the 3D representation. Both must be high enough to distinguish the Cobot and obstacles from the work desk inside the voxel grid. Otherwise the Cobot could collide with objects inside the workspace. A higher resolution to recognize smaller objects also means a higher computational load. This could pose a problem for the performance of the voxel grid updating process. Therefore, we examine whether the accuracy of the low cost 3D cameras used here, is sufficient or whether other low-cost alternatives exist. Another factor to consider are outliers inside the point clouds. They emerge, inter alia, on sharp edges between objects nearer to the camera and objects which are further away. It is determined whether the point clouds have to be filtered to remove outliers, which are false points that emerge at the edges of an object but lie too far outside the object's geometry and seemingly fly in mid-air.

A separate problem is that the Cobot itself is shown as an occupied area inside the voxel grid, since both cameras are mounted above the Cobot and work desk. However, the voxel grid must only contain voxels which represent the surrounding area of the Cobot, not the Cobot itself. The reason is, that the Cobot later interprets the occupied voxels as obstacles to avoid. When the Cobot voxels are not removed from the voxel grid, the Cobot could recognize itself as an obstacle. This could lead to unforeseen problems (e.g. constantly getting stuck while trying to apply a collision avoiding path-planning of the Cobot). This raises the question of how voxels can be detected and marked as non-occupied areas, when they do not belong to the surroundings but to the Cobot.

1.3 Outline

The presented work is divided into 5 Sections:

Section 1 consists of this Introduction and describes the structure of this work. It defines the research focus of this thesis, the motivation of this work and the thesis goals.

Section 2 gives the basics for the realization of the prototype application.

Section 3 describes the used hard- and software to reach this work's goals. The conception of the prototype application and the used implementations are explained. Furthermore, the method for validating and improving the prototype is described.

Section 4 summarizes and discusses the results of the described methods. The effects of the individual methods presented in Section 4 of the prototype application are measured and then analyzed.

Section 5 delivers a conclusion of this work and discusses constraints and remaining problems. Moreover, it gives an outlook on future research.

2 Fundamentals

The following Section illustrates the basics to realize a collision detection for a Cobot. Besides analyzing the requirements of HRC systems and explaining the foundations of the used point cloud pre-processing, the basics of creating a map of the Cobot's environment are described. Additionally, the concept of a robot model including its kinematic and geometric information is explained, since afterwards both the environment map and the robot model are combined and visualized in a virtual work space.

2.1 Requirements Analysis

The ultimate goal of the prototype application is to enable collision detection for a Cobot inside an HRC workspace. The Cobot should work together with humans and therefore the reaction time of the system is the most critical requirement. Other work achieved real time behavior and frame rates of around 30 Hz [1]–[4].

This leads to the main task of minimizing the latency of the prototype as much as possible, which at the same time means increasing the frame rate. The latency is directly related to the computational effort of the prototype, which is, amongst other things, related to memory consumption. Complex computations and a high memory consumption requirement arise from handling dense point clouds. This is due to the fact that they consist of a large number of points in 3D space and thus contain more information to process.

It must be determined, whether and how accurate obstacles are recognized by the prototype. Furthermore, it should be possible to start the prototype with a single command to provide a fast and easy usable solution.

Altogether, the following requirements should be fulfilled:

- The latency of the prototype should be as low as possible. A low latency enables faster reaction times for the robot and thus means a higher level of safety. This automatically means a higher frame rate is necessary.
- The resolution of the voxel grid must be good enough to recognize obstacles such as hands or laboratory equipment.
- Memory consumption should be as low as possible, offering the possibility to add more cameras or changing the computational platform from a big personal computer to an embedded system.

2.2 Pre-Processing

The incoming point clouds consist of a large number of points. A resolution of 640 · 480 pixels still results in 307,200 points, which can pose a high workload on a Central Processing Unit (CPU). Different methods exist to do a pre-processing of the point clouds to reduce the amount of data and improve quality [6], e.g. remove outliers (Section 2.5).

2.3 Pass-Through Filter

One way of reducing the amount of data in point clouds is realized by cutting off one or more dimensions of the point cloud with a Pass-Through Filter (PT-Filter). This means that points which are outside of the PT-Filter's defined range are filtered out of the point cloud. Each point in a 3D point cloud has a X, Y and Z coordinate to represent its position inside 3D space [7]:

$$P = \begin{bmatrix} x & y & z \end{bmatrix}^T, x, y, z \in \mathbb{R} \tag{1}$$

A PT-Filter takes a minimum and maximum value for each dimension which should be filtered. Therefore, to filter the point cloud in the X dimension, X_{min} and X_{max} must be defined. If any point has a lower value X than X_{min} or higher than X_{max}, it is filtered out of the point cloud. Figure 1 shows a PT-Filter example in 2D, with given X_{min} and X_{max}.

2.4 Statistical Outlier Removal Filter

The task of the Statistical Outlier Removal Filter (SOR-Filter) is to remove points of a point cloud, which have been detected as outliers. The SOR-Filter assumes that the Euclidean distance of a point to its neighbors is normally distributed. When a point's mean distance is outside the global distances mean including a standard deviation, the point is removed [8]–[10].

2.5 3D Mapping

One of the fundamental research topics of robotics is the recognition and interpretation of the robot's environment. To interact autonomously with the environment, a robot has to recognize its surroundings and also find its own position in order to navigate freely. This is referred to as Simultaneous Location and Mapping (SLAM) [11] [12]. 2D maps might be sufficient for a mobile robot to navigate autonomously in different (2D) indoor

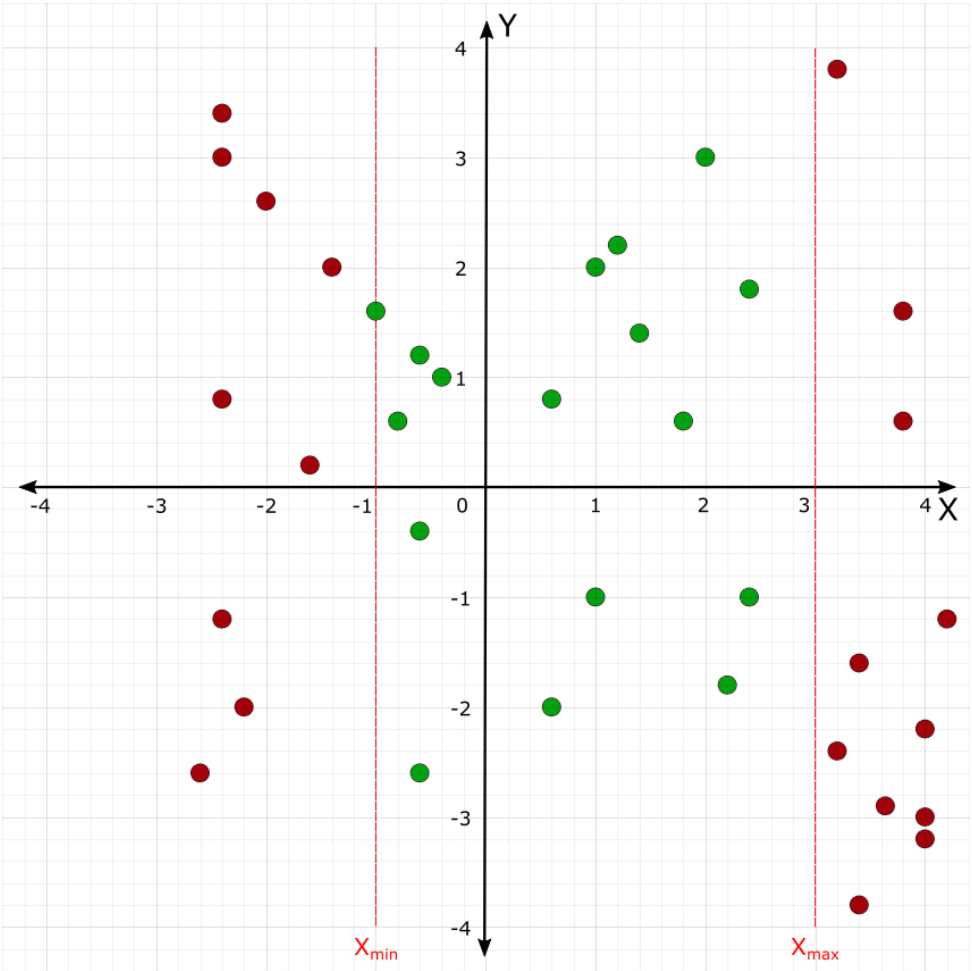

Figure 1: PT-Filter applied with a value of $X_{min} = -1$ and $X_{max} = 3$. Red points indicate that they are removed from the point cloud due to their value, while the green points remain. The red dashed lines represent X_{min} and X_{max}, respectively.

and outdoor situations. However, a 2D representation could easily miss objects with protruding edges, simply because the information about one dimension is missing. A Cobot moves in 3D space and therefore needs a 3D representation for 3D objects to be fully recognized. Thus, the goal is to create a 3D map of the Cobot's surroundings, which the Cobot then can use to avoid obstacles.

Methods like SLAM incrementally build a map of an environment where a mobile robot is moving. However, the 3D cameras in this work are mounted statically above the HRC workspace and see the Cobot's surroundings at all times. This means that the Cobot is always fully recognizable inside the point clouds, regardless of its current position. Creating a map (mapping) is therefore not necessary in the scope of this thesis and the focus lies only on the representation and optimization of the 3D map.

2.5.1 Categories of 3D Mapping

Different approaches exist to create a 3D representation of the Cobot's surroundings. They can be differentiated by how much information about the environment the 3D map contains and are primarily used to reduce the large amount of information in 3D point clouds. The most used techniques in mobile robotics are feature maps, geometric maps and (voxel) grid maps. Figure 2 gives an overview of the listed representations.

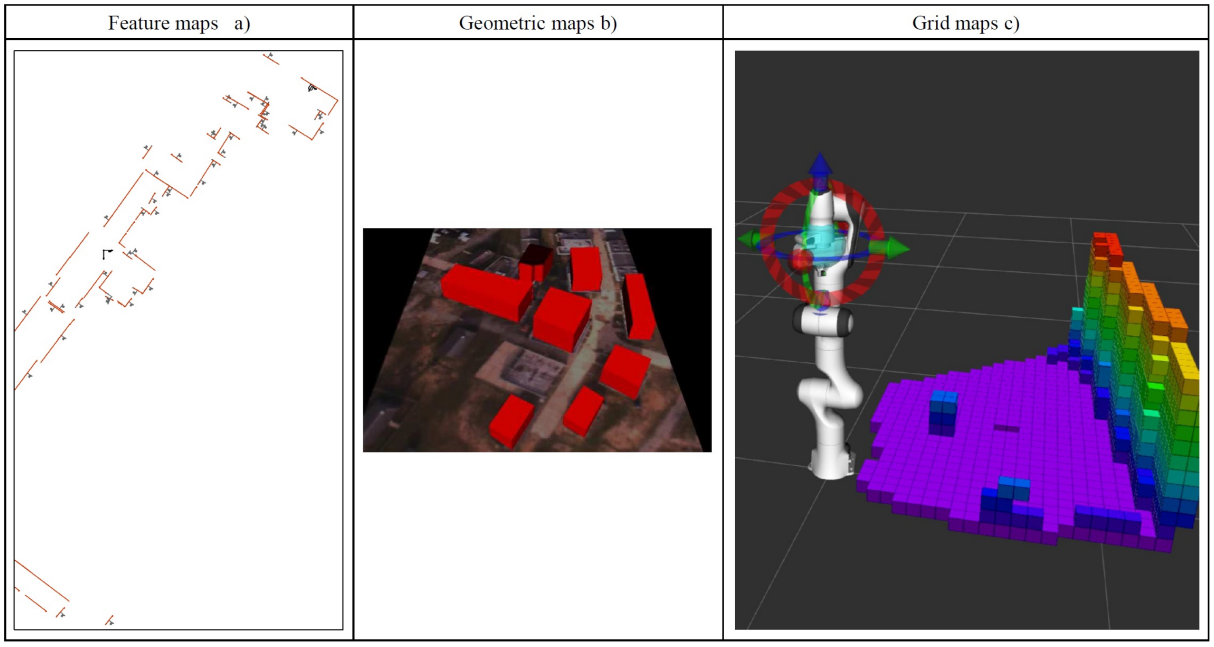

Figure 2: Comparison of different map representations: **a)** 2D map with edges as features [13], **b)** 3D Map consisting of geometric shapes [14], **c)** OctoMap, an occupancy based voxel-grid [15]

The feature-based approach (Figure 2.a) contains only specific properties of the environment. Either physical properties like corners and edges [16], or visual features computed

by algorithms like Scale-Invariant Feature Transform (SIFT) [17] are used. Another compact possibility are geometric maps (Figure 2.b), where perceived obstacles are represented as simple geometric forms or combinations of them [14].

This thesis uses OctoMap – an occupancy-based voxel grid – to realize the prototype, which represent the environment in grids consisting of equally sized cells (Figure 2.c). Each cell represents a part of the environment and holds information about whether the covered area has an obstacle in it. According to Hornung et al. [15], this leads to three requirements 3D mapping in robotic applications. These are the probabilistic representation of the environment, the modelling of unmapped areas and the efficiency.

2.5.2 Probabilistic Representation

3D sensors, be it Light Detection and Ranging (LIDAR) or 3D stereo cameras, are subject to range measurement errors. Those typically range from some milli- to centimeters [15] depending on the sensor. Furthermore, sensor noise poses a problem. This means that surfaces, which should be flat, are shown in a wavy pattern and further complicate an accurate measuring of distances. In addition, reflections or sharp edges between objects can, among other things, cause outliers. Outliers are points which belong to an object for example but lie too far outside of the object's geometry. A probabilistic representation of the robot's surroundings can deal with both sensor noise and outliers. This means that each cell contains a probability whether it is occupied or not.

2.5.3 Modelling of unmapped Areas

When a robot wants to avoid obstacles, blind spots in the 3D map pose a problem. Describing not only occupied and unoccupied areas, but also unmapped areas, a robot can avoid those areas and thus a possible collision with a hidden obstacle. When a 3D sensor detects a specific point inside the 3D image, the space in form of a line between the camera and the point is marked as unoccupied [15]. The voxel (point) itself is marked as occupied and everything else is marked as unmapped.

2.5.4 Memory Requirements and Computing Times

3D point clouds consist of a large number of small points scattered in 3D space and thus memory requirements and computing times are often the major bottleneck in 3D mapping. Different approaches exist to transform the point clouds to occupancy voxel grids, and thus reduce the computational load and the memory consumption [3, 4]. Reducing memory consumption and computing times means reducing reaction times and latency.

2.6 Robot Modelling

A real robot can also be represented as a 3D robot model to enable collision-aware path planning or robot voxel filtering. However, the 3D model should not only represent the Cobot's geometry, but also its *kinematic chain* to enable the same movement behavior as the real robot [18].

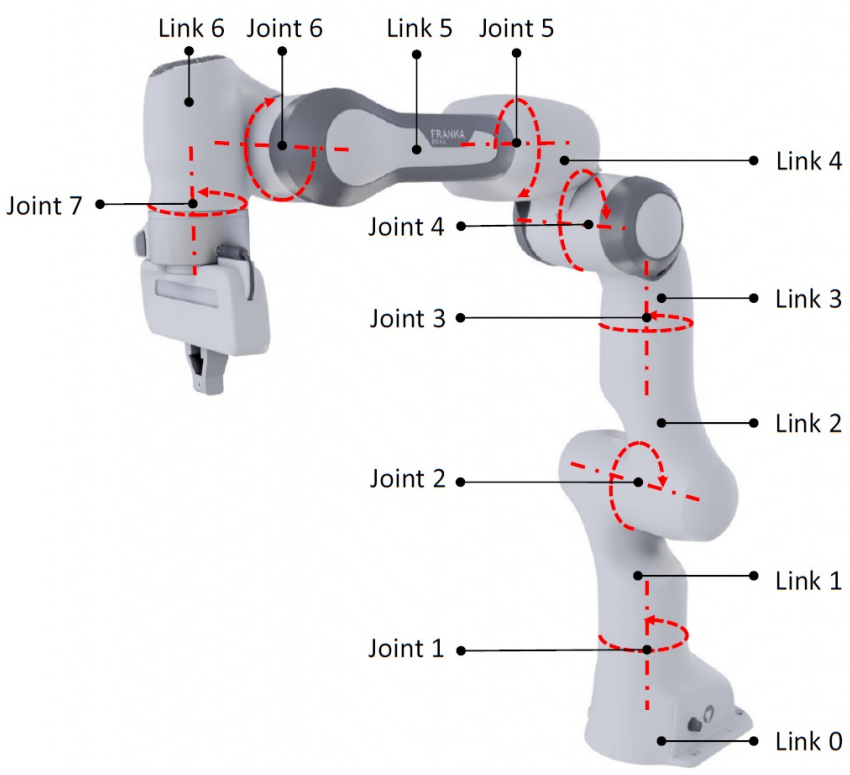

Figure 3: Kinematic chain of the *Franka Panda* Cobot [19]. The robot arm consists of each 7 links and joints. The red circles and dotted lines indicate the rotation axis of each joint. The interchangeable robot gripper would be the 8th joint.

2.6.1 Geometric Modelling

The geometric model represents how the parts of the robots exactly look like, i.e. the visible 3D parts of the robot. Moreover, it holds information about the collision geometry of the Cobot, which is the information about whether the Cobot has a collision with himself or with its environment.

2.6.2 Kinematic Modelling

The kinematic model holds information about the kinematic chain of the robot. The kinematic chain consists of the Cobot's rigid parts and the movable joints between them. Figure 3 shows the used Cobot in this thesis and its kinematic chain.

The rigid parts are also known as links and describe the dimensions of the actual Cobot parts, e.g. the base link of the Cobot. The base link (link 0) is the first part of the Cobot and mounted on the desk on its bottom. Connected to its top lies the first joint (joint 1). This joint is also connected to the next link (link 1) and interconnects both links. Each Cobot link is connected via such joints until the whole Cobot is described.

The kinematic chain describes also which axis each joint can be rotated, and which angles are possible. This is necessary due to the limited possible joint movement of the real Cobot. The kinematic model must have the same movement restrictions as the real Cobot. A possible self-collision of the Cobot due to too high or low joint values can only be known in advance at path-planning when the behavior of the model is exactly the same.

The kinematic chain is also used to describe the exact position of the real Cobot. The positions of the real joints are read out and applied to the 3D model. Alternatively, the joint positions for the kinematic model can be set programmatically to move the robot model into a specific position which is then used as goal for the path-planning.

With this section, we have established the theoretical foundation for this thesis. In the next section, the conceptual architecture and the hardware and software of the prototype are described.

3 Used Soft- and Hardware

In Section 3.1, the necessary steps to reach the goal of this thesis are described. Furthermore, the HRC work space, which is used in this thesis, is shown in Section 3.2. Before defining the methods to validate the features of the prototype, Sections 3.2 and 3.3 also describe what hard- and software is used to realize the prototype.

3.1 Prototype Outline

Figure 13 gives an overview about the necessary steps to create the prototype. In previous work, two 3D Cameras were mounted on the ceiling above the HRC workspace and their point clouds were aligned [8]. Both steps are prerequisites for the prototype described in this thesis. This thesis focuses on the pre-processing of the point clouds, their conversion into an OctoMap, the visualization of the workspace and the Robot-Self Filter (RS-Filter). The goal of this work is to enable a collision-aware Cobot path-planning in future work.

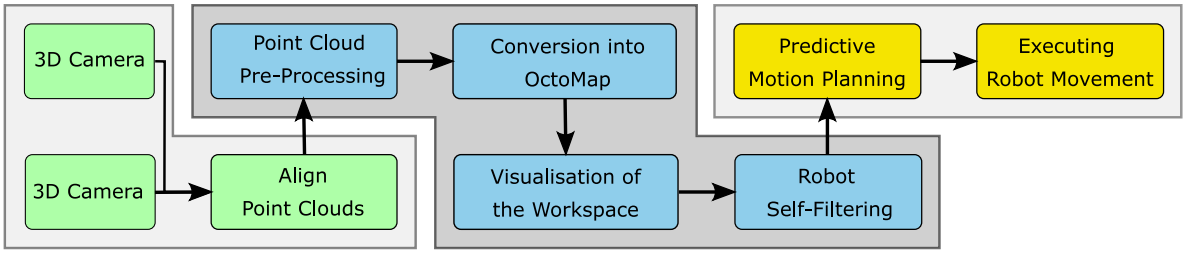

Figure 4: Overview of the steps conducted in the prototype application. The green rectangles show the previous work which was needed to realize the prototype. The blue rectangles show the actual work in this thesis and yellow indicates future work which should be enabled by this thesis.

Knowing the goals, the next Section describes the used hardware setup for the prototype.

3.2 Hardware Setup

In this thesis, the HRC workspace is observed by two Intel D435i 3D cameras, mounted on the ceiling above a HRC workspace. A desk mounted industrial Cobot – Franka Panda [19] – with 7 degrees of freedom (DoF) together with a personal computer are also used (Figure 6). The use of one camera alone results in blind spots in the image by occlusion of various objects in the workspace [8].

Figure 5: The 3D cameras and mounting hardware used in this thesis

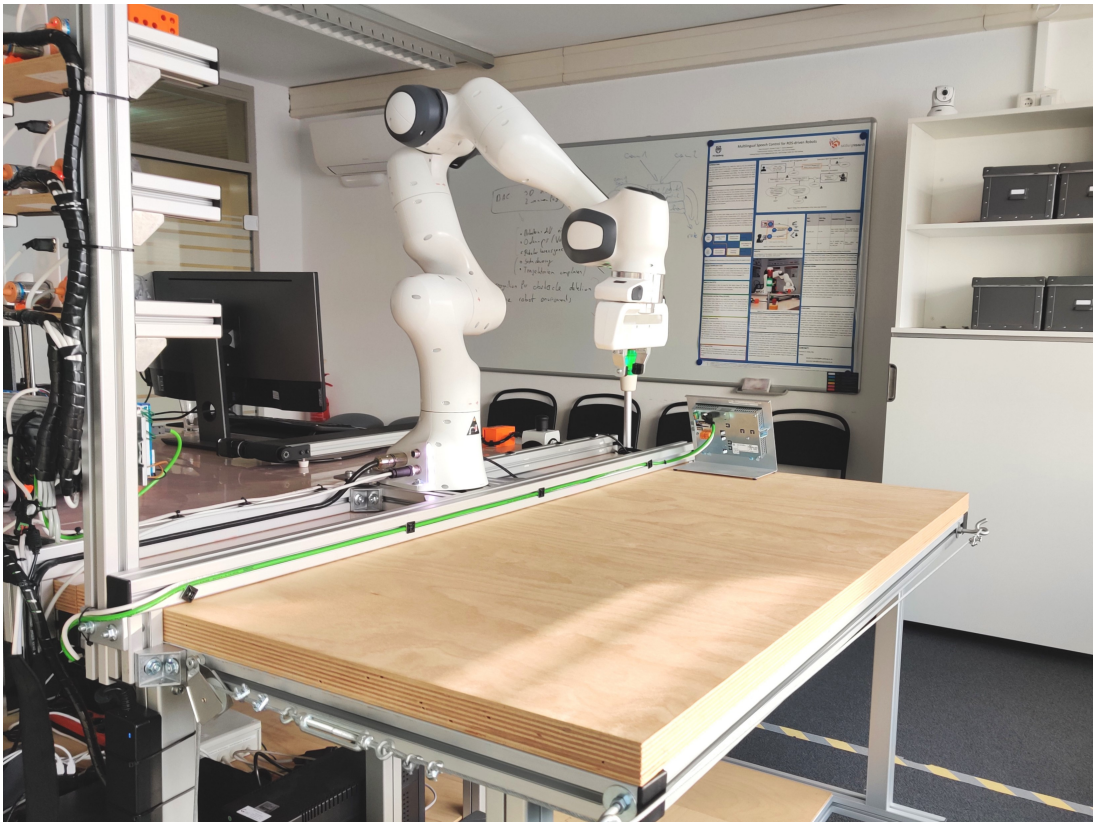

Figure 6: The HRC workspace used in this thesis

3.2.1 Computing Platform

Due to the high computational demands of point cloud processing, the main components of the computer used in this thesis are an Intel i7-8700K 6-core CPU, 32 GB of DDR4 Random-Access Memory (RAM) and a Nvidia 2070 RTX Super Graphics Processing Unit (GPU).

3.2.2 3D Camera

The D435i is a camera released in November 2018 by Intel, which is able to create 3D images. It was designed for applications such as robotics, augmented and virtual reality [20]. Intel already provides wrappers for popular platforms and languages, such as the Robot Operating System (ROS) and C++, with their open-source Intel RealSense Software Development Kit (SDK). Due to its low price of approximately 220 , the benefits of its large user community and the full integration to ROS, this camera is used in this thesis.

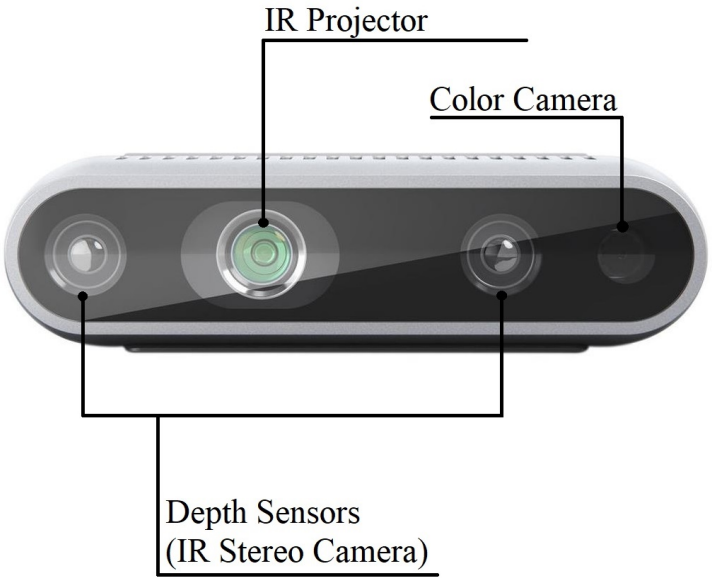

Figure 7: Parts of the Intel D435i 3D Camera, adapted from [20]

This camera uses two parallel infrared (IR) imaging sensors and an infrared projector to create 3D images. Moreover, an extra built-in color camera allows to capture colored 2D images. Figure 7 shows the camera with its most important parts.

The distance of an object in front of the camera is found by calculating the disparity of the object. The disparity is the difference of the location of an object inside the image of the left sensor compared to the right sensor. More specifically, a single point inside

Characteristic	**Intel D435i**
Depth Field of View	85.2 ° · 58 ° (±3 °)
Depth Resolution	up to 1280 · 720
Depth Frame Rate	up to 90 Hz
Depth Range	0.11 m - ∼10 m
RGB Resolution	1920 · 1080 at 30 Hz

Table 1: Characteristics of the Intel D435i 3D Camera [20]

the left image must be matched to the same pixel inside the right image to calculate the disparity [21], [22].

However, finding corresponding pixels may prove difficult when viewing surfaces without texture, e.g. a white wall. The stereo matching algorithm tries to match a white pixel inside the left image with its corresponding white pixel inside the right image. In the case of a white wall this would mean that a lot of pixels are white and thus not distinguishable from each other, which results in so-called dropouts inside the 3D image, i.e. areas inside the 3D image which cannot be stereo matched [23].

The D435i uses an IR projector to tackle this problem. It projects a specific IR pattern into the image. This so-called structured light (SL) improves the 3D image quality significantly by providing the matching algorithm with a known texture to easier find corresponding points [23].

3.3 Software Setup

The prototype runs on Ubuntu Linux 16.04 and uses ROS as its main framework. While the pre-processing of the point clouds is conducted with the Point Cloud Library (PCL) [9], the conversion into voxel grids is realized with OctoMap. The visualization is realized with Rviz [24] and MoveIt [25] which both are introduced in Section 3.3.1.

3.3.1 Robot Operating System

ROS is an open source framework for robotics and one of the most widely used systems not only in research but also in commercial industrial applications [18], [26]. It is a collection of useful tools like visualization and handling of sensor data, e.g. from 3D cameras. Further functionalities amongst others are the geometric and kinematic description of robots (Sections 2.6 and 3.3.5), 3D mapping (Section 2.5) and path planning, which means reasoning how a robot can find a path in its environment to get to a specific goal.

Conceptually, ROS is composed of nodes, topics and messages. Figure 8 shows different

interconnected nodes, which communicate via topics, services and actions. A node is simply a program which executes a specific task (e.g. controlling the motors) and can communicate with other nodes to realize a modular system for robot controlling. This communication happens via so-called topics where a node can publish a message and other nodes can subscribe onto in order to receive the messages. Moreover, a node can offer its functions to others as services (synchronous) and actions (asynchronous). A node can then make a call to invoke those functions. Such a communication infrastructure enables a modular software design for different kinds of robots, e.g. flying drones, humanoid robots or wheeled robots. Every part of the robot can be included into ROS by simply creating a node, regardless if it's an extra sensor or a third hand.

ROS supports programming languages like C++, Python and Java, is free, open-source and has a large community. Moreover, all software packages used in this thesis are parts of ROS. This leads to a significant time saving in the development since all parts can easily be connected with each other and ROS delivers almost every needed software part for this thesis.

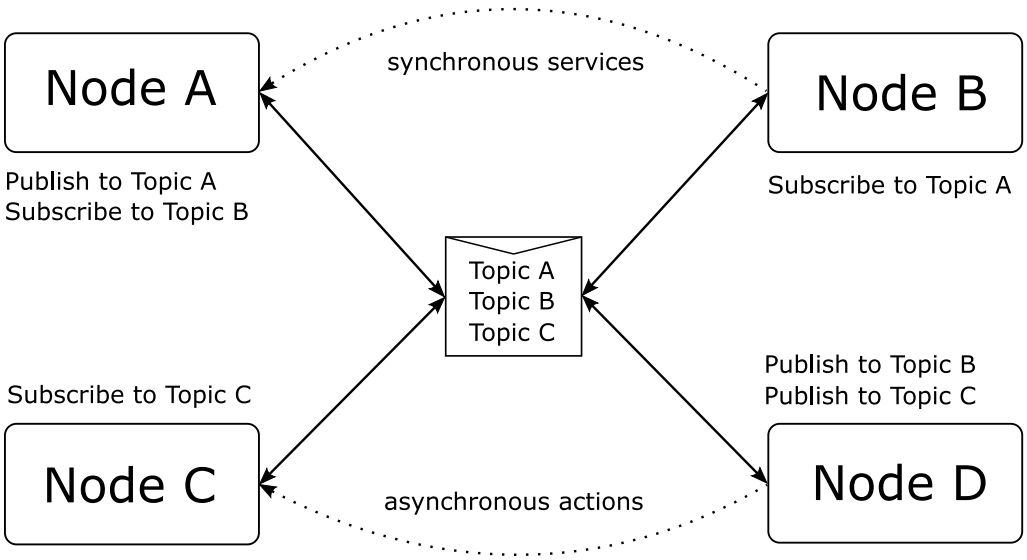

Figure 8: ROS nodes communication over topics, adapted from [27]

3.3.2 MoveIt and Rviz

MoveIt is open-source and the main tool used for robot path/motion planning in ROS [25]. It provides useful functionality needed for this thesis, such as 3D perception. In addition, MoveIt offers plugins for Rviz and uses Unified Robot Description Format (URDF) for describing robots and their motion behavior. Figure 9 shows an overview of the archi-

tecture of MoveIt. The Move Group is MoveIt's main node and communicates via ROS topics, actions and services (Figure 8).

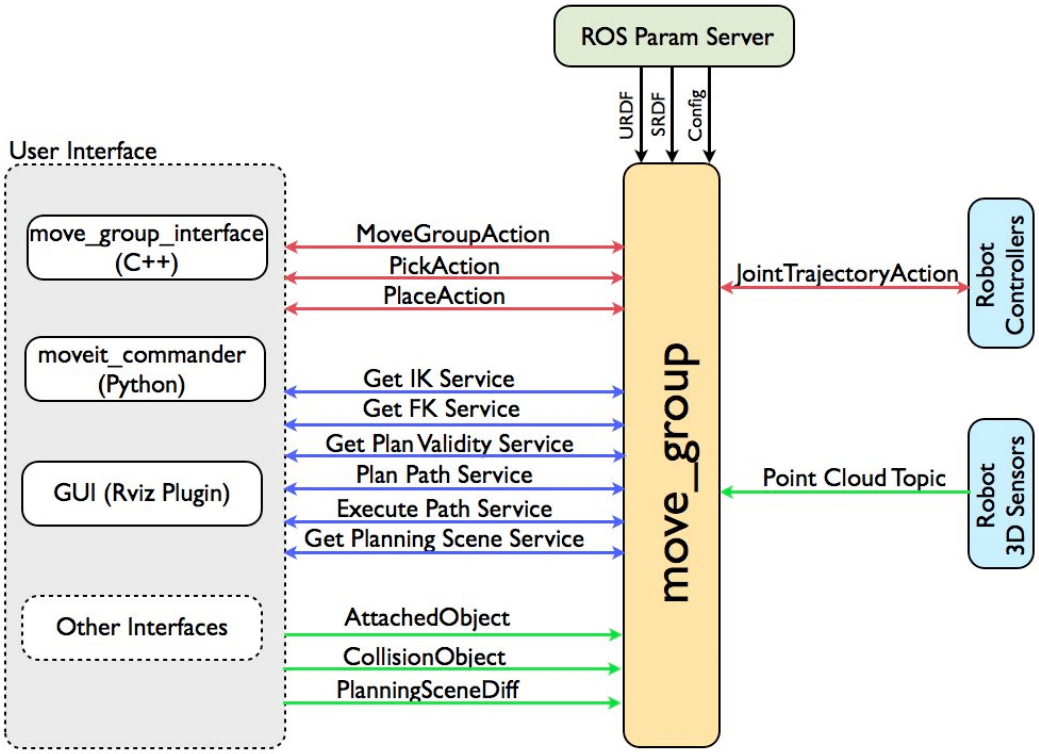

Figure 9: The MoveIt System Architecture [25]. Important parts for this thesis are the Rviz plugin file, the handling of robot 3D sensors and the use of robot descriptions with URDF. Topics are marked green, services blue and actions red.

MoveIt represents the environment of the robot, and the robot itself with its so-called Planning Scene, which is maintained by the Planning Scene Monitor. This monitor listens to different ROS topics to get information about the current states of the robot's joints and sensor data of the environment. The 3D camera's video is streamed via ROS point cloud topics, where MoveIt ties in and handles this data with its Occupancy Map Monitor, which maintains the 3D perception of the environment using OctoMap [25].

Different user interfaces are provided in MoveIt. Programming languages like C++ and Python, or a Graphical User Interface (GUI) can be used. The prototype uses Rviz, since MoveIt already provides a plugin for it.

Rviz is a 3D visualization tool for ROS [24]. It provides a graphical user interface to visualize different types of robots an sensor data. Rviz listens to the topics of MoveIt to visualize the Cobot and the HRC workspace. Furthermore, the robot motion planning can be controlled via Rviz, thus using MoveIt's services and actions to send commands to MoveIt and its motion planners (Figure 9) to ultimately control the Cobot.

3.3.3 Point Cloud Library

The PCL [9] is a programming library for handling 3D data. It is a C++ template library and includes algorithms for registration, segmentation, feature extraction and filtering of point clouds. Different filtering algorithms are used in this work, such as the PT-Filter and SOR-Filter. Moreover, the PCL provides tools for visualization of point clouds, which are also used in this thesis. The PCL is fully integrated into ROS, meaning that it provides classes and functions to subscribe to 3D data from ROS topics and also to publish the modified data again.

3.3.4 OctoMap

Hornung et. al. [15] proposed with OctoMap a method to tackle the most important problems of robot 3D mapping as stated in Section 2.5. OctoMap is technically a grid map (Figure 2), thus dividing the overall volume of the 3D representation into cubic volumes (voxels) instead of points. OctoMap stores the voxel grid inside an octree, which is a hierarchical tree structure where a node can have from 0 to 8 child nodes. Figure 10 shows, that the overall volume is recursively divided into sub-volumes. Each (sub)-volume is therefore divided into 0 to 8 sub-volumes and so on, until a specific given minimum size is reached. This minimum size determines the resolution of the voxel grid and can be changed for different areas of the 3D representation, e.g. a higher resolution on the desk to recognize smaller objects and a lower resolution at the bottom of the room to save memory but still recognize obstacles.

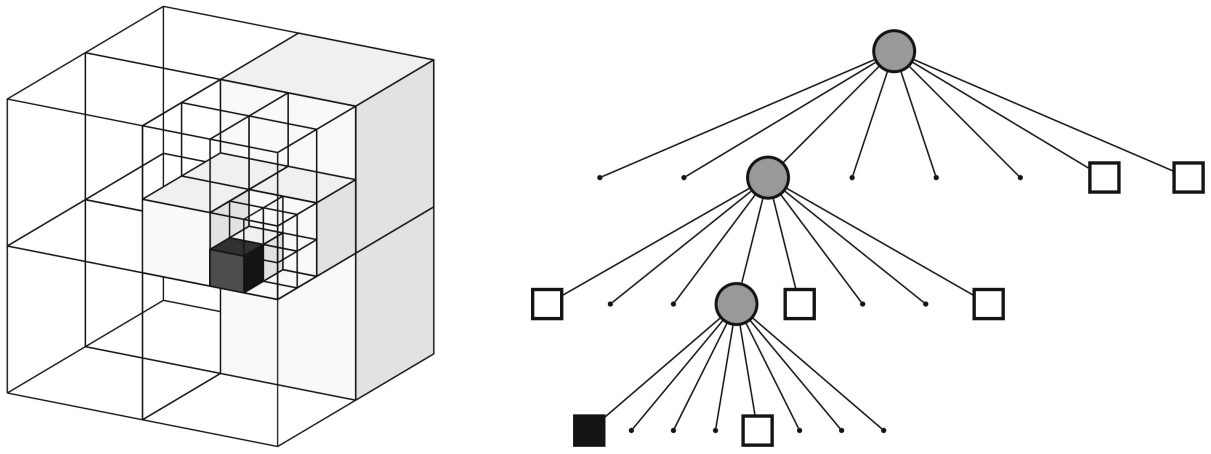

Figure 10: Representation of the 3D environment as an octree voxel grid. The left image shows the voxels with their corresponding covered (sub-) volumes. On the right side is the related octree structure where the data is stored. [15]

3.3.5 Unified Robot Description Format

ROS offers a tool to describe kinematic and geometric properties of a 3D robot model. The URDF is the native format for describing robots in ROS [28]. Basically, URDF includes information in a XML-style syntax. Besides the robot's name, the links, joints and how they are connected via parent-child relationships are described. Links include a visible geometric model and a collision model. While the former is used for visualization, the latter is used for collision detection with both, the robot's own parts and objects in the robot's environment. The joints need parameters for their rotation axis, their limits and the links they connect, such that the virtual robot has the exact moving constraints as the real robot. Listing 12 and Figure 11 each show a section of the URDF as code and visualized.

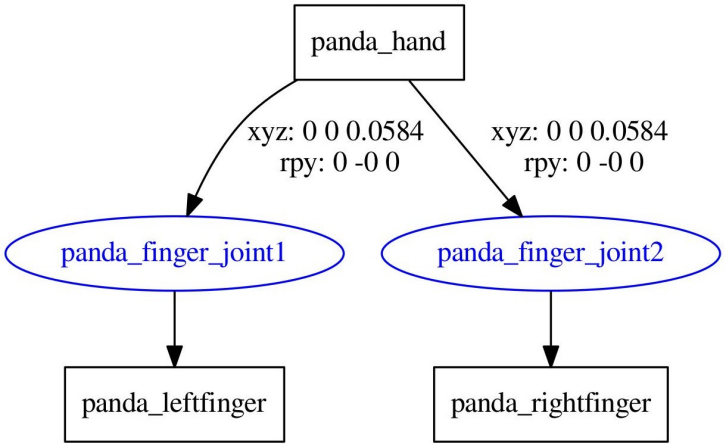

Figure 11: Visualization of a part of the Franka Panda URDF file.

```
1     <joint name="panda_finger_joint2" type="prismatic">
2         <parent link="panda_hand" />
3         <child link="panda_rightfinger" />
4         <axis xyz="0 -1 0" />
5         ...
6     </joint>
```

Figure 12: Part of a the Franka Panda URDF file showing one of the Cobot's joints.

4 Methodical Approach

The approach to create and evaluate the prototype is described in the following Sections. Point cloud pre-processing (Section 4.1) and conversion into OctoMap (Section 4.2) are the first steps to be realized. Their impact on both the frame rate and the memory consumption is analyzed as well as the impact of different OctoMap resolutions. Measuring the accuracy of both steps (Section 4.3) gives clues about their suitability for the prototype. Section 4.5 introduces the tools for the visualization of the workspace and evaluates whether they are comprehensible to use and whether they deliver the necessary functionality for the prototype. The last step is the RS-Filter in Section 4.4 with a subsequent analysis of its efficiency. In summary, in this Section, the previous mentioned requirements of a HRC prototype are measured and their results are presented afterwards in Section 5.

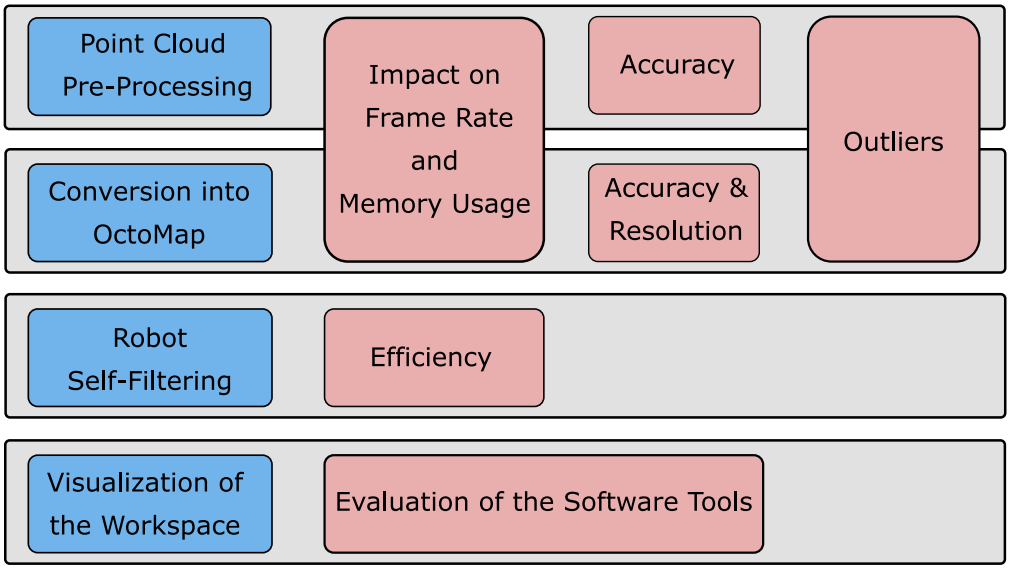

Figure 13: Overview of the methodical approach of this thesis. The blue rectangles are the concrete steps to realize the prototype (Figure 4. The red rectangles represent the evaluation steps of each prototype step.

4.1 Point Cloud Pre-Processing

Before converting the point clouds into an OctoMap, they are pre-processed with the PT-Filter and the SOR-Filter. The node is named */pointcloud_preprocessing* and it retrieves its data from the ROS topics coming from the two 3D cameras, */cam_1/depth/color/points* and */cam_2/depth/color/points*. Figure 14 gives an overview of the ROS nodes and topics.

The PT-Filter is set to cut off the first Z_{min}=0.5 m and Z_{max}=2.4 m of the point cloud. The z-axis is the axis, which points from the center of the camera (sensor) to the point at the desk where the image center lies. These values are set because the cameras are mounted on the ceiling and up to a distance of 0.5 m there are not any obstacles. The value of 2.4 m cuts the point clouds right after the end of the desk. With 2.4 m the whole desk is captured, while the floor and walls of the room are cut out of the image. In addition, a SOR-Filter is also applied and the filter's standard parameters as defined in the PCL [9] are used for this approach.

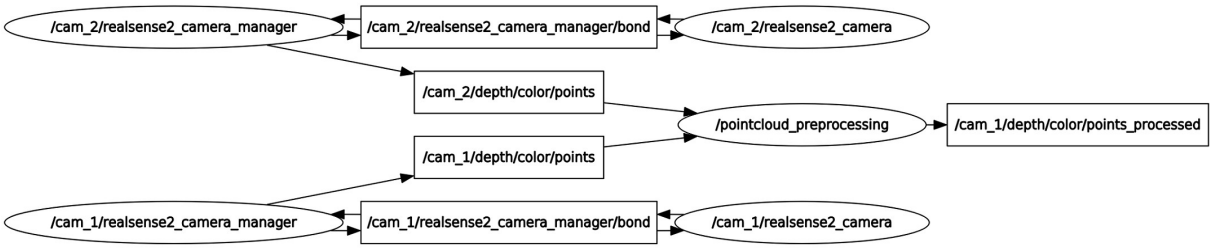

Figure 14: ROS topic graph showing the pre-processing node. The two 3D cameras are visible as (oval) nodes. They each stream their point clouds via their corresponding */cam/depth/color/-points* topics (rectangular). The */pointcloud_preprocessing* node subscribes to this topic, filters the point clouds and then publishes them to new topic */cam/depth/color/points_processed*.

The processed point clouds are afterwards visualized with Rviz. Moreover, the impact of the filter algorithms on the frame rate and memory consumption is measured with *rostopic*, one of ROS' analysis tools. To determine the memory consumption, five frames of the point clouds and voxel grids are extracted from their topics into a file. Afterwards the average file size is determined and compared.

4.2 Conversion into OctoMap

After the pre-processing of the point clouds, they are converted into an OctoMap. This should reduce the memory consumption of the 3D perception. This step determines how much the memory consumption is reduced by comparing the size of a point cloud frame with the size of an OctoMap frame. The conversion to OctoMaps also means computational effort, therefore the impact on the frame rate during the conversion is compared by analyzing the publish rate of the corresponding ROS topics.

In addition, OctoMap offers different parameters to set and optimize. First the impact on memory consumption and frame rate of the voxel size is examined. Second, pixel sub sampling is used to optimize the frame rate. When OctoMap's pixel sub sampling parameter is set to a value n only every n-th pixel is considered for OctoMap conversion, which can raise the frame rate by reducing the computational effort. Different values are set and then the impact on the frame rate is examined.

4.3 Accuracy

It is important for the prototype to get information about how accurate the point clouds and the OctoMaps are. If the deviation from the real world is too high, the Cobot could move into obstacles, thinking the object is further away or in a slightly shifted location.

The accuracy of the prototype is determined in three steps. First the ground truth is measured with a Bosch GLM 40 laser rangefinder with an accuracy of ±1.5 mm [29]. Second the deviation of the point clouds is determined. Third, the deviation of the OctoMap is calculated.

4.3.1 Accuracy of the Point Clouds

Ground truth is a wooden board with dimension of 431.0 mm · 735.1 mm. The board is measured with the laser rangefinder and afterwards a frame of the 3D point clouds is captured and analyzed with CloudCompare [30]. In CloudCompare, the wooden board's dimensions are measured with the built-in measurement tool.

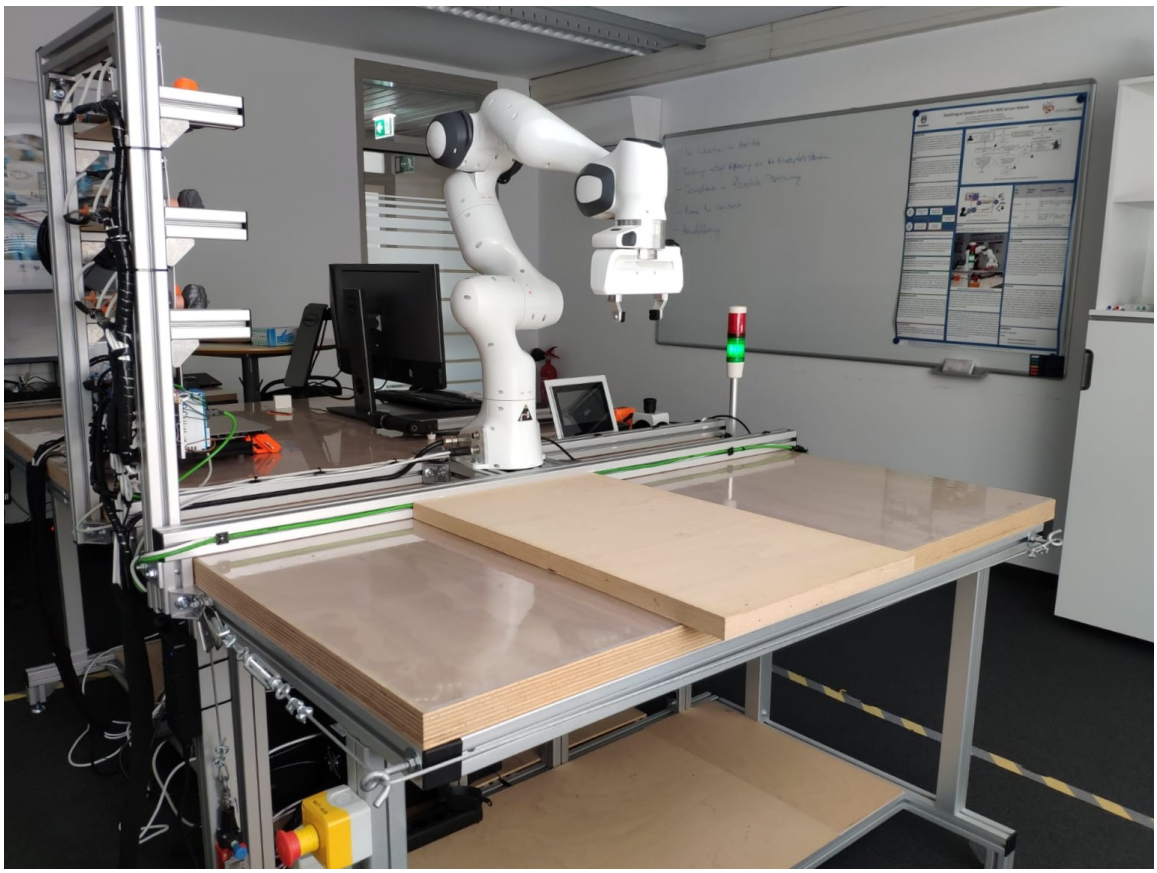

Figure 15: The wooden board which is used for a comparison between the real board and its point cloud counterpart. The dimensions of the real board are measured with the laser rangefinder and then compared to the dimensions which are retrieved with CloudCompare

However, the exact measurement points are hard to determine by the point clouds geometric information alone due to sensor noise. Figure 16 shows a point cloud representing part of the work desk. Despite the real work desk being flat, the point cloud shows a wavy surface as a result of sensor noise. Since the Intel D435 3D Camera takes colored images, the measure points are chosen not only by their geometry, but also their color information. This is visualized in Section 5.1.3.

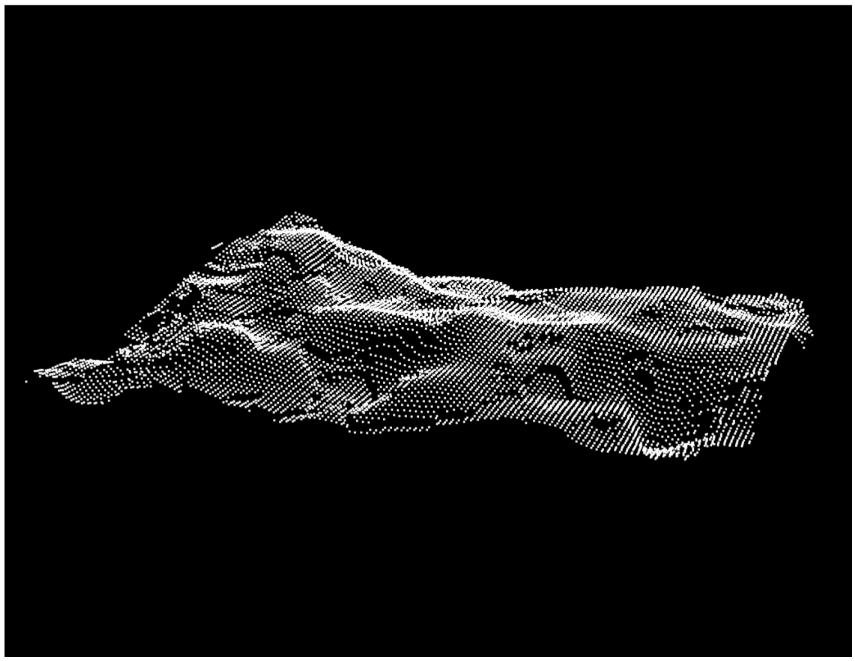

Figure 16: An area of the work desk showing sensor noise. While the real work desk is flat, the point cloud shows a wavy surface.

The above-mentioned sensor noise is measured by cutting out an area of the point cloud showing the work desk since it is a straight surface and thus easy to compare. Both cameras have viewing angles of 45 ° compared to the work desk. However, the cameras are mounted looking at each other, i.e. the axes of their point clouds are mirrored. Both point clouds are statistically analyzed to see what impact the angle has on the noise on each of the cameras. Furthermore, the method of measuring each point cloud's noise is verified by comparing the analysis results of both point clouds.

4.3.2 Accuracy of the OctoMap

Three boxes are used for the ground truth measurement for the OctoMap accuracy. The boxes have dimensions of 10 cm · 10 cm · 10 cm, 20 cm · 20 cm · 20 cm and 30 cm · 20 cm · 20 cm. They are also measured with the laser rangefinder since they are made out of cardboard material. The boxes are placed on the work desk and then captured as 3D point cloud. In Section 4.3.2 the noise of the desk is analyzed and shows a minimum

z-axis value of 1.54 m. Therefore, the point clouds are passthrough filtered exactly at a z-axis value of 1.0 m to 1.53 m. This results in a bounding-box – a box-shaped area inside the point cloud – that only consists of points of the cardboard boxes. Afterwards, the point clouds are converted into OctoMaps and then analyzed.

The volume of the real boxes is calculated and then compared to the volume of the voxels. The necessary voxels to calculate the volume are not only the occupied voxels, but also the not-measured voxels. This is due to the fact, that the area inside the boxes cannot be captured by the 3D cameras, but only the surface. To achieve the real volume, a bounding box – a virtual box-shaped area – is placed over the OctoMap which is exactly as big as the outermost voxels of the box in every direction. Then, the occupied and non-occupied voxels are subtracted by all voxels inside the bounding box to get the number of not-measured voxels. To get the final number of necessary voxels, the number of occupied voxels is added to the number of not-measured voxels.

If one occupied voxel too many is counted at the edge of the object, then automatically, one non-measured voxel less is counted. Since both voxel types are used to represent the whole object, the sum of counted vs. subtracted voxels remains always the same. For the reason that the resolution of every voxel can be determined, the volume can be calculated and compared to the real box volume. In addition to the volume, the dimensions of the object inside the OctoMap are measured by analyzing the edges of the outermost voxels in every axis – which are the same dimensions as the before-mentioned bounding box. The dimensions are then compared to the real boxes to more information about the accuracy.

Knowing the accuracy of the point clouds gives information about what resolution of the OctoMap makes sense. If e.g. sensor noise is too high for specific tasks, a higher resolution of the OctoMap does not make sense, because it would only cost computational power and deliver no benefit. Furthermore, the point clouds are the source material for the OctoMap conversion. If a strong difference in an object's dimension is only measurable at the OctoMap but not the point clouds, there might still be a way to optimize the OctoMap accuracy by using a higher resolution. A strong deviation can happen, when only a few pixels of an object reach beyond an outermost voxel of the same object. However, those pixels are enough for OctoMap to place a voxel inside this area. A lower resolution then means that more space is perceived as occupied, because a larger voxel is used for those few pixels.

4.3.3 Outliers

The problem of outliers described in the foundations is tackled during point cloud pre-processing with the SOR-Filter. However, the computational demand of this filter can be

too high to realize the needed latency of the prototype. Therefore, besides the SOR-Filter, the impact of the *point_subsampling* parameter of OctoMap on outliers is analyzed.

4.4 Robot Self-Filtering

After pre-processing and conversion into OctoMap, the Cobot-model needs to be removed from the OctoMap. The OctoMap only contains collision data and provides the robot with information about its surrounding obstacles. However, the Cobot itself is seen inside the point clouds and thus also inside the OctoMap. This could hinder the Cobot from moving, since itself may constantly be interpreted as an obstacle. This problem can be tackled with a RS-Filter algorithm. The Cobot 3D model is already placed inside the virtual workspace and uses the real Cobot pose information provided by the robot hardware. The voxels consisting of Cobot parts are then filtered out by the algorithm [31], [32].

To measure the efficiency of the RS-Filter the OctoMap is filtered with a PT-Filter around the Cobot. Only a box-shaped area around the Cobot is checked for points. The height of this box is from the bottom of the Cobot model until up to 0.2 m above it. The width is roughly 0.2 m side wards from the robot. These limits ensure that every possible robot-voxel position is considered. Even when a voxel of 10 cm size emerges at one of the edges of the robot, it should be taken into account for the measurement. If only one point can cause a voxel in a specific position, even when the point is on the outermost edge of the voxel. Five frames are extracted and the remaining Cobot-voxels are counted.

4.5 Visualization of the Workspace

The filtered and pre-processed OctoMap together with a Cobot 3D model need to be visualized to use MoveIt's graphical path-planning functions. MoveIt is the main path-planning tool for ROS and offers a Plugin for Rviz, which can visualize different sensor data like voxel grids as well as robot models. Therefore, Rviz is used for the visualization of the HRC workspace.

The first step is to add the 3D Cobot model into the virtual environment. Franka [19] already provides a 3D model of their Cobot via the URDF (Section 3.3.5). The 3D model is loaded into MoveIt/Rviz and placed into the middle of the workspace.

The second step deals with the 3D perception of the environment. MoveIt offers a so called PointCloud Occupancy Map Updater, which reads the exact same format which the 3D-camera streams (*sensor_msgs/PointCloud2*) and uses OctoMap to convert them into a occupancy map/voxel grid. Afterwards the Occupancy Map Updater publishes the voxel grid to the Planning Scene, which in addition contains the 3D model of the Cobot.

The Planning Scene contains every information of the workspace to enable collision-aware path-planning.

The voxel grids of the 3D cameras are already aligned, but they are not aligned with the Cobot model. Thus, the third step is to transform the voxel grids such that the Cobot model is aligned with the Cobot from the voxel grid. However, the Cobot can be more easily recognized inside the point clouds, which thus are transformed before they are converted into the voxel grid. The information about the transformation remains unchanged and therefore is also applied to the voxel grid. ROS offers its own tool for transformations, the tf package [33]. With this tool, the voxel grids/point clouds are manually transformed.

Having described our methodology we will next present and discuss the results of the experiments with the prototype.

5 Results and Discussion

This Section shows the results of the methodical approach in Section 4 with the computational platform of Section 3. Section 5.1 focuses on the quality of the point clouds and methods to improve and measure it. Section 5.2 shows the evaluation of the OctoMap conversion process and the reachable accuracy and resolution, which is important to know whether the prototype is suited for fine-granular tasks such as recognition and handling of smaller objects. The frame rate and memory consumption are evaluated during preprocessing of the point clouds and the OctoMap conversion (Sections 5.1 and 5.2). Further important indicators of the accuracy are the removal of outliers inside the point clouds or voxel grid and the removal of Cobot-voxels with the RS-Filter in Section 5.3. After those steps, the OctoMap and the Cobot 3D model are visualized and the used tools evaluated.

5.1 Point Cloud Pre-Processing

The effects of the point cloud pre-processing are shown visually. Afterwards the frame rate and memory consumption are analyzed without and during the preprocessing. At last, the accuracy is measured and discussed.

5.1.1 Visual Evaluation

As can be seen in Figure 17, the PT-Filter filter removes the point clouds some centimeters beyond the desk's edges. This value need not be perfectly accurate, because the only constraint is that no parts of the desk are removed. This means, the point cloud is filtered at some point beyond the desk's edges. The whole desk is still visible, but the floor and thus a high number of (for this thesis' goals) unnecessary points are removed. How this is affecting the frame rate and memory consumption is discussed in the following Section.

5.1.2 Frame Rate and Memory Consumption

Both 3D cameras stream their point clouds with roughly 30 Hz. The measurements showed (Figure 18), that the PT-Filter has no effect on the frame rate. However, the SOR-Filter reduces the frame rate to roughly 1 Hz. Even when the points are reduced with the PT-Filter filter before applying the SOR-Filter – this means fewer points for the SOR-Filter to iterate through – the frame rate declines sharply. All tests are conducted on ROS-Kinetic (Ubuntu 16.04) and PCL 1.11.99 and are running on the system described in Section 3.2.1. The frame rate and memory consumption are measured with the tool

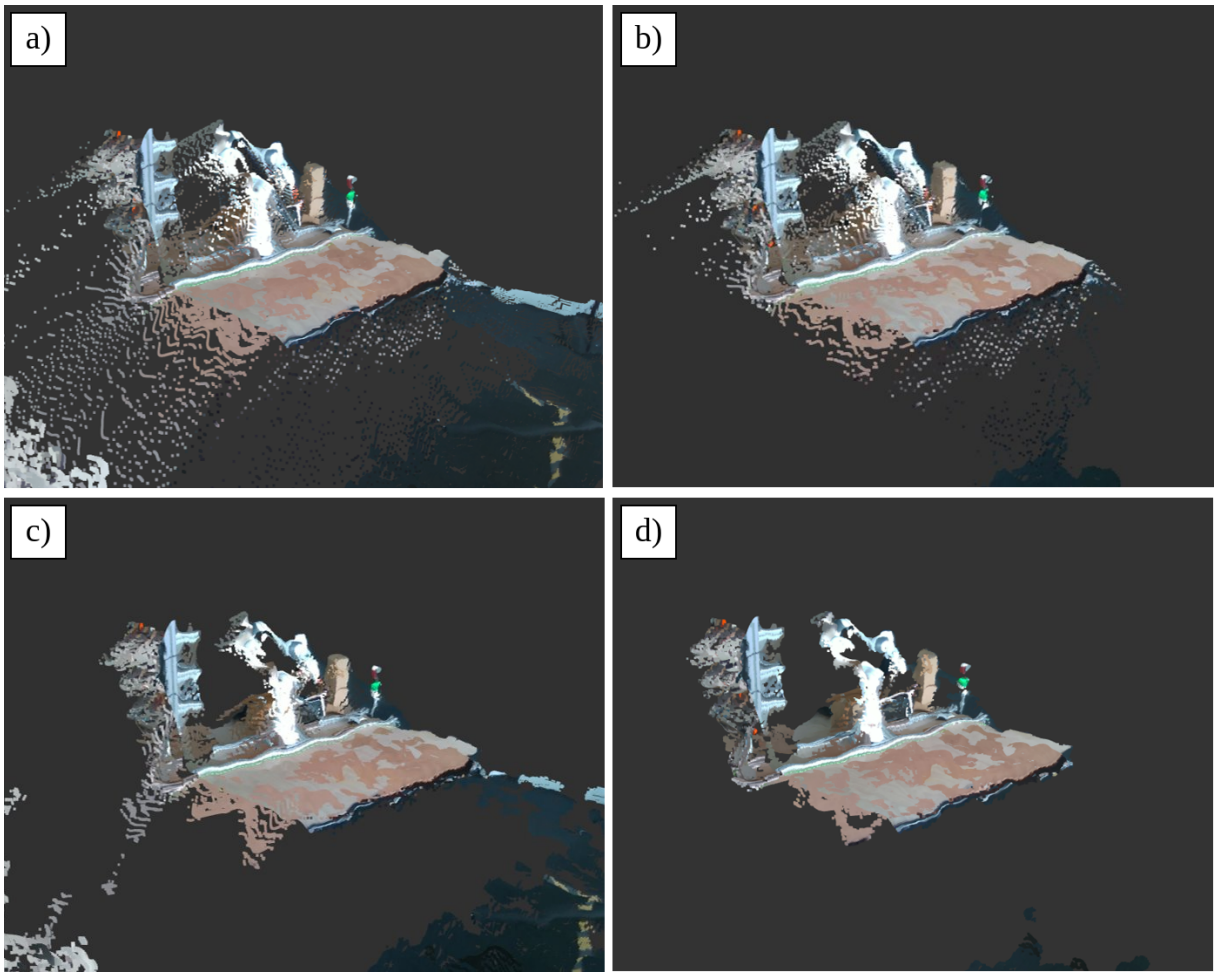

Figure 17: Comparison of unfiltered and passthrough filtered point clouds. a) Original unfiltered point clouds. b) Point clouds after a PT-Filter – Z_{min}=0.5 m and Z_{max}=2.4 m – is applied. c) SOR-Filter only. d) Both a SOR-Filter and a PT-Filter are applied.

rostopic, which can analyze ROS-topics. After subscribing to the chosen camera topics, rostopic directly prints out all necessary information needed for this work.

While the PT-Filter efficiently reduces the number of points, the PCL's SOR-Filter is unsuitable for the prototype's requirements. The frame rate analysis with rostopic shows that the frame rate f sinks from around 29.8 Hz to only 1.15 Hz. Calculating the time duration of one frame (period) $T = \frac{1}{f}$ yields the latency, which rises from 33.6 ms to 885 ms when filtering the clouds with the SOR-Filter. Such reaction times of almost a second can be too slow for the prototype to react fast enough to moving obstacles such as humans.

Removing a high number of points from the point clouds has an immediate impact on the memory consumption (Figure 19). The PT-Filter reduces the memory consumption by nearly 26 % and the SOR-Filter by 12 %. Applying both filters results in a reduction of almost 31 %.

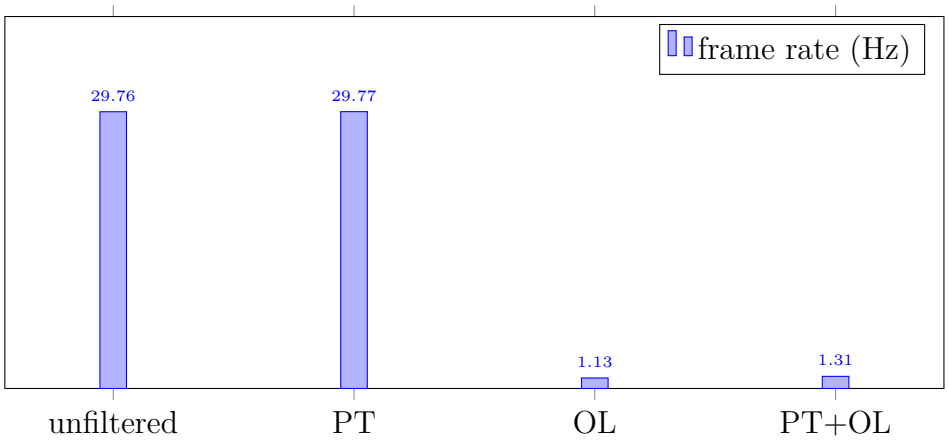

Figure 18: Frame rate comparison of the point cloud pre-processing. Values retrieved with rostopic on ROS-Kinetic (Ubuntu 16.04), PCL 1.11.0, i7-8700K, 32 GB RAM.

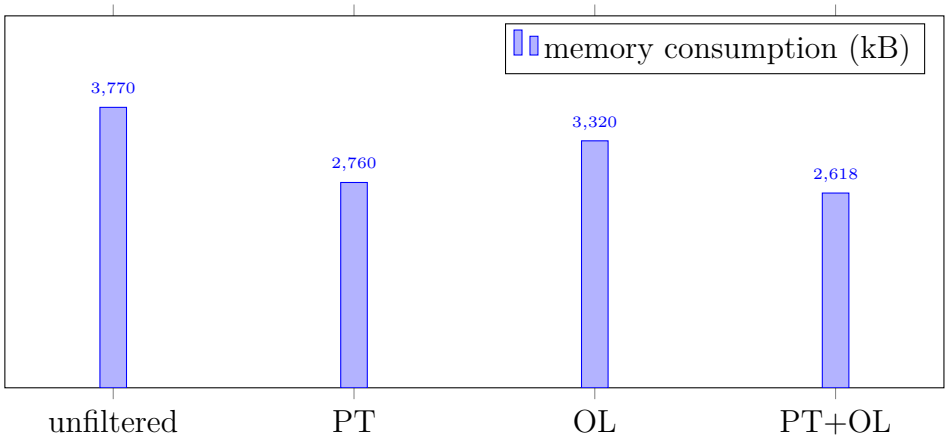

Figure 19: Memory consumption comparison of the point cloud processing. Values retrieved with rostopic on ROS-Kinetic (Ubuntu 16.04), PCL 1.11.0, i7-8700K, 32 GB RAM.

5.1.3 Accuracy

The real-world dimensions of the wooden board are 431.0 mm · 735.1 mm. Measuring the same board's width in the point cloud in CloudCompare (Figure 20) results in 431.7 mm which corresponds to a deviation of under 0.1 %. The measured length is 720.7 mm – a deviation of approximately 2 %.

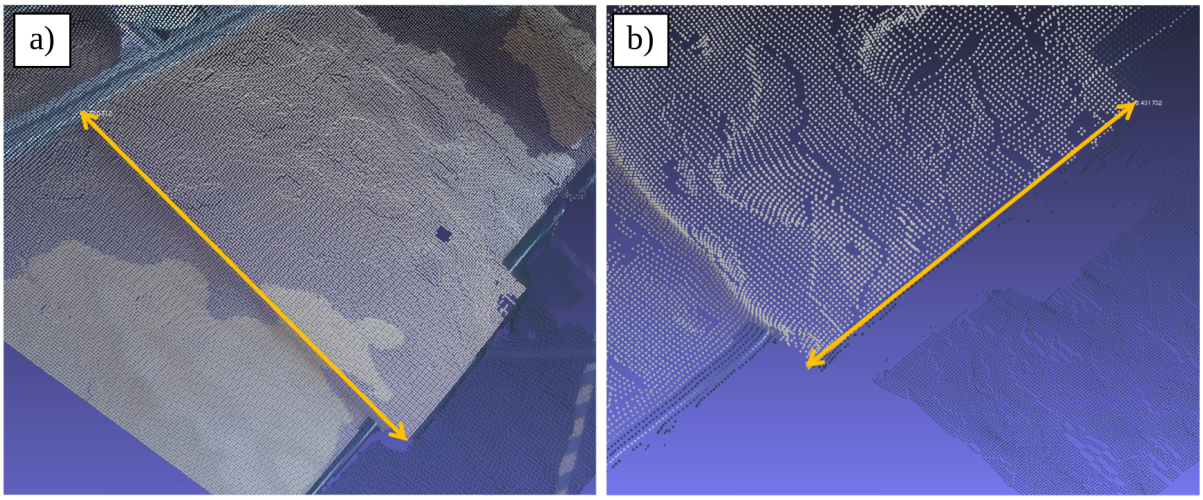

Figure 20: Measuring the dimensions of a wooden board on the desk with CloudCompare. a) Width is 431.7 mm. b) Length is 720.7 mm.

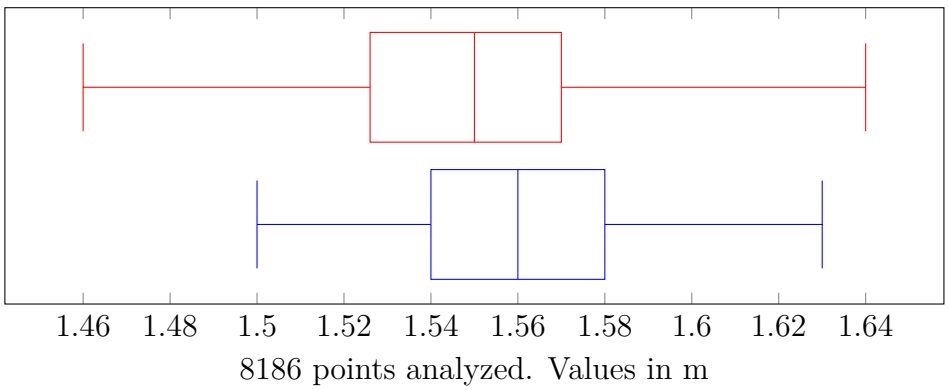

Figure 21: Noise analysis of the point cloud showed as box plot. Both plots show the same work desk from a 180 ° rotated view. The standard deviation is 0.02 m, or 0.03 m, and the mean is 1.65 m or 1.57 m.

An area of 0.8 m · 1.1 m of the desk is captured and the z-dimension (height) of the points analyzed to get information about the strength of the sensor noise. (Figure 22 and 21). It is shown, that the median is at about 1.56 m and most of the points lie in the interval from 1.54 m to 1.58 m. However, there are points, that are located at 1.5 m or 1.63 m, which means a deviation from the mean (1.57 m) from 6 cm to 7 cm. Thus, smaller objects are not distinguishable from the desk, which makes the used 3D cameras unsuitable for fine-granular tasks such as grasping small objects with the Cobot gripper.

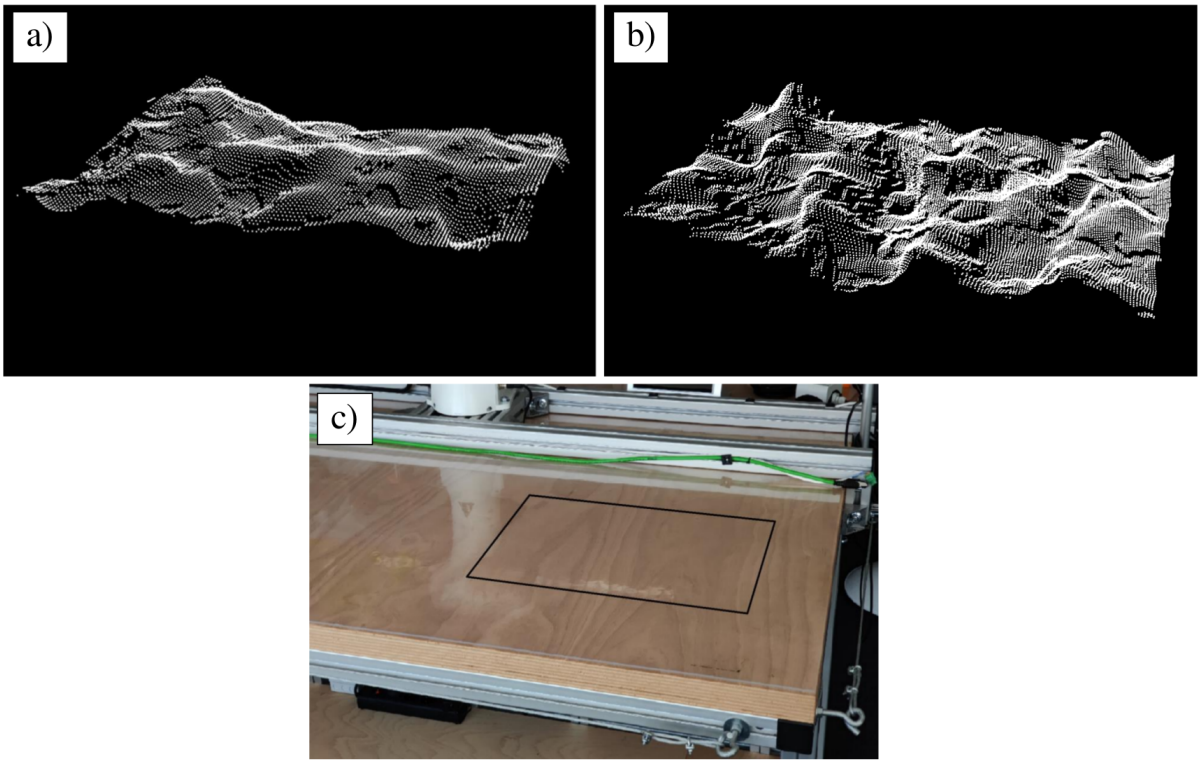

Figure 22: The analyzed area of the work desk. Size is 0.8 m · 1.1 m. a) and b) show the point cloud of the corresponding desk area from different points of view. (with mirrored axes, i.e. rotated by 180 °). c) shows the real work desk.

5.2 Conversion into OctoMap

After the point cloud pre-processing, the conversion into an OctoMap follows. The prototype later uses the OctoMap for collision-detection. The process of OctoMap conversion is visually evaluated and the impacts on the frame rate and memory consumption are analyzed. Both values are important for the suitability of the prototype for tasks such as HRC. At the end of this section the accuracy and resolution of the OctoMap are analyzed and discussed.

5.2.1 Evaluation

To avoid a further reduction of the frame rate, a minimum voxel size of 5 cm was chosen. In Figure 23 a), the voxel size is 5 cm and the outlines of the desk and the shelf are recognizable. When choosing a voxel size of 10 cm (Figure 23 b)) the shelf and desk are recognizable as obstacles, but the level of detail is highly reduced.

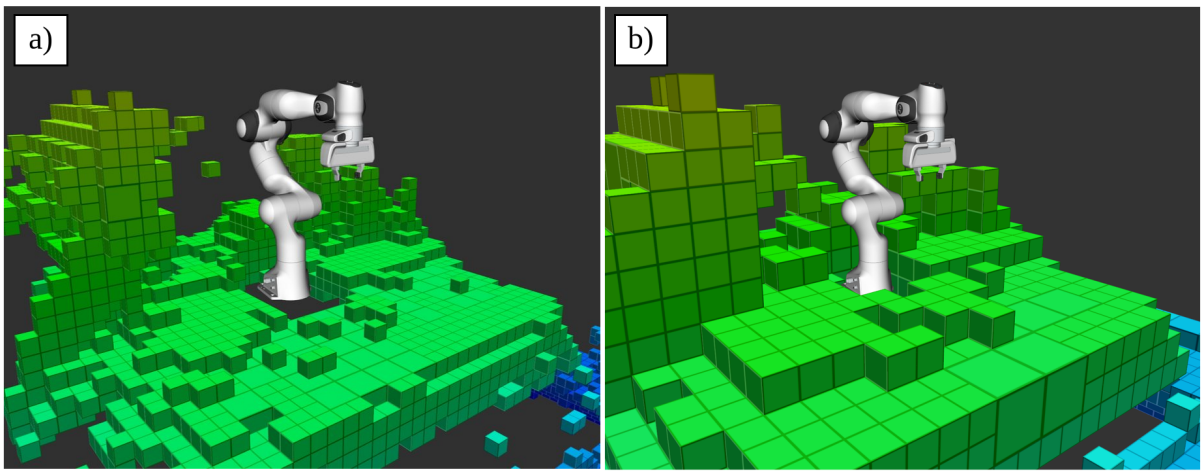

Figure 23: Comparison of OctoMap with different leaf sizes. a) 5 cm and b) 10 cm.

5.2.2 Frame Rate and Memory Consumption

Converting the point clouds into an OctoMap showed a significant reduction in memory consumption (Figure 25). The unfiltered cloud is reduced by about 96 % and the OctoMap has an average of only about 140 kB in size when a voxel size of 5 cm is used. This reduction increases to 99 % when choosing a voxel size of 10 cm.

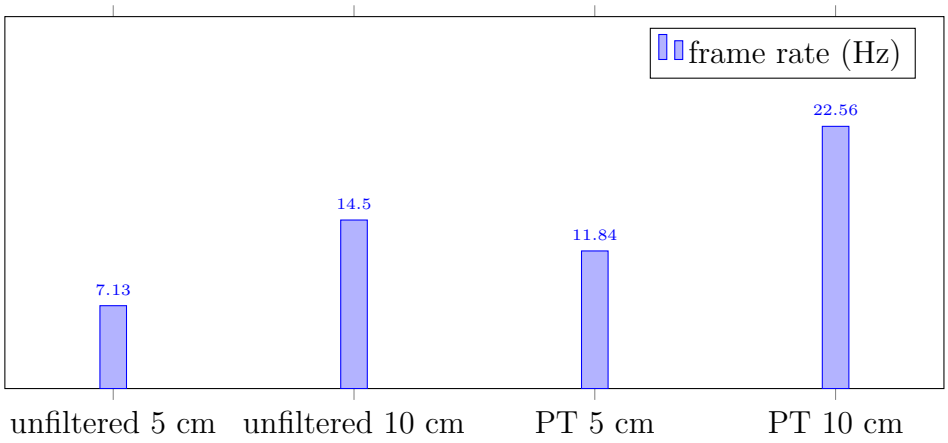

Figure 24: Frame rate comparison of OctoMap with different voxel sizes. Values retrieved with rostopic on ROS-Kinetic (Ubuntu 16.04), OctoMap 1.9.5, i7-8700K, 32 GB RAM.

With the memory consumption decreasing, the frame rate is rising with a bigger voxel size (Figure 24. However, with a voxel size of 5 cm a maximum frame rate of roughly 12 Hz is achievable when applying a PT-Filter before OctoMap conversion. This frame rate results in a latency of around 83 ms. The memory consumption and frame rate are again measured with the tool rostopic and the tests are conducted on the system described in Section 3.2.1. The conversion of the point clouds is carried out with OctoMap 1.9.5.

The second parameter to examine is the point sub sampling of OctoMap. Figure 26 shows the effect of different parameters of *point_subsampling*. Originally, with a voxel size of

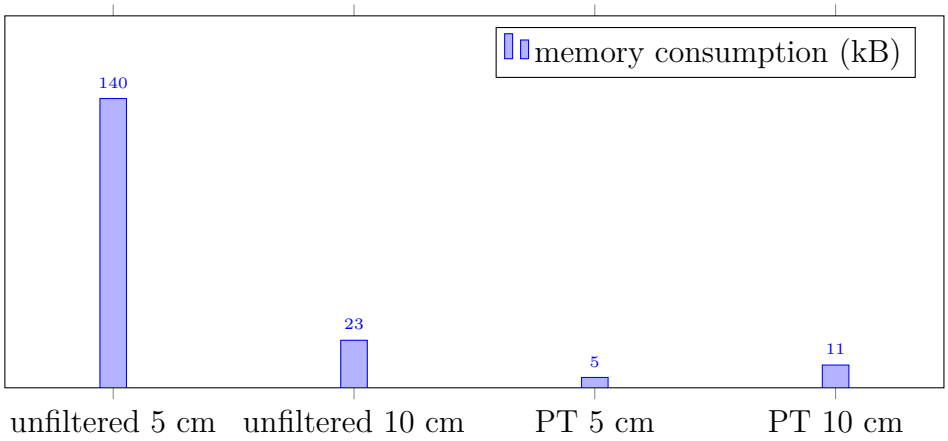

Figure 25: Memory consumption comparison of OctoMap with different voxel sizes. Values retrieved with rostopic on ROS-Kinetic (Ubuntu 16.04), OctoMap 1.9.5, i7-8700K, 32 GB RAM.

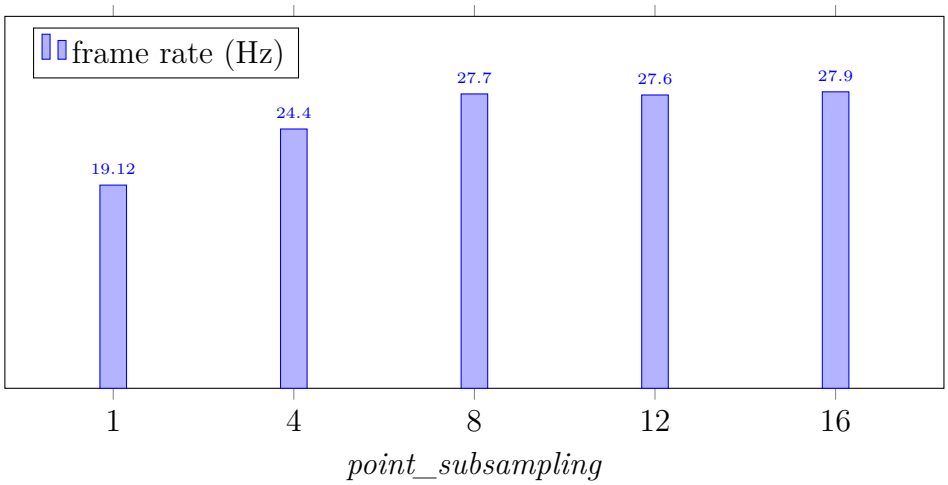

Figure 26: Frame rate comparison when using different values for point sub sampling. Values retrieved with rostopic on ROS-Kinetic (Ubuntu 16.04), OctoMap 1.9.5, i7-8700K, 32 GB RAM.

5 cm, the frame rate is approximately 19 Hz. When changing the parameter to higher values, the frame rate increases up to 27.7 Hz. However, no improvement can be seen when using values higher than 8. Nevertheless, when setting higher values than 4 for point sub sampling, a side effect occured. There are seemingly fewer point outliers in the voxel grid when examining the visualization. This effect is examined further in the next Section.

5.2.3 Accuracy and Resolution

The OctoMap volume shows strong deviations from the real-world. The average out of 5 images for each box results in the volumes as given in figure 28. While the volume deviation for the 10 cm squared box is almost 49 %, this value rises up to 61 % for the 20 cm squared box and 55 % for the box. As seen in Figure 27 this could be due to the fact, that the point cloud behind the object – on the left side of Figure 27.a – has a lot of points which neither belong to the cardboard box nor the desk. They emerge from the imperfect 3D reconstruction method of the 3D cameras on the edge between the cardboard box and the desk behind. These seemingly flying points – in mid-air between the desk and the box – let the number of voxels and therefore the calculated volume rise.

Beside the volume, the dimensions of the boxes inside OctoMap are compared to their real-world counterparts. Figures 29, 30 and 31 show this comparison for each chosen cardboard box. The values are retrieved by taking the coordinates of the outermost voxels in each axis of the box and then calculating the dimensions of the object. The tests show a deviation beginning from 5 cm and up to 15 cm. However, the z-axis shows the smallest deviation in each of the tests. With the voxel boxes being equal or bigger in every axis, the prototype seems still able to react to obstacles without running into them. If the voxel boxes are smaller than their real-world counterpart, the Cobot assumes that the object is smaller and – as a consequence – collide with it.

Not only the volume and the dimensions of objects inside the 3D environment compared to the real world need to be accurate. Also outliers – seemingly flying voxels – remain. They do not belong to any real-world object in the HRC workspace and can be seen in Figure 32.a under the Cobot gripper and above the desk. They emerge from the Cobot but neither really belong to the Cobot nor the work desk. As mentioned in Section 5.1.2 the SOR-Filter showed a great impact on the latency and thus is not suitable for a usable prototype.

However in Section 5.2.2, a side effect of sub-sampling the 3D points at OctoMap-conversion was discovered. The flying points around the robot are reduced at a visual examination in Rviz (Figure 32). Figure 33 shows a significant decline in a box-shaped space of 0.5 m around the Cobot by applying different values for the sub-sampling setting.

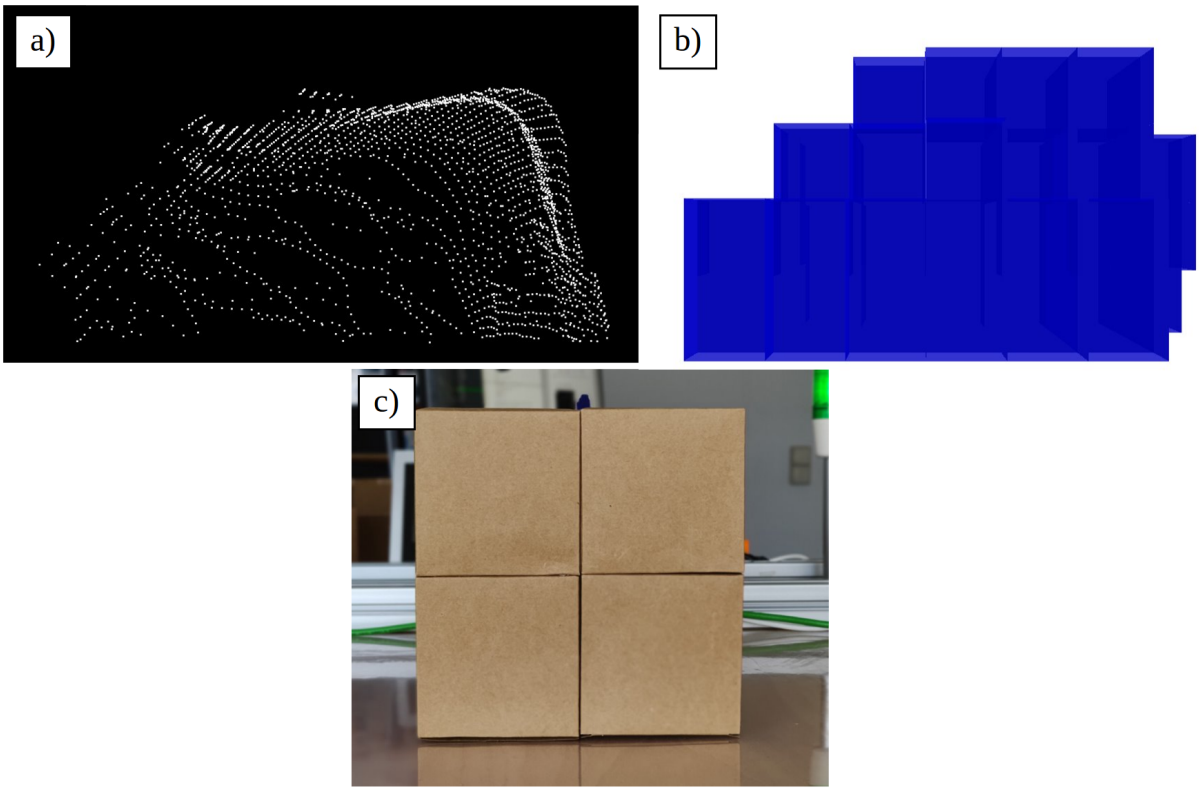

Figure 27: Comparison of a) a point cloud, and b) the same object converted into an OctoMap with a voxel size of 5 cm. c) The real-world object is box-shaped and has a size of roughly 20 cm · 20 cm · 20 cm. It is assembled from four smaller boxes with a size of about 10 cm · 10 cm · 10 cm each.

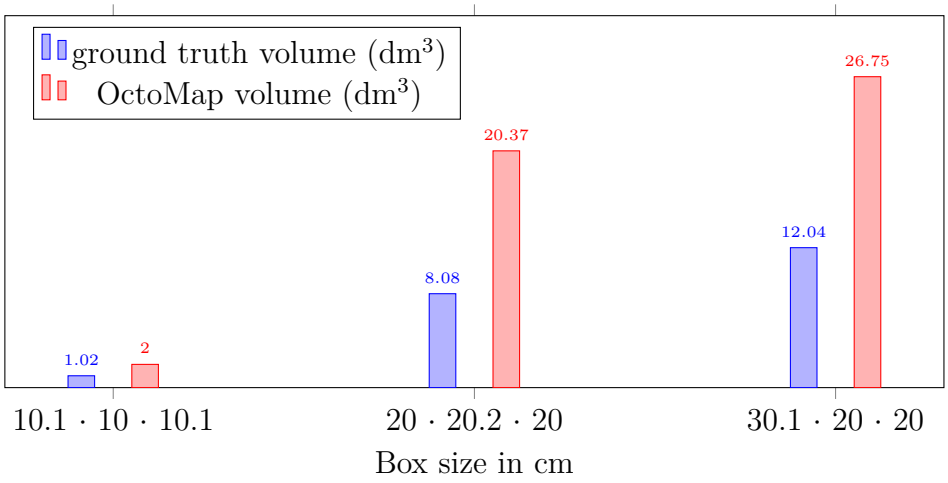

Figure 28: Volume of real-world boxes compared with the volume of their corresponding voxels. Voxel size is 5 cm.

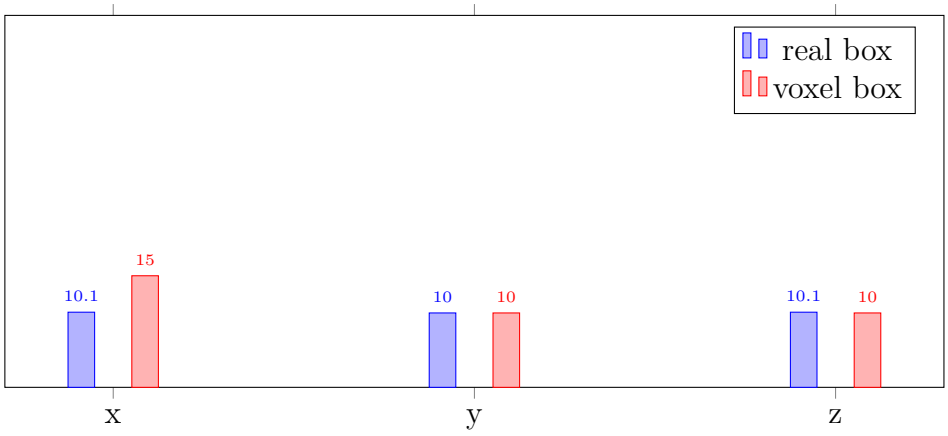

Figure 29: Dimensions of the real-world 10 cm · 10.1 cm · 10 cm box compared with the dimensions of its corresponding voxels inside OctoMap. Voxel size is 5 cm.

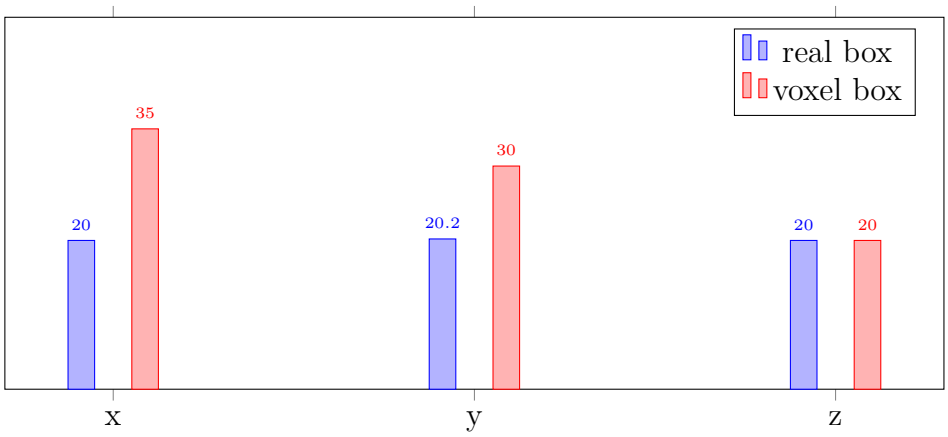

Figure 30: Dimensions of the real-world 20 cm · 20.2 cm · 20 cm box compared with the dimensions of its corresponding voxels inside OctoMap. Voxel size is 5 cm.

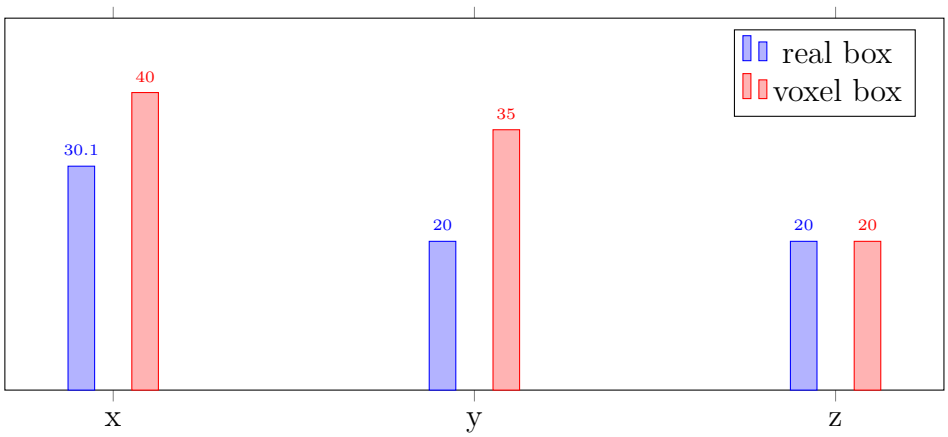

Figure 31: Dimensions of the real-world 30.1 cm · 20 cm · 20 cm box compared with the dimensions of its corresponding voxels inside OctoMap. Voxel size is 5 cm.

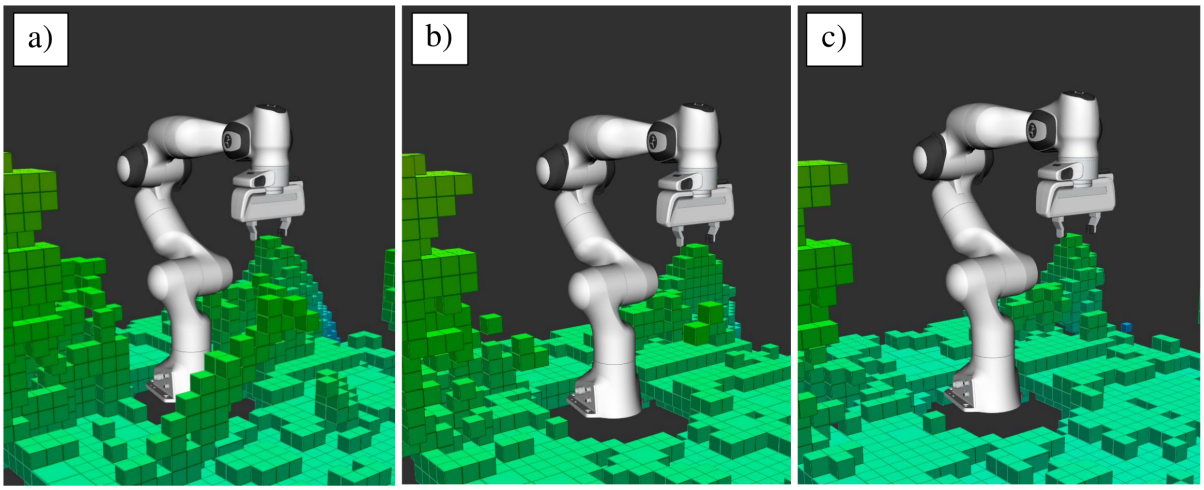

Figure 32: Comparison of different *point_subsampling* values and their impact on the OctoMap outliers. a) Every point is considered. b) Every 8th point is considered. c) Every 16th point is considered.

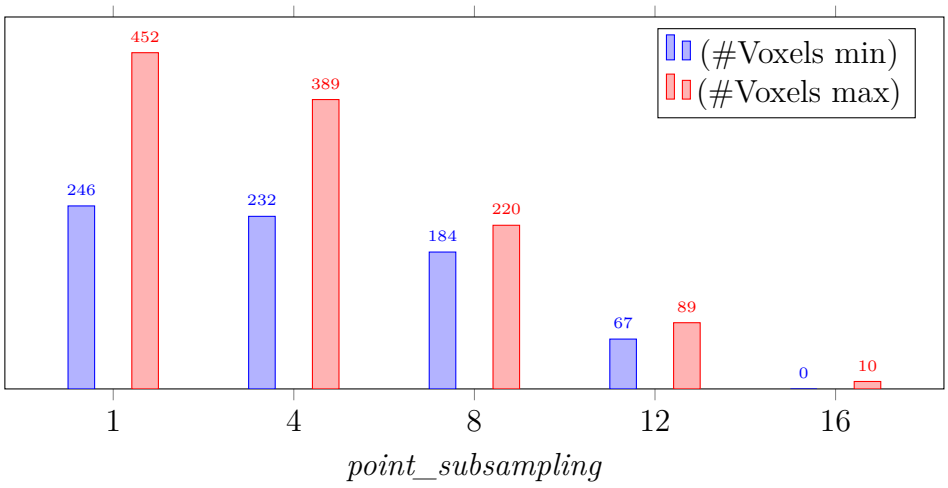

Figure 33: Plot, showing the effect on outlier voxels by sub-sampling 3D points. Values retrieved with rostopic on ROS-Kinetic (Ubuntu 16.04), OctoMap 1.9.5, i7-8700K, 32 GB RAM.

Now that the pre-processing of the point clouds and their conversion into an OctoMap is finished, the next step is to remove the Cobot-voxels from the OctoMap. The efficiency of the RS-Filter is described in the next section.

5.3 Robot-Self Filtering

The RS-Filter is easily configurable via a configuration file provided by MoveIt. Adjusting the padding values leads to a decrease of robot voxels and with the values *padding_scale*=1 and *padding_offset*=0.05 the point was reached to eliminate all disruptive voxels. Figure 34 and 35 show a comparison of the different filter settings.

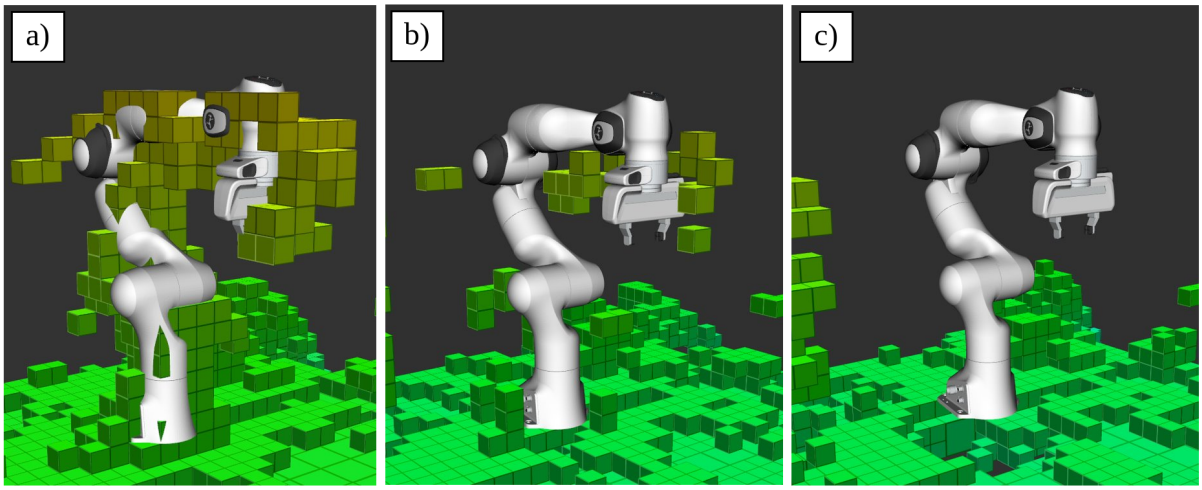

Figure 34: Comparison of different values for the padding parameter of the RS-Filter. a) *padding_scale*=0 b) *padding_scale*=1, b) *padding_scale*=1.1 and c) *padding_scale*=1 and *padding_offset*=0.05

A deactivated RS-Filter shows roughly 350 voxels in the defined area. Turning the *padding_scale* to 1 results in about 44 voxels and Setting *padding_scale* to 1 and *padding_offset* to 0.05 eliminates all disruptive voxels. However, the RS-Filter uses the 3D Cobot model, checks if it overlaps with voxels and then removes those voxels. The aforementioned parameters add a padding around the 3D model, meaning that also voxels are removed which are inside this padding and not only inside the model. An offset value of 0.05 means that there is an extra padding of 5 cm around the model and thus also ignoring possible obstacles inside this area. This leads to the problem that small objects of roughly a size smaller than 5 cm inside the filtered area not fully recognized. Besides the size, those objects must be very close to the robot and therefore human heads should be fully recognized.

The previous Sections describe how the point clouds are pre-processed and then converted into an OctoMap. With the Cobot-voxels filtered out, the prototype is almost finished.

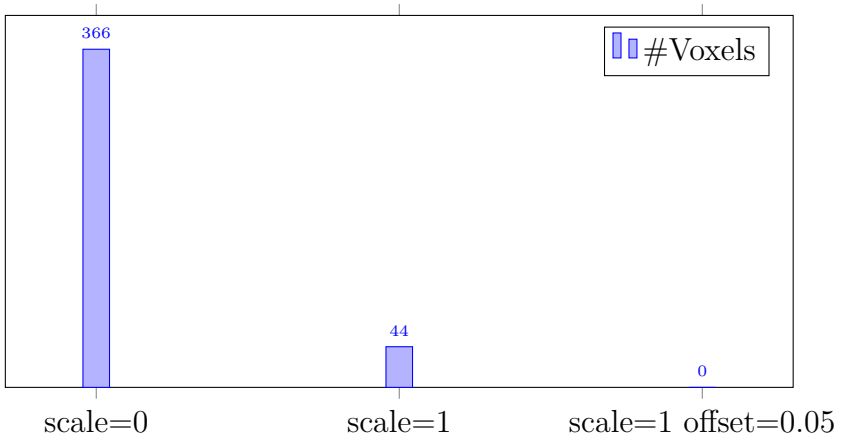

Figure 35: Plot of a RS-Filter with different parameters. The number of voxels decreases with a higher padding scale and offset.

The next step are the visualization of the HRC workspace and the evaluation of the two used tools for this goal.

5.4 Evaluation of Rviz and MoveIt

Using MoveIt and Rviz has two main advantages. First, both flawlessly work together via the provided MoveIt/Rviz plugin. This plugin adds a new user panel into Rviz, where the robot model and the OctoMap can be visualized. Even the robot model can be moved with the mouse inside Rviz, enabling to plan collision-aware robot movement with MoveIt's integrated path-planners. Rviz visualizes not only the aforementioned information, but also the planned movement trajectory before the real robot is controlled. This enables a visual possibility of intervention when testing out new implemented functions.

Second, MoveIt and Rviz share the same modularity that ROS has. This means that specific parts of the prototype can be easily switched to others. For example, changing the 3D sensor or adding new sensors can be achieved by simply publishing the 3D data to MoveIt's occupancy monitor topics.

Furthermore, predefined 3D models can be added to the planning scene, e.g. the desk inside the HRC workspace can be added as 3D plane. The desk then can be filtered out of the point cloud to further improve the frame rate but is still recognized by the robot as obstacle. When path-planning is conducted in MoveIt, instead of a real robot, a simulated robot inside software such as Gazebo [34] can be controlled. In addition, the path-planning algorithms also can be switched.

All in all, MoveIt and Rviz enable the development collision detection and prevention for robots in a modular and comprehensible way. The modularity and the fact, that they are both free to use and open source is especially interesting in prototyping and research.

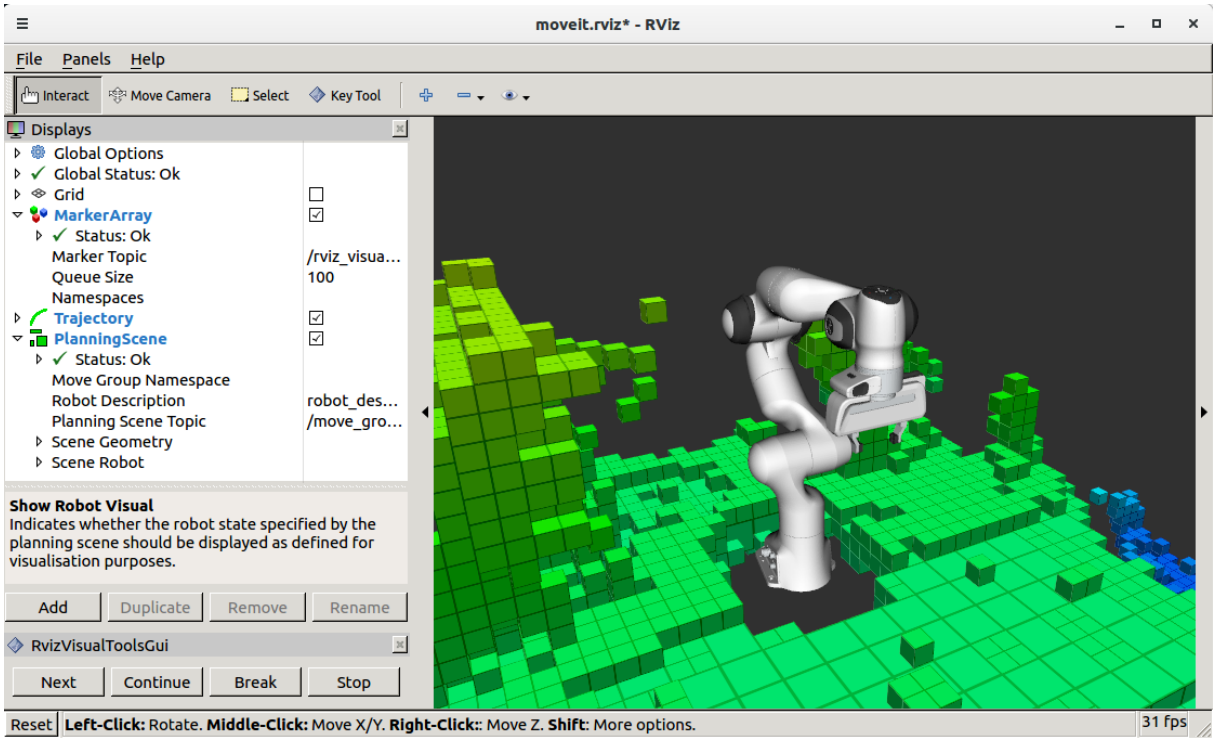

Figure 36: Rviz showing the Cobot model and the environment as OctoMap. On the left side is MoveIt's PlanningScene plugin visible, which maintains the OctoMap, the Cobot model and enables collision-aware path-planning for the Cobot

Another useful function of MoveIt is the Robot-Self Filter which is described in the next Section.

6 Conclusion and future work

The work on the prototype to let a Cobot recognize its environment with 3D cameras started with the pre-processing of the point clouds and their conversion into an OctoMap. Both steps were analyzed in terms of the reachable frame rate, memory consumption and accuracy.

Conversion into OctoMaps reached frame rates of nearly 28 Hz (±0.3 Hz), which translates to a latency of roughly 36 ms (±1 ms). Whether this is low enough can be determined when tests with a moving robot and a moving obstacle inside the HRC workspace are conducted in future work. A problem of the PCL and OctoMap is the fact, that they are running on a CPU. Other work has shown a significantly increase in the efficiency when using a GPU [1], [35]. Especially the SOR-Filter decreases the frame rate to 1 Hz, which is not usable for such a prototype. Implementing those filters on GPUs could help increasing the frame rate and enabling a much finer resolution of the OctoMap.

Besides the frame rate, the memory consumption was evaluated. Applying a PT-Filter to the point clouds already reduces the data by 26 %. Furthermore, the conversion into OctoMap results in a strong decline of nearly 96 % from roughly 3.770 kB per frame to around 140 kB. A relatively small map is used in this work, but when an autonomous robot creates a map of a larger area, the use of OctoMap can save large amounts of resources.

After the evaluation of the frame rate and the memory consumption, the accuracy of the prototype is analyzed. The point clouds of the used 3D cameras showed strong sensor noise of approx. ±7 cm, which makes the prototype unsuitable for fine-grain tasks like the grasping of small objects on the work desk with the Cobot-gripper. Future work can focus on using LIDARs instead of stereo cameras, since they have much less sensor noise and an improved accuracy. It must also be examined what effect the frame rate increasing *point_subsampling* parameter has on the accuracy. When points are skipped, then theoretically objects can be missed, even when objects usually consist of much higher numbers of points than 8 or 16. However, avoiding obstacles is a matter of utmost importance for worker safety in HRC work spaces. Even though the cardboard boxes were shown too large inside the OctoMap – their perceived volumes differed from 49 % to 61 % compared to the real boxes –, they all were recognized as obstacles. The outer edges of the real-world boxes were smaller than those of the voxel boxes. Deviations from 0 cm to 15 cm (0 % to 43 %). were observed. Decreasing the voxel size can potentially adress both problems, but then the frame rate decreases significantly.

The last measurement part is the efficiency of the RS-Filter. Using different filter parameters decrease the Cobot voxels step by step until no more disrupting voxels remain. The

Cobot can move without detecting itself as an obstacle and thus the prototype is ready to plan paths inside the HRC workspace.

At last, the OctoMap and the 3D Cobot model were visualized and both main software tools MoveIt and Rviz were evaluated. In conclusion, it can be said that ROS delivered all necessary software parts for this prototype. The used tools are comprehensible to use, save time and avoid costly reinvention from already researched software. The prototype reached the set goals and is ready to use for the future work on collision aware path-planning.

Bibliography

[1] A. Hermann *et al.*, "Unified GPU Voxel Collibion Detection for Mobile Manipulation Planning", in *IEEE International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 4154–4160.

[2] C. Juelg *et al.*, "Efficient, Collaborative Screw Assembly in a Shared Workspace", in *Intelligent Autonomous Systems*, vol. 15, Springer International Publishing, Dec. 2018, pp. 837–848.

[3] A. Hermann *et al.*, "Anticipate your Surroundings: Predictive CollisionDetection between Dynamic Obstacles and PlannedRobot Trajectories on the GPU", in *European Conference on Mobile Robots (ECMR)*, Sep. 2015, pp. 1–8.

[4] C. Morato *et al.*, "Toward Safe Human Robot Collaboration by Using Multiple Kinects Based Real-Time Human Tracking", *Journal of Computing and Information Science in Engineering*, vol. 14, no. 1, pp. 56–63, 2014.

[5] G. Michalos *et al.*, "Design Considerations for safe Human-Robot Collaborative Workplaces", *Procedia College International pour la Recherche en Productique (CIRP)*, vol. 37, pp. 248–253, Dec. 2015.

[6] M. Miknis *et al.*, "Near real-time Point Cloud Processing using the PCL", in *International Conference on Systems, Signals and Image Processing (IWSSIP)*, IEEE, Sep. 2015, pp. 153–156.

[7] J. Otepka *et al.*, "Georeferenced Point Clouds: A Survey of Features and Point Cloud Management", *ISPRS International Journal of Geo-Information*, vol. 2, no. 4, pp. 1038–1065, 2013.

[8] A. Niedermüller, C. Untner, and A. Meier, "Entwicklung einer Methodik zur Zusammenführung von Bildern zweier 3D-Kameras in Mensch-Roboter-Arbeitsplätzen", Bachelorarbeit, Informationstechnik und System-Management, Fachhochschule Salzburg, Puch bei Urstein, Feb. 2020. [Online]. Available: `https://github.com/nerovalerius/registration_3d/blob/master/BAC_1_niedermueller.pdf`.

[9] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)", in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Shanghai, China, May 2011, pp. 1–4.

[10] Corinne Stucker, "Semantic Point Cloud Filtering", Master Thesis, Swiss Federal Institute of Technology ETH, Zurich, Switzerland, Sep. 2017.

[11] A. Nüchter, "Semantische dreidimensionale Karten für autonome mobile Roboter", Dissertation, Institut für Informatik III, Rheinische Friedrich-Wilhelms-Universität, Bonn, May 2006.

[12] T. Vater, "Inline-Auswertung und -Korrektur multipler, komplexer, fusionierter, bildgebender Sensoren für das automatisierte Fahren", Bachelorarbeit, Fakultät Technik und Informatik, Hochschule für Angewandte Wissenschaften, Hamburg, Apr. 2018.

[13] K. O. Arras, "Feature-based Robot Navigation in known and unknown Environments", Dissertation, Microengineering Department, École Polytechnique Fédérale de Lausanne (EPFL), France, 2003.

[14] D. F. Wolf and G. S. Sukhatme, "Localization and Mapping in Urban Environments Using Mobile Robots", *Journal of the Brazilian Computer Society*, vol. 13, no. 4, pp. 69–79, 2007.

[15] A. Hornung *et al.*, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees", *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[16] S. M. Ahmed *et al.*, "Edge and Corner Detection for unorganized 3D Point Clouds with Application to Robotic Welding", in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 7350–7355.

[17] S. Se, D. Lowe, and J. Little, "Vision-based Mobile Robot Localization And Mapping using Scale-Invariant Features", in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, IEEE, vol. 2, 2001, pp. 2051–2058.

[18] C. Ramer, "Arbeitsraumüberwachung und autonome Bahnplanung für ein sicheres und flexibles Roboter-Assistenzsystem in der Fertigung", Dissertation, Technische Fakultät der Friedrich-Alexander-Universität, Erlangen-Nürnberg, Sep. 2018.

[19] Franka Emika, *Panda - Technical Data*, May 2018. [Online]. Available: `https://www.generationrobots.com/media/panda-franka-emika-datasheet.pdf` (visited on 03/25/2020).

[20] Intel Corporation, *Depth Camera D435*, 2018. [Online]. Available: `https://www.intelrealsense.com/depth-camera-d435/` (visited on 03/18/2020).

[21] R. A. Hamzah and H. Ibrahim, "Literature Survey on Stereo Vision Disparity Map Algorithms", *Journal of Sensors*, vol. 2016, pp. 1–23, 2016.

[22] H. Hirschmuller, "Stereo Processing by Semi-Global Matching and Mutual Information", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2007.

[23] K. Konolige, "Projected Texture Stereo", in *2010 IEEE International Conference on Robotics and Automation*, IEEE, 2010, pp. 148–155.

[24] W. Woodall, *Rviz*, 2018. [Online]. Available: `http://wiki.ros.org/rviz` (visited on 04/08/2020).

[25] I. A. Sucan and S. Chitta, *MoveIt*, 2019. [Online]. Available: https://moveit.ros.org/ (visited on 03/28/2020).

[26] T. Foote, *Community Metrics Report*, July 2019. [Online]. Available: http://download.ros.org/downloads/metrics/metrics-report-2019-07.pdf (visited on 03/23/2020).

[27] A. Topalidou-Kyniazopoulou, "Motion Planning Strategy For a 6-DOFs Robotic Arm In a Controlled Environment", Master Thesis, Computer Science Department, University of Bonn, Germany, Feb. 2017.

[28] ROS, *URDF*, 2019. [Online]. Available: www.ros.org/wiki/urdf (visited on 04/10/2020).

[29] Bosch, *GLM 40 Laser-Entfernungsmesser*, 2017. [Online]. Available: https://www.bosch-professional.com/at/de/products/glm-40-0601072900 (visited on 04/25/2020).

[30] Daniel Girardeau-Montaut, *CloudCompare*, 2019. [Online]. Available: http://www.cloudcompare.org/ (visited on 04/25/2020).

[31] J. Mieikis *et al.*, "Multi 3D Camera Mapping for Predictive and Reflexive Robot Manipulator Trajectory Estimation", in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2016, pp. 1–8.

[32] J. Bohren *et al.*, "Towards Autonomous Robotic Butlers: Lessons Learned with the PR2", in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 5568–5575.

[33] ROS.org, *ROS tf*, 2017. [Online]. Available: http://wiki.ros.org/tf (visited on 03/29/2020).

[34] Open Source Robotics Foundation, *Gazebo*, 2014. [Online]. Available: http://gazebosim.org/ (visited on 04/22/2020).

[35] A. Hermann *et al.*, "GPU-based Real-Time Collision Detection for Motion Execution in Mobile Manipulation Planning", in *16th International Conference on Advanced Robotics, (ICAR)*, Nov. 2013, pp. 1–7.