MASTER THESIS

An autodidactic Robot System for Object Reconstruction

Author: Dimitrij-Marian Holm
enrolment number (Matrikelnummer): 00512012
running number (Fortlaufende Nummer): 4

A thesis submitted in fulfillment of the requirements
for the masters course
APPLIED RESEARCH IN ENGINEERING SCIENCE
Robotic and autonomous Systems

Department of Electrical Engineering and Information Technology

Date of registration: 01.04.2018
Date of completion: 03.09.2018

MASTER THESIS

An autonomous robot system with the ability to acquire object information by manipulation for the purpose of object reconstruction

Author: Dimitrij-Marian Holm
enrolment number (Matrikelnummer): 00512012
running number (Fortlaufende Nummer): 4

Supervisor: Prof. Dr. habil. Alfred Schöttl

A thesis submitted in fulfillment of the requirements
for the masters course
APPLIED RESEARCH IN ENGINEERING SCIENCE
Robotic and autonomous Systems

Department of Electrical Engineering and Information Technology

Date of registration: 01.04.2018
Date of completion: 03.09.2018

Declaration of Authorship (Erklärungen der/des Bearbeiterin/s)

Name:

Vorname:

- Ich erkläre hiermit, dass ich die vorliegende Masterarbeit selbständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe.
- Sämtliche benutzte Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate sind als solche gekennzeichnet.

Ort, Datum:

Unterschrift:

- Ich erkläre mein Einverständnis, dass die von mir erstellte Masterarbeit in die Bibliothek der Hochschule München eingestellt wird. Ich wurde darauf hingewiesen, dass die Hochschule in keiner Weise für die missbräuchliche Verwendung von Inhalten durch Dritte infolge der Lektüre der Arbeit haftet. Insbesondere ist mir bewusst, dass ich für die Anmeldung von Patenten, Warenzeichen oder Geschmacksmuster selbst verantwortlich bin und daraus resultierende Ansprüche selbst verfolgen muss.

Ort, Datum:

Unterschrift:

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

Einverständniserklärung

Einverständniserklärung zum Einstellen einer Abschlussarbeit in den Bestand der Hochschulbibliothek

Thema der Abschlussarbeit

An autonomous robot system with the ability to acquire object information by manipulation for the purpose of object reconstruction

VerfasserIn

Dimitrij-Marian Holm

BetreuerIn

Prof. Dr. habil. Alfred Schöttl

Erklärung der Studierenden/des Studierenden

Ich erkläre mein Einverständnis, dass die von mir erstellte Abschlussarbeit in den Print-Bestand der Hochschule München übernommen wird oder in elektronischer Form über HM Digital publiziert wird.
Ich wurde darauf hingewiesen, dass die Hochschule in keiner Weise für die missbräuchliche Verwendung von Inhalten durch Dritte infolge der Lektüre der Arbeit haftet. Insbesondere ist mir bewusst, dass ich für die Anmeldung von Patenten, Warenzeichen oder Geschmacksmustern selbst verantwortlich bin und daraus Ansprüche selbst verfolgen muss.
Der bibliographische Nachweis meiner Publikation erfolgt lokal (OPAC), weltweit (WorldCat) und über Internet-Suchmaschinen.

Ort, Datum | Name, Unterschrift

Erklärung des Unternehmens/der Organisation

(gegebenenfalls vom Betreuer/der Betreuerin zu streichen und abzuzeichnen)
Es besteht Einverständnis, dass die Abschlussarbeit in die Bibliothek der Hochschule München aufgenommen wird.

Ort, Datum | Name, Unterschrift

Einverständniserklärung des/der betreuenden Hochschullehrers/-lehrerin bzw. Lehrbeauftragten

Es besteht Einverständnis, dass die Diplom-/Master-/Bachelorarbeit in die Bibliothek der Hochschule München aufgenommen wird.

Ort, Datum | Name, Unterschrift

"The truth may be out there, but the lies are inside your head."

Terry Pratchett

UNIVERSITY OF APPLIED SCIENCE MUNICH

Abstract

Department of Electrical Engineering and Information Technology

Master of Science

An autodidact Robot System for Object

by Dimitrij-Marian Holm

German:
Objekterkennung ist eines der fundamentalen Bereiche in der Entwicklung von intelligenten autonomen Systemen. Um ein autonomes System zu entwickeln, das die Umgebung wahrnimmt und mit ihr interagieren kann, sind viele weitere Aufgaben zu bewältigen. Diese wären z.B. eine Verarbeitung der von den Sensoren gelieferten Informationen, die eine Darstellung vorhandener Objekte ermöglicht, welche notwendig für die Objekterkennung ist. Ohne die Möglichkeit der Manipulation des Umfeldes sind autonome Systeme nur eingeschränkt befähigt, detailliertere Informationen aus der Umgebung zu sammeln. Des weiteren ist Objekterkennung ein inkrementeller Bestandteil für weiterführende Aufaben.

Aus diesem Grund wurde das Projekt zur Entwicklung eines autonomen Systems umgesetzt, dass durch Umgebungsinteraktion spezifische Daten von Objekten erhebt und diese zu einem 3D-Modell zusammensetzt. Der Ansatz kombiniert alle Schritte, die für die Erzeugung eines digitalen Modells von Objekten gebraucht werden, ausgehend von einer Vorverarbeitung der Eingangsdaten über Objektextraktion, Umgebungsmanipulation zur genauen Untersuchung bis hin zur Datenfusion, das letztendlich zu einem 3-D Modell führt. Diese Ergebnisse können für sämtliche weiteren Aufgaben verwendet werden, in denen 3D-Modelle benötigt werden.

English:
Object recognition is one of the major topics in developing intelligent autonomous systems. But to develop a fully autonomous system which is able to gain information about the environment including interaction within, a lot of additional tasks are required such as processing of the sensor inputs leading to a representation of different objects which is needed for object recognition. Without the manipulation possibility of the surrounding an autonomous system might not be possible to gather detailed information about objects. In addition object recognition results are needed for further processing or interactions.

This is why we introduce our autonomous system development project. The approach includes all necessary process steps to gain a digital representation of an object. It starts with preprocessing of inputs, extracting data of interest, environment manipulation to obtain additional information and finally data fusion to accomplish a depiction of the gained data. The result can be used for every task which needs a 3D-Representation of objects as a point cloud or a mesh.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

PC	**P**oint **C**loud
PCL	**P**oint **C**loud **L**ibrary
OR	**O**bject **R**ecognition
OOP	**O**bject **O**rientated **P**rogramming
SLP	**S**elf **L**earning **P**ipeline
TCP	**T**ool **C**enter **P**oint
GPD	**G**rasp **P**ose **D**etection
GPG	**G**rasp **P**ose **G**enerator
HAF	**H**eight **A**ccumulated **F**eatures
nn	**n**earest **n**eighbour
ds	**d**owns**s**ized
curv	**curv**ature
ac	**a**ngle **c**loud
spe	**s**urface **p**ose **e**stimation

1 Introduction and Overview

Household robotics, collaborative manufacturing units, support in nursing and healthcare, just to mention some fields of application, autonomous systems getting more and more involved into our life. They shall support humankind in every area and make our life more convenient. To reach this goal a necessary requirement of the autonomous system is a fundamental understanding of its environment, the system needs an intelligent behaviour for this. One big research topic is object recognition, developing different approaches, algorithms and technologies to give systems the proficiency to recognize objects, making it possible to distinguish between different parts of the environment. For example an intelligent household robot needs to know what a bottle is if it is asked to serve one. A huge database is necessary to seduce parts of the environment and associate them with stored information allowing the system to find and recognize objects. Regardless if the recognition algorithm is based on classical mathematical models or modern machine learning techniques like neuronal nets, it is necessary to gather information about objects.

As mentioned in [1], one approach, the supervised object recognition, needs information about the object a-priori, making it inevitable to generate these information before an object recognition can succeed. It is only used for recognition. One benefit is the flexibility of these approaches in recognition, but depending on the technique a huge data set for neuronal nets or other information like a 3D-model are needed before using. These information are created externally which is not desired for autonomous systems. Autonomous systems should not depend on external created data, it would deny any reaction to a dynamic environment, a system able to achieve information by its own is more useful.

On the other hand and what leads more to a autonomous system, unsupervised object recognition removes the constraints of the need of data by an additional detection part. The huge advantage is the possibility to gain information by its own, but there are still several restrictions as mentioned in [1] like the reduced flexibility the supervised object recognition is confronted with.

The Project was developed to combine both general meanings of the approaches. It is not the goal to improve or enhance a current object recognition task, supervised or unsupervised. The main idea is to develop a system using the idea of both approaches supporting each other. It is a kind of hybrid, because the object recognition still needs information before the recognition can work. What it makes hybrid is the point that the autonomous system can gain the information while it is working within the environment, accumulates more and more data for object recognition over the time, the benefit of unsupervised recognition. The information is calculated to a suitable data for supervised object recognition approaches to keep the flexibility benefit.

1.1 Overview

The Project contains a pipeline which is able to generate object data for object recognition tasks. The only input needed is a point cloud as shown in figure 1.2. With this information the approach calculates and synthesises the required data for further tasks. The pipeline itself contains the following tasks (fig 1.1):

- Preprocessing
- Segmentation
- Tracking
- Grasp Pose Estimation
- PC-Controller
- Registration
- Mesh Assembly

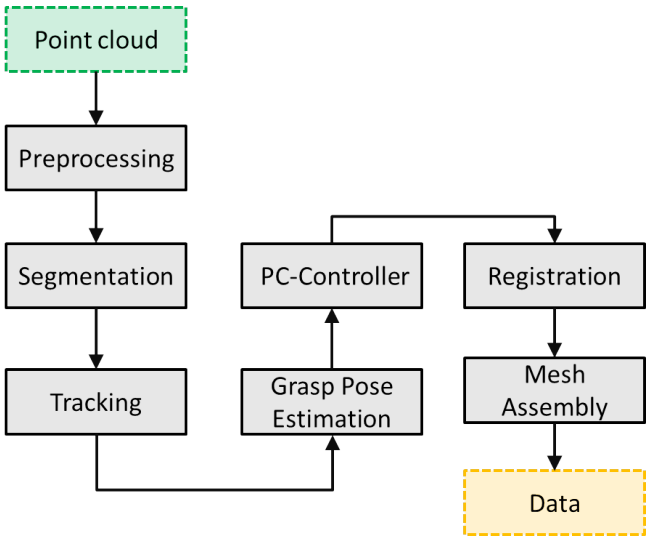


FIGURE 1.1: The whole Pipeline of the project. An input (green box) is represented by a point cloud. Grey boxes are processing tasks. The yellow box is the final output, data which can be used for object recognition or other tasks.

The autonomous system as well as the pipeline need at least sensors for point cloud acquiring and a tool for object manipulation, a robot arm with a gripper for example. The working space of the autonomous system, the environment might be a kitchen or a whole flat, or even outside of a building. Within the environment there are objects, which can occur as a cup for example and may exist several times. Similar to the principles of OOP the objects general representation is a class. An object is just a specific instance of a class, this means several cups share the same class, cup, but have different attributes as design, color and position. Object recognition tries to find the occurring class within an environment.

FIGURE 1.2: A snapshot of an input. It represents a point cloud with pixel coordinates (x,y), color (rgb) and depth information (z) for every pixel. This point cloud shows a environment as it is used in our test scenario.

1.2 Related Work

While the pipeline contains a lot of processing tasks, several of them were developed in the last project [1]. It contains the Preprocessing, Segmentation, Tracking, PC-Controller, Registration and Mesh Assembly parts. Main focus was to find an approach to create the necessary OR data based on an raw input. One restriction was the lack of environment manipulation by the autonomous system, which is forced in this thesis. An overview without manipulation can be found in figure 1.3.

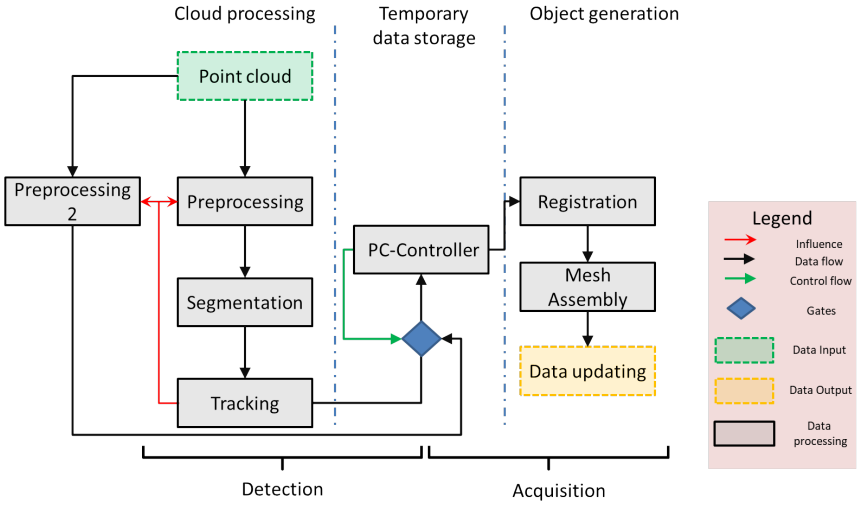


FIGURE 1.3: The Pipeline without the current manipulation extension as developed in [1]. An explanation of the pipeline is given in the sections below.

As mentioned the pipeline input is a raw point cloud as shown in figure 1.4. It contains a red vase, a table and walls. This input is used to demonstrate the processing steps.

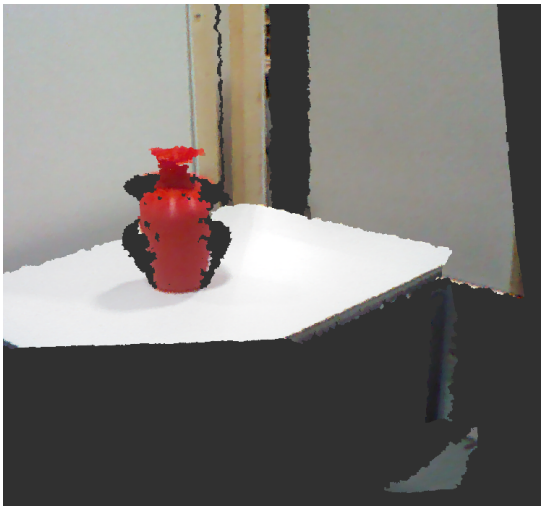


FIGURE 1.4: Another input. This input is used as a showcase for further pipeline results.

1.2.1 Preprocessing

Preprocessing contains several calculation tasks, downsizing, outlier removal, smoothing and plane removal. Downsizing decreases the resolution of the input to achieve a faster processing speed in further tasks. To diminish occurring noise within the PC and reduce possible mismatches, outliers (single points with no related neighbours) are removed and the whole cloud is smoothed due to the occuring stairway effect as shown in figure 1.6. In addition plane removal profits from smoother point clouds as possible plane surfaces are better detected. As the focus is on household items which are on a table the algorithm can extract unneeded information such as walls and tabletops. The remaining PC (figure 1.5) only contains regions of points within a certain amount of points. This are our possible objects of interest. For a deeper explanation how the tasks are working, see [1].

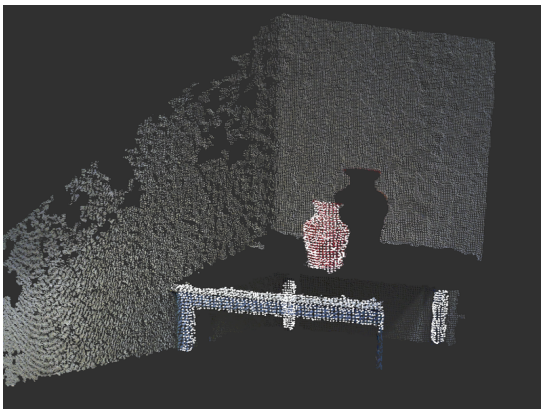


FIGURE 1.5: After preprocessing the point cloud has no outliers, big plane surfaces like walls were removed, remaining surface are smoothed and contains a lesser dense of points. The remaining points are shown as white dots.

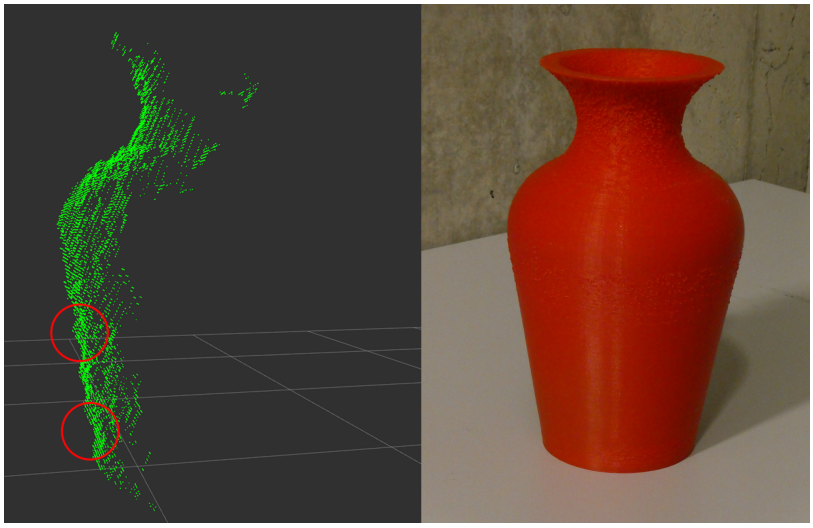


FIGURE 1.6: Due to the angle of the camera system stairway effects may occur on the surface of objects, marked with red circles. Here, a vase shows the stairway effect in the PC while the real object has a even surface.

1.2.2 Segmentation

At this time the point cloud is reduced an undesired points are removed, but the remaining points are not structured nor necessary information are extracted. The Segmentation separates the remaining PC so the pipeline can distinguish between different clouds which contains objects. The sub PCs are shown in figure 1.7. Every color represents an object.

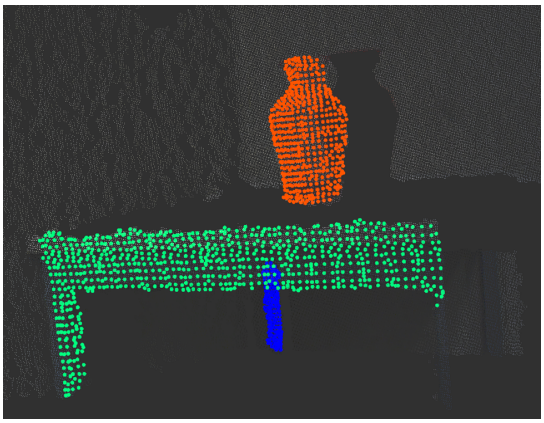


FIGURE 1.7: The PC is clustered into different sub PCs to distinguish between different possible objects (red = vase, green = front side of table, blue = a table leg).

1.2.3 Tracking

Out of the set of sub PCs the tracking task chooses one point cloud. the geometrical centroid is calculated (see figure 1.8, green dot), stored and a flag that a object is locked and tracked is set. As new point clouds reach the pipeline and the remaining information after preprocessing and segmentation are handled to the tracking task, the centroids of all sub PCs are calculated and compared with the stored one, the comparing parameter is the distance between the centroids. A valid tracking counts, if at least one calculated centroid exists within a certain range to the stored centroid. If there is no valid centroid found, the tracking task will wait for further incoming point clouds, trying to re-track the object. After several non successful comparisons the tracked target is seen as lost and a new object will be tracked. If an object is locked, the tracked PC is handled to the gate of the PC-Controller.

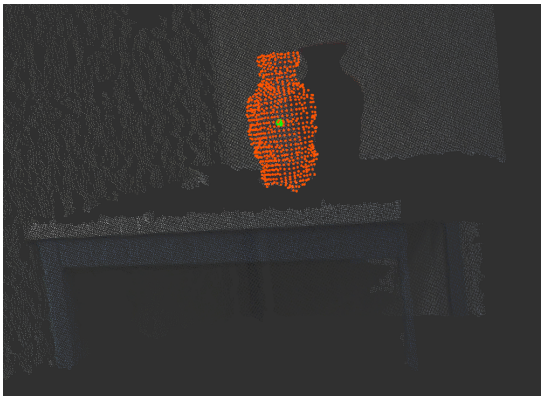


FIGURE 1.8: The tracking algorithm chooses one PC out of the sub PCs which is now the object of interest for the pipeline (red points shows the PC of the object of interest, the green dot represents the tracking point).

The influence as a red arrow in figure 1.3 (points from tracking to preprocessing) enables a local point cloud filter in preprocessing, removing all points which are not

in a certain neighbourhood around the centroid. This allows additional computation effort savings due to there is no need to use the whole point cloud while only a certain area is significant. In figure 1.9 the green dots show the remaining points. Furthermore the task preprocessing 2 starts, using spatial information from tracking to carve out a full resolution point cloud. It is deeper described in 1.2.6.

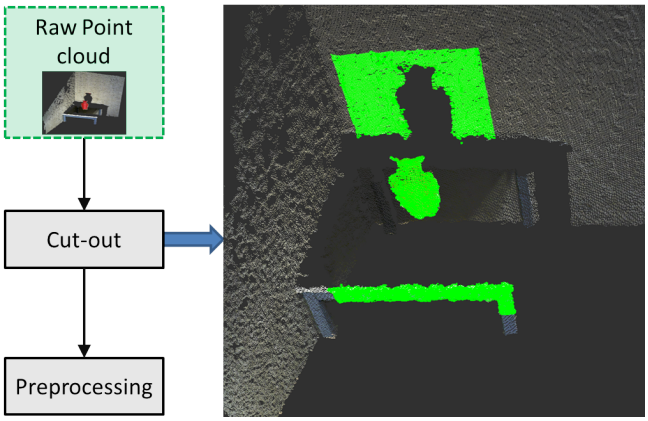


FIGURE 1.9: If the Tracking task is locked to an object, preprocessing reduces the PC. Every point, expect the neighbourhood of the centroid, is removed from the point cloud. Remaining points are shown as green dots

1.2.4 PC-Controller

The PC-Controller is a data storage and steering unit. As the cloud processing tasks (1.3, left column) works for every incoming point cloud frame, the PC-Controller waits for an external signal allowing the tracked PC to pass through the gate and will be stored in the PC-Controller. The signal is send manually (we refer to the development status in [1]), but the signal is only valid if the tracking task has locked an object. If a certain amount of n PCs are stored in the controller, the PC-Controller activates the Registration task and transfers the stored PCs.

As the approach does not consists to change the pose of the system itself but rather expect a pose change of the object, the stored PCs are superimposed as shown in figure 1.10.

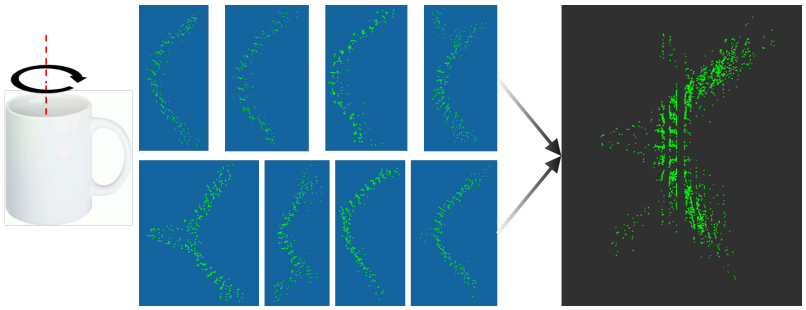
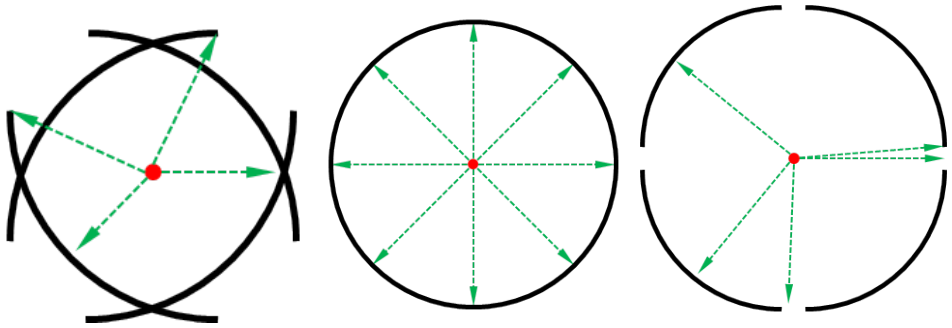


FIGURE 1.10: With the same viewpoint, the stored sub PCs are superimposed. This is an example of the resulting PCs of a cup. The cup is rotated around the red dashed line. The stored PCS are next to the cup with the blue background. If all PCs are matched together, the PC is superimposed (right side, black background).

The superimposed PCs must further registered to each other to gain the desired object information representation.

1.2.5 Registration

Main idea of this registration approach is the contemplation of the distance between a circle perimeter and its center, which is always the same, regardless of the position on the perimeter(figure 1.11). If we divide the perimeter in several parts, we have an similar scenario as the superimposed PCs, as only the surface of an object is visible. The only distinctions are that our PCs are not distributed around the center and the surface might not be occur as a perfect perimeter. One more practical example is shown in figure 1.12. It shows the superimposed PCs of a cup.



(A) collapsed circle (B) perfect shaped circle (C) expanded circle

FIGURE 1.11: three possible states of a circle. The distances (green arrows) between the center of gravity (red dot) and the perimeter (black) in (A) and (C) are not equal. In (B) the distances are equal and therefore represents the best registration.

First, the approach calculates a geometrical centroid using all current point in the PCs. Second, a vector v_{init} is calculated between the camera position and the centroid (camera position as initial point, centroid as terminal point), the length of the vector $||v_{init}||_2$ represents our initial scaling $r = 1.0$. While the camera experiences no pose change, a modification in the scaling value r leads to a new vector length and therefore to a new centroid position. In this special case the shifting of the centroid position leads to the same results as expanding or collapsing the perimeter parts towards the centroid. For several scaling values the distances between the resulting

shifted centroids and the points of the fixed point clouds are calculated and stored. The amount of different scaling r is based on the iteration parameter ΔR. In conclusion every scaling parameter has its array of distances and every array is sorted ascending to gain a discrete function f with the distances in ascending, storing the distances between points and centroid.

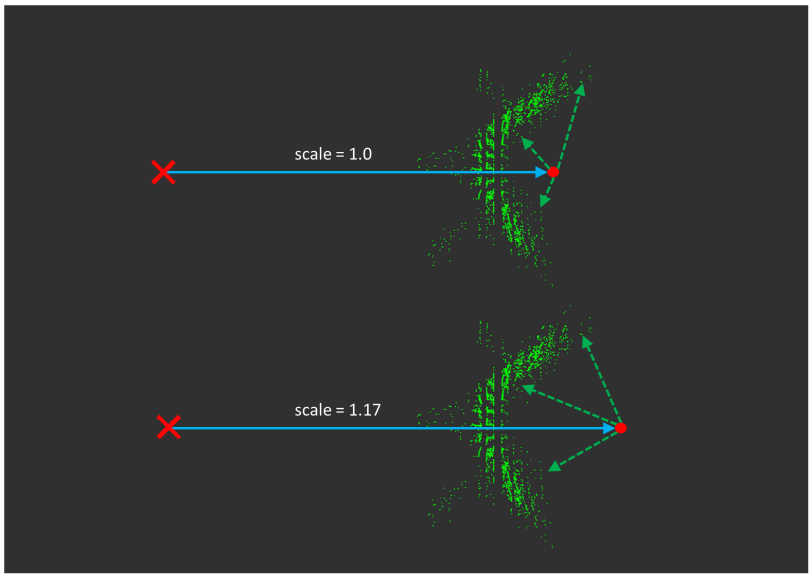


FIGURE 1.12: The Algorithm calculates a vector (blue arrow) between the camera position (red cross) and a geometric centroid (red dot) once, it receives the scaling of 1. As the scaling factor changes, the centroid is shifted and the distances (green arrows) between centroid and the points within the point clouds are calculated.

With 1.1 we can calculate the quality parameter Q for every scaling r , Q is therefore a vector with dimension $\mathbb{R}^{\Delta R \times 1}$ The equation judges the curvature of the sorted distances. Figure 1.13 shows the sorted distances of one scaling $r = 1.002925$.

$$Q_k = \sum_{j=1}^{i} |f_k(j-1)'' - f_k(j)''| \tag{1.1}$$

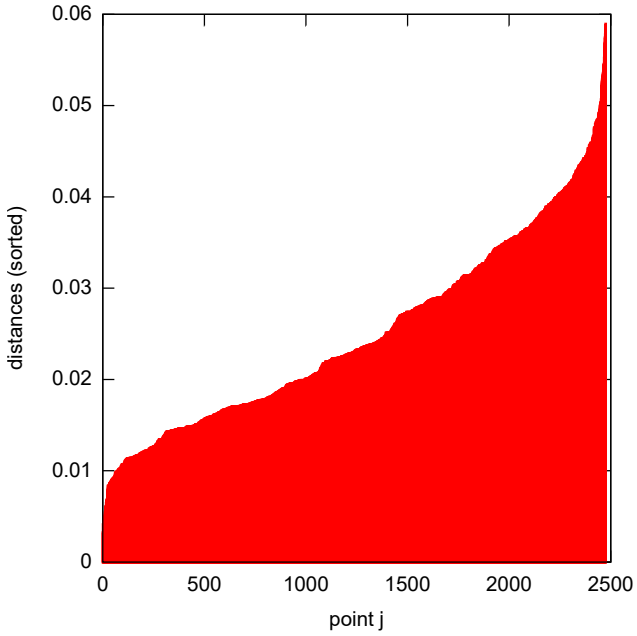


FIGURE 1.13: Sorted distances for every point within the stored sub PCs.

If all quality parameters Q are gained for different scaling factors r, the best scaling factor r_{opt} is calculated by searching the lowest quality parameter Q_{opt} as shown in figure 1.14

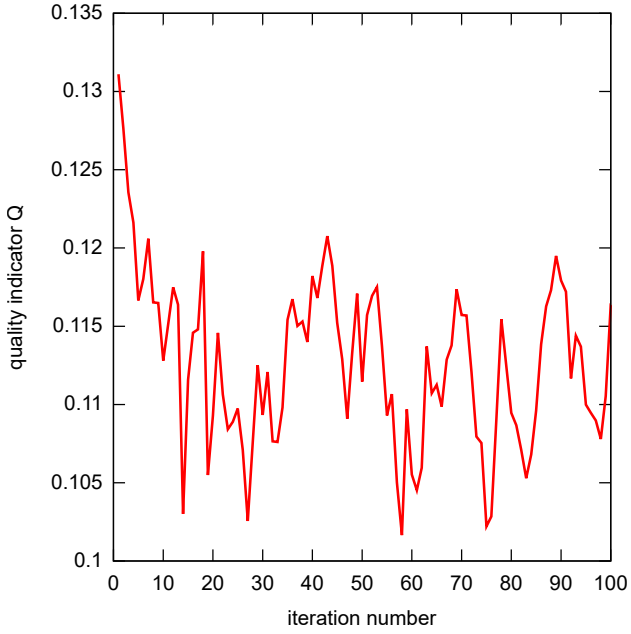
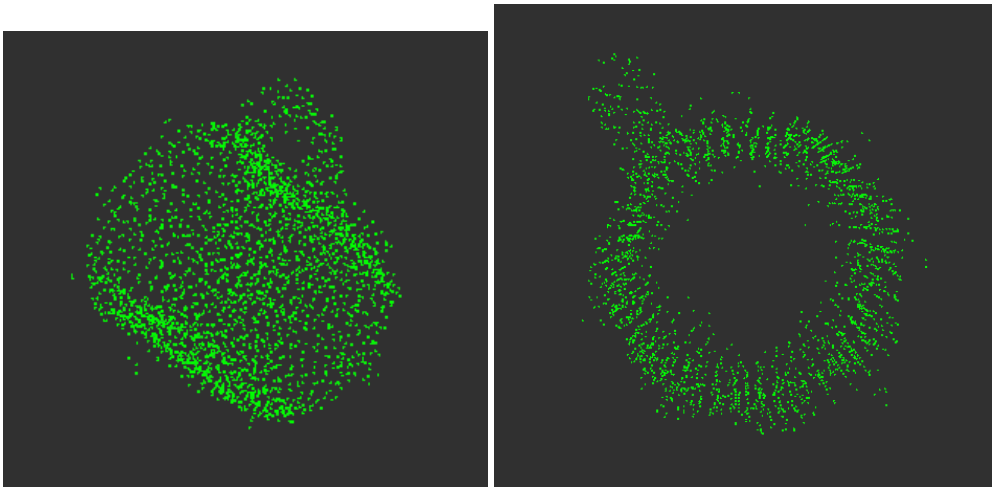


FIGURE 1.14: The resulting quality parameters 1.1 based on iterations. The iterations define the change of r, here with $\Delta R = 100$

The scaling factor r_{opt} represents the best position of the rotation axis for the

registration. The rotation axis is always in direction of the gravity (it was the y-axis in [1], in this autonomous system it is defined as z-axis). Using this factor in combination with the initial vector v_{init} the optimal centroid is found by $v_{init} \cdot r_{opt}$ as our rotation point $p_{rot,opt}$ from where the PCs are rotated. Every PC is rotated around this point with the z-axis as rotation vector. Using the stored n PCs, the rotation angle $\alpha = \frac{2\pi}{n}$ is calculated and used for the index $k = 1, 2, 3, ..., n$ to rotate every point cloud k with the angle $k \cdot \alpha$ and are combined to one big PC. A result of the registration algorithm is shown in figure 1.15.



(A) a cup registered, top view (B) a cup registered, default view

FIGURE 1.15: A result of the registration algorithm (A). A good approach to the main idea, the approximation of circles, is visible on B.

1.2.6 Mesh Assembly

After the Registration all required information are acquired to create a 3D-Mesh. The final result of the whole Pipeline is shown in figure 1.16. The created mesh quality is strongly based on the resolution of the final PC. Since we reduced the input PC in the beginning of our pipeline Preprocessing) and use the input for the rest of the pipeline, the quality of the mesh differs. Figure 1.3 has additional tasks compared to the original based on [1]. To gain the best possible quality it is necessary to reduce any change of the input to a minimum, modifying the behaviour of our pipeline.

If the Tracking task locked an object, the Preprocessing 2 algorithm starts. With spatial information based on the tracked object and the corresponding PC we can extract our high resolution PC from the input. We generate a cube around the tracked point cloud and all points of the raw input point cloud which are outside of this cube are excluded, gaining a point cloud only containing a high resolution image of the object. Additional processing steps beside the extraction are only to remove outliers and smooth the point cloud. If the PC-Controller stores a PC snapshot, the input contains two point clouds. The first one is still the reduced PC from the Tracking, the second is the full resolution PC from Preprocessing 2. All calculations remains the same in the Registration, the low resolution PCs are used for quality score calculation. Only for the final registration step, the rotation around the centroid $p_{rot,opt}$ the new, high resolution PCs are used and handled to the Mesh Assembly. With an

manageable increase of computer and storage afford the pipeline is able to create much better 3D-Meshes.

FIGURE 1.16: A 3D-Mesh of a vase, created with a PCL-algorithm.

1.2.7 Conclusion

The pipeline based on [1] for 3D-Mesh synthesizing which is needed for object recognition tasks is a helpful approach for developing an autonomous system with the ability of self learning. A missing possibility for environment manipulation from [1], which is done manually in the approach, constraints the usage. To fulfil the requirements of self learning the approach is expanded with the possibility of environment interaction by a robot arm.

2 Grasp Pose Detection

Previously the pipeline was only an input driven passive algorithm with a straight processing order and no possibility to interact with the environment. Extending the pipeline with the possibility to interact with objects the approach develops to an active system, leading to new challenges. One major challenge is the grasp pose estimation which calculates a valid grasp position for a robot gripper in order allowing the robot to grasp the object. The used input PC pc_{init} for the grasp pose detection is the PC after Preprocessing 2 (figure 1.3), a smoothed high resolution PC with removed outliers. It contains the points $p_k^{init}(x, y, z)$ where k is the k'th point inside pc_{init}:

$$pc_{init} = \{p_1^{init}, p_2^{init}, ..., p_n^{init}\} \tag{2.1}$$

More general, a point cloud is defined as:

$$p \in pc, \text{ with } p = \{p_1, p_2, ..., p_n\} \tag{2.2}$$

and to obtain the k'th point in a certain point cloud:

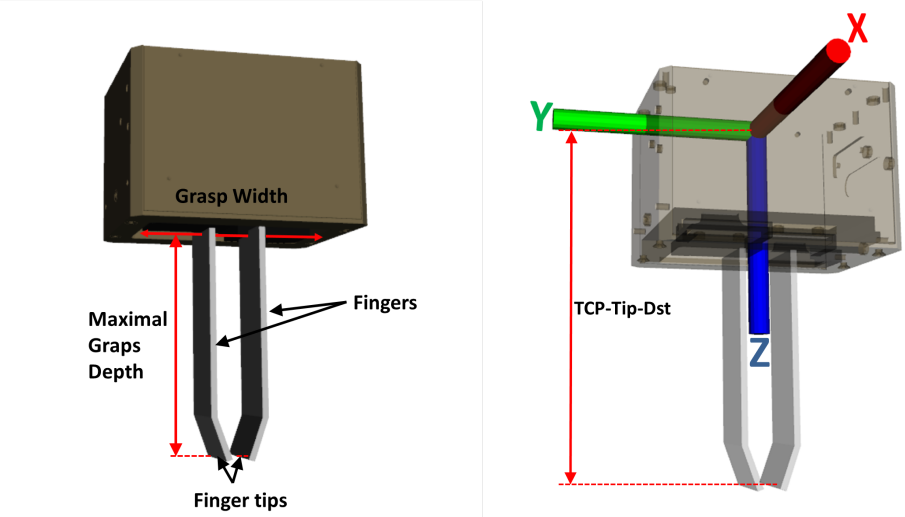
$$pc(k) = p_k \tag{2.3}$$

If n is used as an index in a point it is defined as maximal number of points in the point cloud (e.g. , p_n^{init} is the last point in pc_{init}). $n(pc)$ describes the amount of points in the point cloud. If the notation $pc(n)$ occurs it means that it is the last point in the point cloud, so $pc_{init}(n) = p_n^{init}$. As several object or classes might occur next to each other in our environment (but with a minimum of 3 cm distance between object surfaces, [1]), we defined a grasping vector which is in direction of the gravity (1.15, z-axis, blue). One of our requirements is to rotate objects, this restriction reduces the probability for unwanted collisions between robot and objects if we grasp from above.

As seen in figure 2.1, our approach is based on parallel gripper, allowing antipodal grasps which needs two grasping points. Based on of the gripper and the grasping direction, the objects geometry itself is restricted as followed: a full visibility of the geometry if the viewpoint is set to the top (Object is seen from z-Axis). A partial visibility of the front view should also be present but there are exceptions. It allows us using the front geometry to adapt grasping points and estimate the covered parts of the object by extrapolating the top geometry in z direction, like a inverted watershed algorithm. As a conclusion the non visible geometry should be as straight as possible in z direction.

2.1 Related Work

First approaches uses 2D-Image as imformation source, but they have limited possibilities of detection as used in [2]. Even more approaches occur when depth information comes into account, allowing the use of geometric interpretations of point



(A) a model of the used gripper.

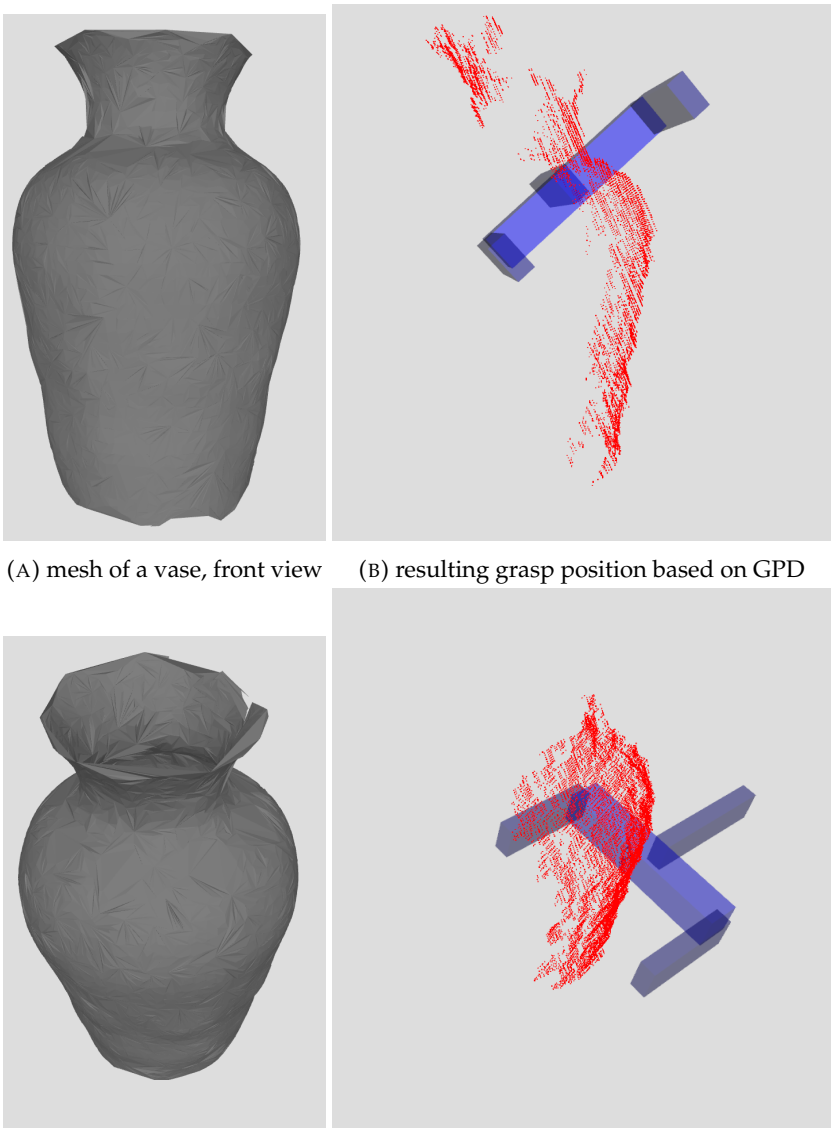
(B) The coordinates axis (red = x, green = y, blue = z) and the TCP (Intersection of the axis) of the gripper. GCP-Tip-Dst is the euclidean distance between the TCP an the finger tips in z direction.

FIGURE 2.1: A model of the used gripper (A). The left figure shows the antipodal grasp fingers which are movable in y direction. The Grasp Width y_{g_max} describes the maximum distance which is possible between the fingers. The Maximal Grasp Depth z_{g_max} is the most depth the gripper can reach with the finger tips if an object is between the fingers. The TCP and the axis are shown on B.

clouds, like spherical harmonics [3] [4], generalized cylinders [5], symmetric seeking models [6] [7], superquadratic [8] or hyperquadric approaches [9]. All these approaches unite the fact that they try to solve the grasping problem by calculating graps quality metrics [10]. The valuation is the success rate, it represents the perenctage of successful grasps, for example 73% in [11]. Based on the tremendous progress in machine learning over the past years a new way for finding good values for the metrics occurs [12], [13], [14], [15]. The success rate in [11] was raised by 15% to 88% by the support of machine learning. All in all it is possible to distinguish between three areas of grasp research: classic analytic methods, approaches supported with machine learning and full machine learning (like deep learning and convolution nets). The previous introduced algorithms are mostly the combination between analytic and machine learning methods , a fully machine learning approach is [16], where only reinforcement learning and convolution layer were used with the goal to rotate an object within a robot hand (5 finger, similar to a human hand). It is not directly a grasp pose estimation approach, but it still needs to grasp objects in order to executes the given task. To teach the neuronal net, the robot trains approximately 100 years if the system would perform the grasp training in reality (but 50 h in the simulation setup).

First we implemented the grasp pose detection [17] because of the high success rate in grasping novel objects, which reaches 93% in cluttered scenes. This approach uses the combination of regular methods, a grasp pose generator [18], which creates a bunch of possible antipodal grasp candidates using geometric conditions in combination with machine learning for classifying grasps as viable or not. As our

object is unknown and based of our sensor input we only know a small part of the objects surface (the back is never visible), the results of GPD was not satisfying for our needs. First there is no possibility to define a favour grasping vector and second, through the unknown back of the object, the grasping pose was estimated within the object itself (2.2), denying any use for our scenario.



(A) mesh of a vase, front view (B) resulting grasp position based on GPD

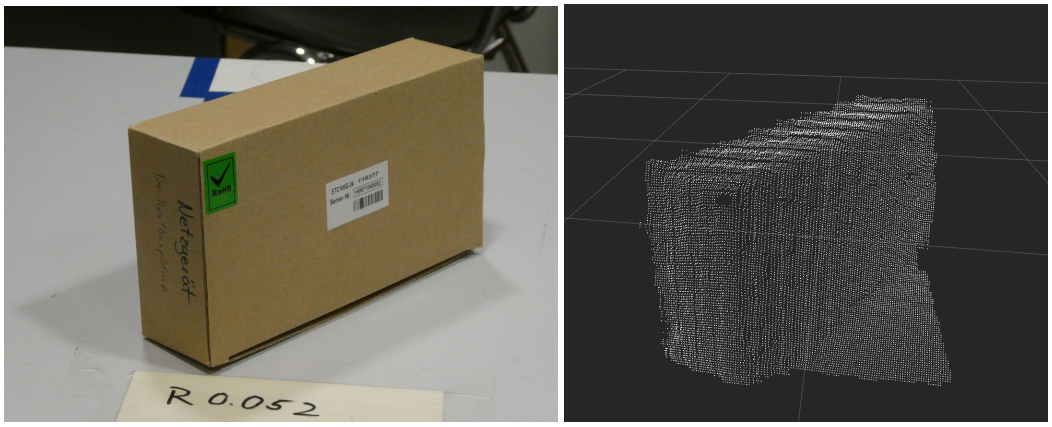
(C) mesh of a vase, top view (D) resulting grasp position based on GPD

FIGURE 2.2: The resulting grasp pose estimations based on the GPD ([17]). It shows that the algorithm does not satisfy the requirements of our scenario.

The second approach is HAF-Grasping [19]. It contains also a machine learning approach for supporting grasp pose calculating. One big advantage is that HAF-Grasping is designated as usable in cases where the object are unknown, so no a-priori information are needed, a geometrical primitive fitting to estimate the possible shape of the object is not used. The second benefit is the use of a grasping vector as utilized in our idea. The approach calculates height information (based on the grasping vector) of clustered point clouds in order to gain a topological description and is developed for antipodal gripper which is also a requirement for our robot

system. The average success rate is 85 % for grasping different objects. By comparing the differences between height regions they gain an abstraction of the shape provides usable parameters for machine learning. The results of this approach are compared with the results provided by our approach in section 4.

2.2 Concept



(A) the real world object, a box

(B) the resulting grasp point as used for further tasks

FIGURE 2.3: A real object (box) (A) and the resulting point cloud (B) as it is provided by the tracking task (1.3) which is used as an example for further descriptions.

Like the whole pipeline the Grasp Pose Detection is organized as a pipeline (figure 2.4). Using the input from Preprocessing 2 (figure 1.3), the output is the position of the TCP $p_{tcp}(x, y, z)$ and the orientation of the gripper as quaternion $q_{tcp}(x, y, z, w)$.

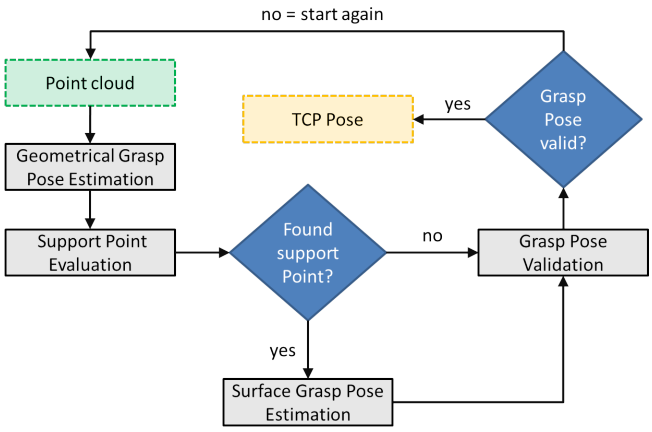


FIGURE 2.4: The green box represents the input PC, the yellow box the TCP-Pose as output. Grey boxes shows different tasks and the data flow is shown by black arrows, while the blue diamond shapes are decision based gates.

2.2.1 Geometrical Grasp Pose Estimation

To gain a first, coarse grasp pose we use the geometrical information of the PC. The lowest, z_{min}, and highest, z_{max}, occurring z entries in the input PC are collected. As the gripper has a limitation in grasping, based on the finger tips related to the TCP in z direction z_{g_max}, it is useful to remove all points which are not graspable, so the remaining PC pc_1^{cut} is given as:

$$pc_1^{cut} = \{p \in pc_{init} \mid z_{max} - p(z) < z_{g\_max}\} \tag{2.4}$$

The outcoming result pc_1^{cut} is shown in figure 2.5.

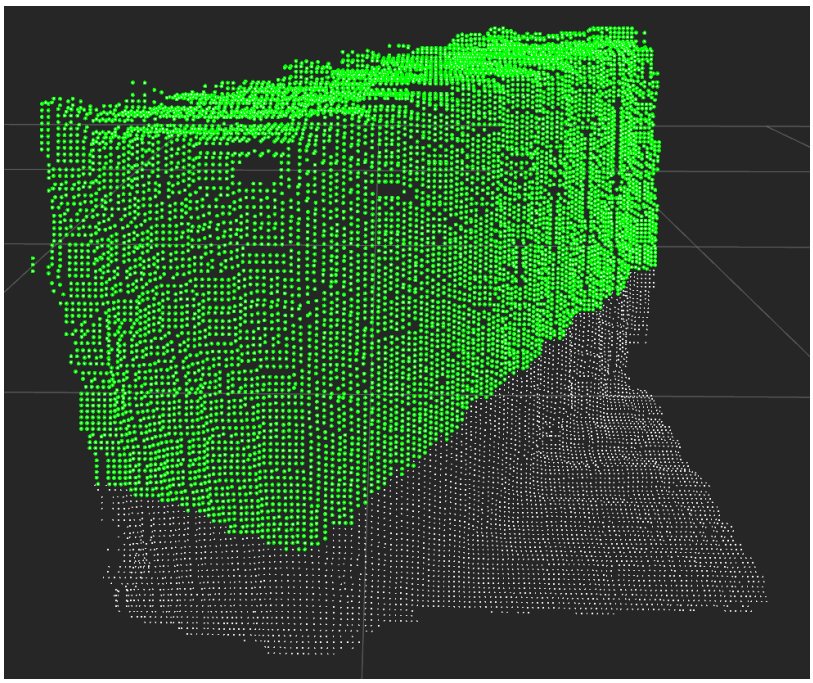


FIGURE 2.5: A PC where the point were removed which are outside the grasping depth. The white dots are points of the input PC pc_{init}, the green dots the remaining points in pc_1^{cut}.

Since the grasping vector points in z direction most following algorithms will use the x and y coordinates for calculation. With the assumption that our vision sensor gives us enough information about the object top surface we can calculate four points p_1^{corner}, p_2^{corner}, p_3^{corner}, $p_4^{corner} \in pc_1^{cut}$ with the restrictions that all of the points are in maximal distance between each other, described in algorithm 15. Figure 2.6 shows the calculated corner points from pc_1^{cut} as green dots with a top view.

Calculate corner points (pc_1^{cut})

$p^{rand} = random\ point\ from\ pc_{cut}$
if $max(\|p^{rand}(x,y) - p_k^{cut}(x,y)\|_2)$ **then**
  $p_1^{corner} = p_k^{cut}(x,y,z)$
end
if $max(\|p_1^{corner}(x,y) - p_k^{cut}(x,y)\|_2)$ **then**
  $p_2^{corner} = p_k^{cut}(x,y,z)$
end
if $max(\|p_1^{corner}(x,y) - p_k^{cut}(x,y)\|_2 + \|p_2^{corner}(x,y) - p_k^{cut}(x,y)\|_2)$ **then**
  $p_3^{corner} = p_k^{cut}(x,y,z)$
end
if $max(\|p_1^{corner}(x,y) - p_k^{cut}(x,y)\|_2 + \|p_2^{corner}(x,y) - p_k^{cut}(x,y)\|_2 + \|p_3^{corner}(x,y) - p_k^{cut}(x,y)\|_2)$ **then**
  $p_4^{corner} = p_k^{cut}(x,y,z)$
end
return p_1^{corner}, p_2^{corner}, p_3^{corner}, p_4^{corner}

Algorithm 1: The algorithm used for corner point calculation. While p_1^{corner} is calculated between two random points, the subsequent points are always in dependency with the previous points. For every corner point calculation k is iterated between 1 and n as the maximum points within pc_1^{cut}.

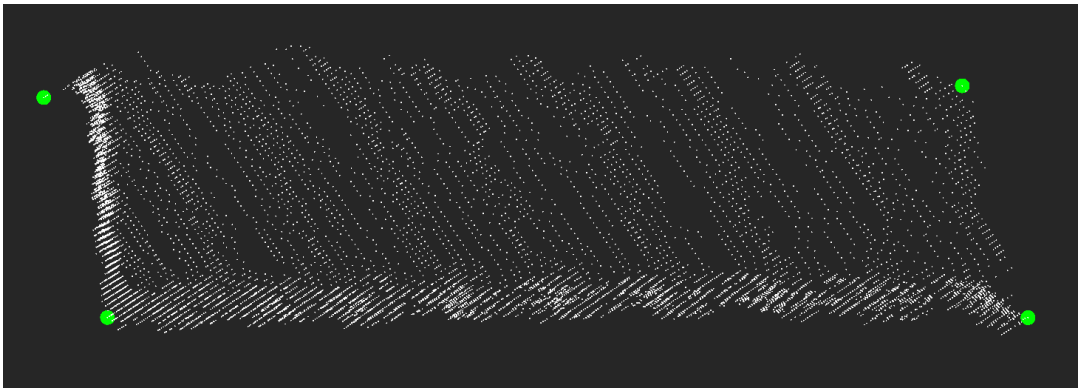


FIGURE 2.6: The calculated four corner points p_1^{corner}, p_2^{corner}, p_3^{corner}, p_4^{corner} of the PC (green dots).

The four corner points are paired together in two with the condition of the nearest neighbour, so supporting vectors $\mathbf{v}_{1\_2}^{sup}$, $\mathbf{v}_{3\_4}^{sup}$ are constructed between them (figure 2.7). With the support vectors and the corner points the points $p_{1\_2}^{sup}$ (equ. 2.5) and $p_{3\_4}^{sup}$ (equ. 2.6) are calculated.

$$p_{1\_2}^{sup} = pc_1^{corner}(x,y) + 0.5 \cdot \mathbf{v}_{1\_2}^{sup}(x,y) \tag{2.5}$$

$$p_{3\_4}^{sup} = pc_3^{corner}(x,y) + 0.5 \cdot \mathbf{v}_{3\_4}^{sup}(x,y) \tag{2.6}$$

Using $p_{1\_2}^{sup}$ and $p_{3\_4}^{sup}$ to span the vector $\mathbf{v}_5^{sup}$ and therefore calculate p^{TCP} with equation 2.7. While equation 2.7 only calculates the x and y coordinates the z value

is given by 2.8. $\Phi_g = \{x \in \mathbb{R} \mid 0 < x < 1\}$ is the percentage of grasp depth based on the depth extend of the PC. Because the TCP is in a higher position than the finger tips, TCP-Tip-Dst must be added to the z coordinates.

$$p^{TCP}(x,y) = p^{sup}_{1\_2}(x,y) + 0.5 \cdot \mathbf{v}^{sup}_5(x,y) \tag{2.7}$$

$$p^{TCP}(z) = (1 - \Phi_g) \cdot z_{max} - dg \cdot z_{min} + \text{TCP-Tip-Dst (fig.2.1 B)} \tag{2.8}$$

With p^{TCP} equation 2.9 calculates $\mathbf{v}^{gripper}$ with the condition that $\mathbf{v}^{gripper}$ and $\mathbf{v}^{sup}_5$ are perpendicular to each other, so $\mathbf{v}^{gripper} \perp \mathbf{v}^{sup}_5$. While p^{TCP} describes the position of the TCP, $\mathbf{v}^{gripper}$ is used for the orientation of the TC to describe the full pose of the gripper.

$$\mathbf{v}^{gripper}(x,y) = \mathbf{v}^{sup}_5(-y,x) \tag{2.9}$$

For further estimations it is necessary to find the position for the finger tips. As the rotation and position for the gripper is known, $\mathbf{v}^{gripper}$ has currently the length of $\mathbf{v}^{sup}_5$ ($\|\mathbf{v}^{gripper}\|_2 = \|\mathbf{v}^{sup}_5\|_2$) but the finger tips only have a maximum spread, y_{g_max}, so it is useful to reduce the length of $\mathbf{v}^{gripper}$. Equ 2.10 computes the factor $r_{gripper}$ for compression to reduce the length of $\mathbf{v}^{gripper}$ to half of y_{g_max}.

$$r_{gripper} = 0.5 \frac{\text{Grasp Width}}{\|\mathbf{v}^{gripper}\|_2} \tag{2.10}$$

With the gained compression factor $r_{gripper}$ and p^{TCP} the finger tip points p^{tip}_1 and p^{tip}_2 can be constructed by equations 2.11 and 2.11. These two points describes our finger tip positions for the gripper (figure 2.10, C). The z coordinates are calculated in 2.13 and used for both finger tips as the grasping vector is along the z axis.

$$p^{tip}_1(x,y) = p^{TCP} + r_{gripper} \cdot \mathbf{v}^{gripper}(x,y) \tag{2.11}$$

$$p^{tip}_2(x,y) = p^{TCP} - r_{gripper} \cdot \mathbf{v}^{gripper}(x,y) \tag{2.12}$$

$$p^{tip}_1(z) = p^{tip}_2(z) = (1 - dg) \cdot z_{max} - dg \cdot z_{min} \tag{2.13}$$

The distance between the two estimated finger tip points is exactly the Grasp Width y_{g_max} which is the maximum spread between the real fingers (equ. 2.14).

$$\|p^{tip}_1(x,y) - p^{tip}_2(x,y)\|_2 = y_{g\_max} \tag{2.14}$$

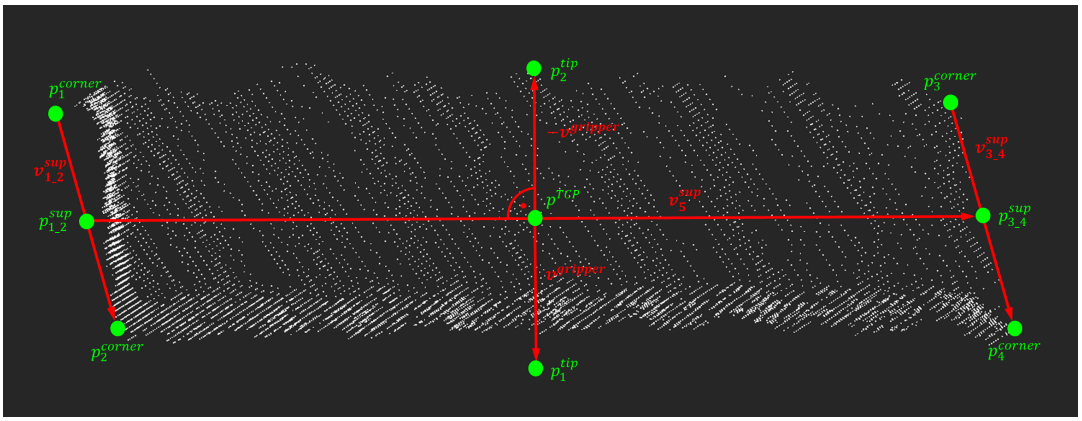
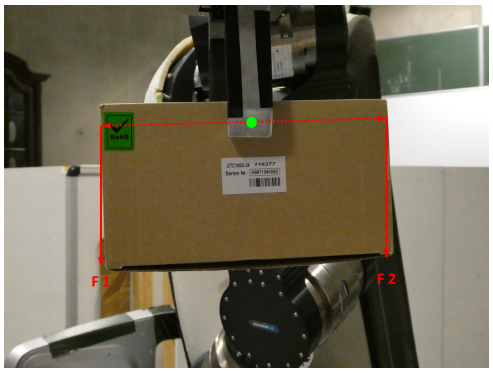
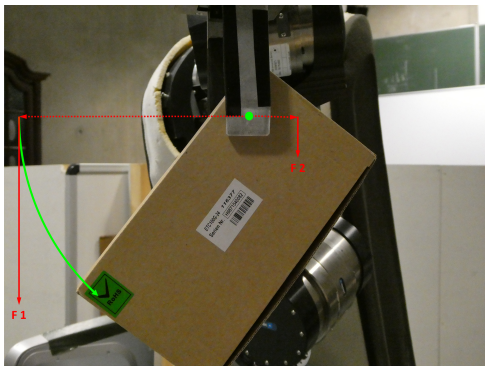


FIGURE 2.7: The calculated points (green dots) and the resulting vectors (red arrows). The paired corner points (p_1^{corner} and p_2^{corner}, p_3^{corner} and p_4^{corner}) spanning the supporting vector $\mathbf{v}_{1\_2}^{sup}$ and $\mathbf{v}_{3\_4}^{sup}$. The centre of these vectors represents supporting points, $p_{1\_2}^{sup}$ and $p_{3\_4}^{sup}$. v_5^{sup} is the vector between $p_{1\_2}^{sup}$ and $p_{3\_4}^{sup}$ and p^{TCP} is the estimated position for the TCP. By dropping a perpendicular vector $v^{gripper}$ from $\mathbf{v}_5^{sup}$, the grasp finger tip points are calculated as p_1^{tip} and p_2^{tip}. The euclidean distance between p_1^{tip} and p_2^{tip} is calculated as the grasp width (figure 2.1 A). p^{TCP}, $\mathbf{v}^{gripper}$, p_1^{tip} and p_2^{tip} describing the first gripper pose estimation.

The three points, p_1^{tip}, p_2^{tip} and p^{TCP} are the first grasp pose estimation based on geometrical information. Calculating the grasping points as close as possible to the centre of the object is one benefit. The approach assumes that the density of the tracked object is the same, so that the object has no change in density within. With a increasing deviation of the center of an object the probability of erroneous grasping increases as figure 2.8 demonstrates. Grasping an object is very similar to the physics of lever with an pivot element as the finger tips. Shifting away from the center of the object increases the difference between forces and increases the change of object movement in the gripper, if the resulting force exceeds the grasp stiction F_f (static friction). The approach reduces the probability of pose estimations shifted too far away from the center of the object and prevents from object movement which is not suitable for grasping.



(A) An ideal grasping position. The tips (green dots) are aligned to the estimated center of the object and the two resulting forces $F1$ and $F2$ (red arrows) based on the lever law are annihilating each other so no torque and therefore no motion will occur.



(B) At this grasp point one force is higher than the other resulting in a moment which rotates the object (here: rotation, green arrow) around the pivot elements(finger tips, green dot) leading to an erroneous grasping, the object might fall off the gripper for worst case. Such grasping position is prevented by the geometrical grasp pose estimation.

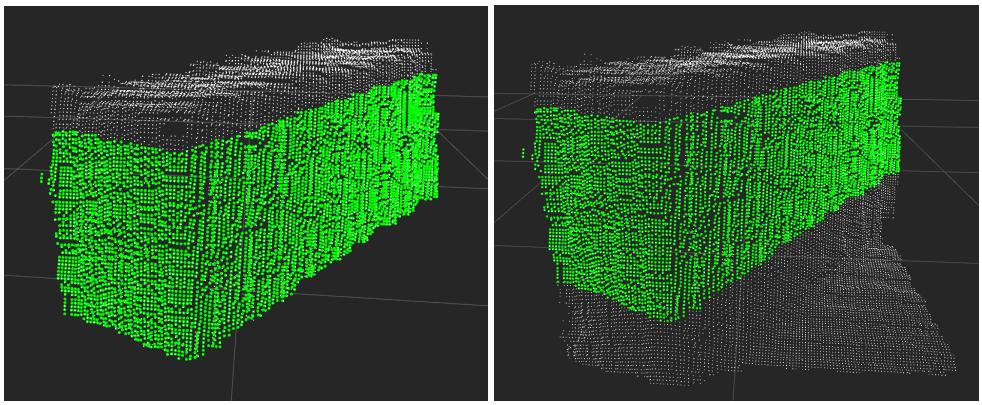
FIGURE 2.8: With the assumption that the density of every object remains the same it is necessary to involve the grasping position in relation with the geometry of an object. A ideal position is the centre of an object (A), while grasping more and more at edges might create failures (B) as a moment of force appears.

After the first grasp pose estimation the next calculations are subdivided into two parts, 2.2.3 and 2.2.4. Depending on pc_1^{cut} one calculation way is chosen. Regardless of the chosen part, the result is handled to 2.2.5 afterwards. The decision whether supporting points and subsequently the algorithm in 2.2.4 or the special alignment 2.2.3 will be used, is described in 2.2.2.

2.2.2 Support Point Evaluation

To check if there are usable supporting points for the surface pose estimation and prevent unstable grasps, first of all we cut off the top and bottom of pc_1^{cut} to obtain pc_2^{cut} (equ. 2.15), where Δ_{cut} is the cut off depth threshold (fig. 2.9) . Cutting off the top disables supporting points too close to the top while cutting off the bottom part of pc_1^{cut} prevents further grasp points estimations which might be difficult to reach as they are close to the tabletop the object stands on and risks a collision between finger tips and tabletop.

$$pc_2^{cut} = \{p \in pc_1^{cut} \mid z_{min} + \Delta_{cut} < p(z) < z_{max} - \Delta_{cut}\} \tag{2.15}$$



(A) pc_2^{cut} with the underlying pc_1^{cut} (B) pc_2^{cut} with the underlying pc_{init}

FIGURE 2.9: This PC, pc_2^{cut} (green dots), is a result of pc_1^{cut} (white dots) with further point reduction (A). pc_2^{cut} (green dots) in combination with pc_{init} (white dots) is shown in (B). The top was cut off in a certain distance from z_{max}. In this case there was no bottom cut off because the object depth is deeper than the Maximal Grasp Depth (pc_1^{cut} was reduced, figure 2.5), so a finger tip - tabletop collision is not feasible.

Equation 2.18 and 2.19 search for points p in pc_2^{cut} whose lengths between p and one finger tip (p_1^{tip}, p_2^{tip}) are within a given range (d_1^{sup} and d_1^{sup} with a parameter $\Phi_{sup,1} = \{x \in \mathbb{R} | 0 \leq x \leq 1\}$ 2.16 2.17). If there is no support point which fulfils the condition in both equations, as a result $pc_{tip_1}^{support} = pc_{tip_2}^{support} = \emptyset$, the grasp pose detection algorithm continues with 2.2.3. Otherwise $pc_{tip_1}^{support}$ and $pc_{tip_2}^{support}$ are carried to the surface grasp pose estimation.

$$d_1^{sup} = max(\|p_1^{tip} - p \in pc_2^{cut}\|_2) \tag{2.16}$$

$$d_2^{sup} = max(\|p_2^{tip} - p \in pc_2^{cut}\|_2) \tag{2.17}$$

$$pc_{tip_1}^{support} = \{p \in pc_2^{cut} \mid \|p(x,y) - p_1^{tip}(x,y)\|_2 < \Phi_{sup} \cdot d_1^{sup}\} \tag{2.18}$$

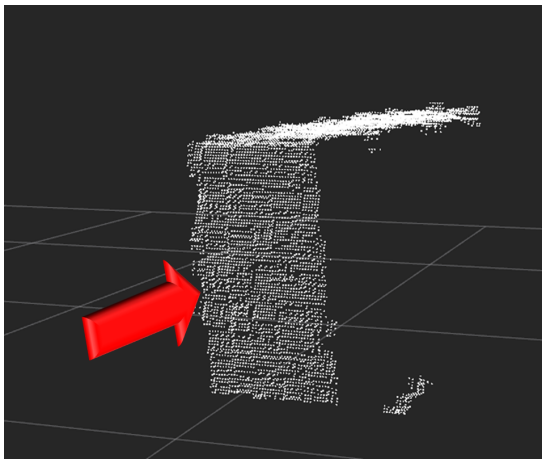
$$pc_{tip_2}^{support} = \{p \in pc_2^{cut} \mid \|p(x,y) - p_2^{tip}(x,y)\|_2 < \Phi_{sup} \cdot d_2^{sup}\} \tag{2.19}$$

2.2.3 Grasp Points for Special Viewpoint Alignments

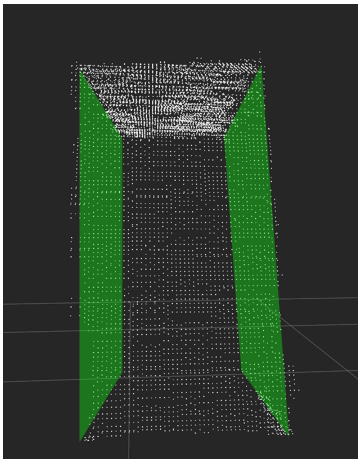
Due to the decision of no capable supporting point were found near p_1^{tip} or p_2^{tip}, this approach allows to cover the resulting special case of viewpoint constellations as seen in figure 2.10. If the viewpoint allows a direct front view to the object and the object sides geometry is equal or smaller than the front side there will be no usable information about these sides. With the assumption the autonomous system has a whole top surface visibility of an object, the correlation between the lack of side information and the top surface information allows an estimation of the side surface which must be at least a plane surface (figure 2.10, B), but it can also be any geometry as long as it stays within the geometry borders, created by the PC visibility, e.g. concave forms.

The top surface is not used for further grasp pose estimations or as support points as grasping on the top of an object does not guarantee a safe grasping position for the

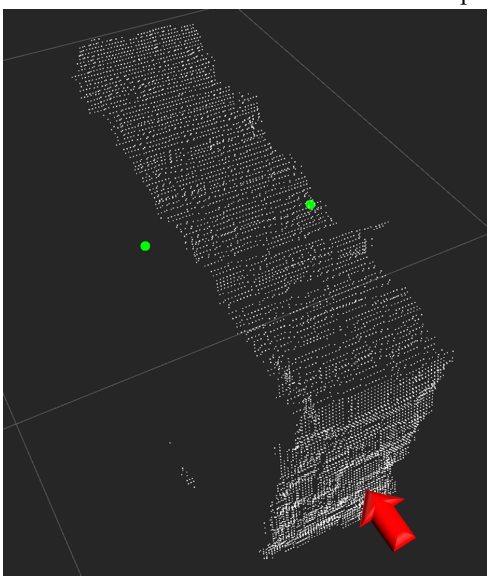
gripper [19] or as seen in pc_2^{cut}, where the top surface was cut off due to this safety restriction.



(A) Front view of the object. The viewpoint is represented as the red arrow. Because of the special viewpoint position there are no information about the object sides, only top and front information is accessible. The grasp pose estimation still finds graspable points.



(B) Occupying the viewpoint gives an different view of the object. The lack of side information in combination with knowledge of the top geometry we can assume there must be at least plane surfaces on the sides.



(C) The resulting grasp points for the finger tips based on the first estimation approach. The viewpoint is the same as in A, shown as red arrow.

FIGURE 2.10: A special case for the grasp pose estimation, as there are no information about the side geometry. The first pose estimation approach exploits a assumption of plane surfaces if there is a lack of side and given top geometry information.

As there are no further information or possible support points for a better grasp pose estimation and the approach take care of the weight distribution of an object, the estimated grasp points (p_1^{tip}, p^{TCP} and p_2^{tip}) are handled over for grasp evaluation 2.2.5.

2.2.4 Surface Grasp Pose Estimation

As a possible grasp pose was estimated in the section before, the second grasp point estimation uses local information close to the finger tips p_1^{tip}, p_2^{tip} to improve the quality of the grasp and adapt it to local surface shapes.

One of our requirements is to stay as close as possible to the center of the object to prevent grasping failures by virtue of the weight of the object. Due to this requirements the incoming point clouds $pc_{tip_1}^{support}$, $pc_{tip_2}^{support}$ are further reduced. For a first reduction we calculate distances between p_1^{tip} and all points in $pc_{tip_1}^{support}$ as well as the distances between p_2^{tip} and $pc_{tip_2}^{support}$, so we receive two vectors, $\mathbf{v}_1^{dst}$ (2.20) and $\mathbf{v}_2^{dst}$ (2.21).

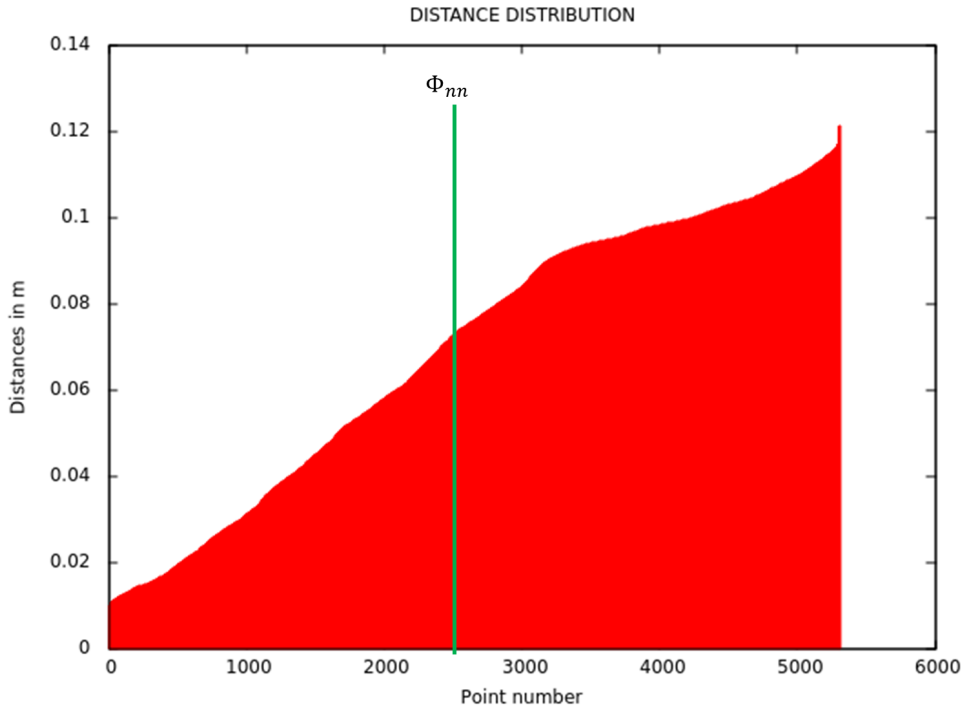


FIGURE 2.11: The distribution of distances between a finger tip and corresponding PC (e.g. p_1^{tip} and $pc_{tip_1}^{support}$). The green line shows the cut off, in this example $\Phi_{nn} = 0.5$, so only the closest 50% will remain in the resulting PC.

$$\mathbf{v}_1^{dst} = \begin{pmatrix} \|p_1^{tip} - pc_{tip_1}^{support}(1)\|_2 \\ \|p_1^{tip} - pc_{tip_1}^{support}(2)\|_2 \\ \vdots \\ \|p_1^{tip} - pc_{tip_1}^{support}(n)\|_2 \end{pmatrix} \tag{2.20}$$

$$\mathbf{v}_2^{dst} = \begin{pmatrix} \|p_1^{tip} - pc_{tip_2}^{support}(1)\|_2 \\ \|p_1^{tip} - pc_{tip_2}^{support}(2)\|_2 \\ \vdots \\ \|p_2^{tip} - pc_{tip_2}^{support}(n)\|_2 \end{pmatrix} \tag{2.21}$$

With the parameter $\Phi_{nn} = \{x \in \mathbb{R} | 0 \leq x \leq 1\}$ the vectors with the distances are reduced. The parameter is the percentage of how many points from $pc_{tip_1}^{support}$ and $pc_{tip_2}^{support}$ shall remain after the reduction (figure 2.11). For example a $\Phi_{nn} = 0.5$ means that the resulting point cloud $pc_{tip_1}^{nn}$ has $0.5 \cdot n(pc_{tip_1}^{support})$ points and $n(pc_{tip_2}^{nn}) = 0.5 \cdot n(pc_{tip_2}^{support})$. The remaining points are the points which have the closest distance to the finger tip. An example of a resulting point cloud is show in figure 2.12, the remaining distances are linked with the underlying points.

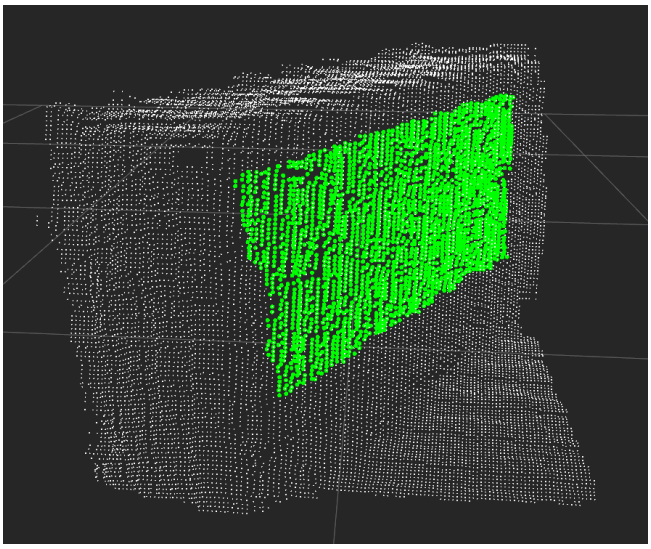


FIGURE 2.12: A point cloud (green dots) after the nearest neighbour algorithm has cut out all the points which exceed a certain distance to the corresponding finger tips

As the gripper has a given sparse extension there is no need for a high resolution grasp pose estimation, possible grasp points can be relative far apart from each other. If a gripper has a grasping surface of several centimetres there is no need to check every millimetre of the object surface for possible grasp positions as we assume there is no abrupt change in the surface. With the downsizing algorithm from [20] we reduce the amount of points in $pc_{tip_1}^{nn}$ and $pc_{tip_2}^{nn}$. The results, pc_1^{ds} and pc_2^{ds} are shown in figure 2.13.

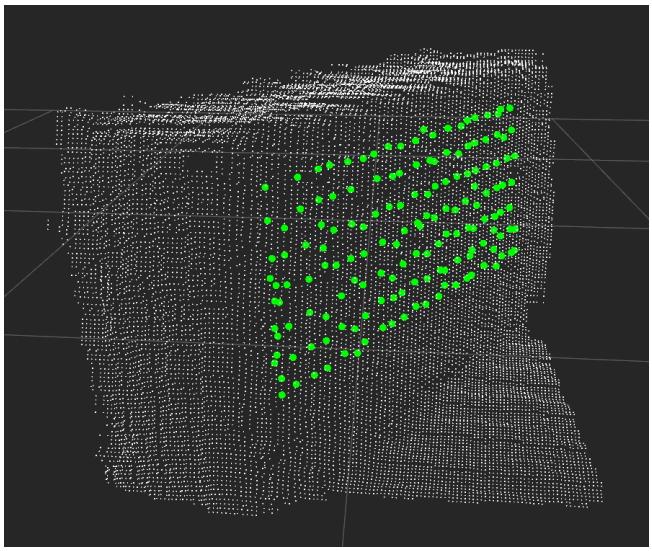


FIGURE 2.13: The green dots show pc_1^{ds} as the point cloud remaining after downsizing the neighbourhood PC $pc_{tip_1}^{nn}$. The white dots are the points in pc^{init}.

For the first filter the result of the principal curvature k_1 as maximal curvature, k_2 as minimal curvature are used. A function is also included in the PCL [21]. At a given point p on the investigated surface with the use of a normal vector $\mathbf{v}_n$ we can construct normal planes. The intersection between the planes and the surface results in a 2-dimensional space. Fitting a osculating circle inside this space and calculating the resulting inverse radius $\frac{1}{r}$, giving curvatures of the point p. A deeper explanation is found in [22]. For every point p in pc_1^{ds}, k_{res} is calculated (equ. 2.22). Table 2.1 shows the different geometric surface shapes based on k_1 and k_2.

	$k_1 < 0$	$k_1 = 0$	$k_1 > 0$
$k_2 < 0$	Concave ellipsoid	Concave cylinder	Hyperboloid surface
$k_2 = 0$	Concave cylinder	Plane	Convex cylinder
$k_2 > 0$	Hyperboloid surface	Convex cylinder	Convex ellipsoid

TABLE 2.1: Surface shape classes based on k_1 and k_2 [23].

Ideal surfaces for the gripper to grasp are surfaces with the same geometrical shape as the finger gripper. In this case we talk about a flat surface as our grasping areas are flat. Hence the best surface is described if k_1 and k_2 are both 0, and this is our only condition as we just want to calculate the curvature and don't care if it is convex or concave. Wit the sum of the absolute value of both parameters we gain a feasible result k_{res} for each point p. In conclusion the curvature calculation output is a vector $\mathbf{v}_1^{k_{res}} \in \mathbb{R}^{n(pc_1^{ds})\times 1}$ and similar $\mathbf{v}_2^{k_{res}}$ (2.23). A chart how the curvatures are occurring shows figure 2.14, based on pc_1^{ds} (figure 2.13).

$$k_{res} = |k1| + |k2| \tag{2.22}$$

$$\mathbf{v}_1^{k_{res}} = \begin{pmatrix} |k1(pc_1^{ds}(1))| + |k2(pc_1^{ds}(1))| \\ |k1(pc_1^{ds}(2))| + |k2(pc_1^{ds}(2))| \\ \vdots \\ |k1(pc_1^{ds}(n))| + |k2(pc_1^{ds}(n))| \end{pmatrix} \tag{2.23}$$

$\mathbf{v}_2^{k_{res}}$ is calculated the same way, with $pc_2^{ds}(1)$ instead of $pc_1^{ds}(1)$.

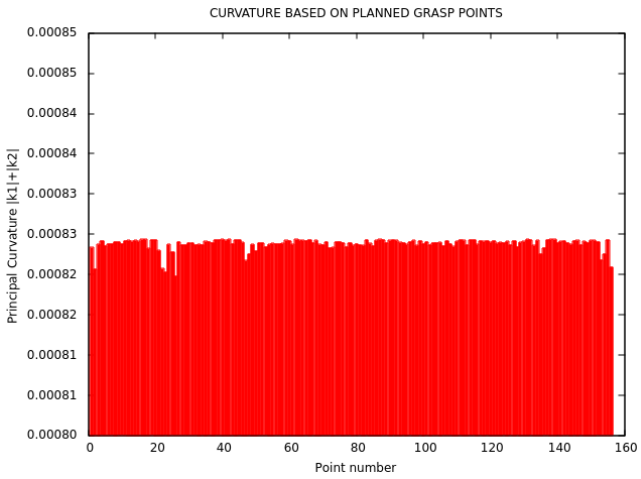


FIGURE 2.14: The vector $\mathbf{v}_1^{k_{res}}$ with its entries $\mathbf{v}_1^{k_{res}}(1), ..., \mathbf{v}_1^{k_{res}}(n)$). As expected in this example, the values as well as the deviation is very low.

Similar to Φ_{nn}, $\Phi_{curv} = \{x \in \mathbb{R} | 0 \leq x \leq 1\}$, is a parameter to reduce the current PCs pc_1^{ds} and pc_2^{ds} by a given percentage, so $n(pc_1^{curv}) = \Phi_{curv} \cdot n(pc_{tip_1}^{ds})$ and $n(pc_2^{curv}) = \Phi_{curv} \cdot n(pc_{tip_2}^{ds})$. The PCs contain the percentage of points with the lowest curvature (figure 2.15).

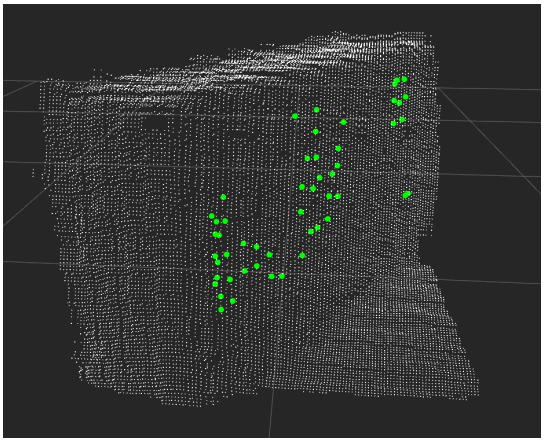


FIGURE 2.15: The green dots depict pc_1^{curv}. It is the point cloud remaining after downsizing the neighbourhood PC $pc_{tip_1}^{nn}$. The white dots are the points in pc^{init}.

The result of the previous calculations is a point cloud, reduced by several parameters and filtered by the curvature to gain the best plane surfaces. The next task is to examine the obliqueness of the planes close to the points in pc_1^{curv} and pc_2^{curv}. Since our grasping vector is the z axis and our approach covers antipodal grasping with a parallel gripper and utilizes friction ($F_f = \mu \cdot F_{r1}$ with $F_{r1} = F_{gripper} \cdot cos(\alpha)$ and μ as the coefficient of friction) to hold objects, the best grasping surface is a plane parallel to the gripper fingers. As described in figure 2.16, plane surfaces would result in an alpha of 0, so the maximum force $F_{r1} = F_{gripper}$ can be transferred from the gripper to the object, resulting in maximum friction force. Besides the weight force $F_G = mg$, $F_Z = sin(F_{gripper})$ appears in the same direction, increasing the total force ($F_G + F_Z$) which needs to be compensated by F_f in order for a stable grasp.

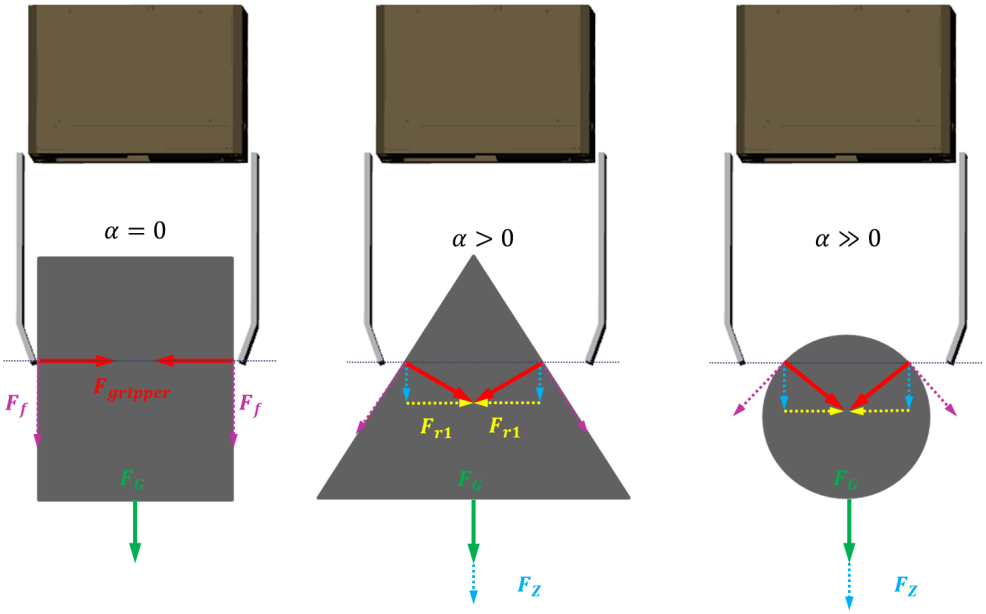


FIGURE 2.16: As the finger tips are closed in direction of the blue dashed lines, the grasping forces $F_{gripper}$ (red arrows) are orthogonal to the contact point surface of the object. The resulting angle α between $F_{gripper}$ and the blue dashed line affects the resulting friction force F_f (purple dashed line) by F_{r1} (yellow dashed lines). In addition $F_{gripper}$ amplifies the weight force, F_G (green arrow) by F_Z (blue dashed arrows). In conclusion, a higher α decreases the resulting friction force F_f while increasing the total force $F_g + F_Z$ which pulls the object out of the gripper. Therefore a parallel plane is the safest grasping surface.

Based on this cognition it is necessary to calculate the obliqueness of our remaining points in our point clouds. For the approach our initial assignment is to find a certain amount of points given from pc_1^{cut} near to the points given from pc_1^{curv} and pc_2^{curv}. The parameter $\Phi_{ac} = \{x \in \mathbb{R} | 0 \leq x \leq 1\}$ describes the percentage of points from $n(pc_1^{cut})$ which shall be linked to the points in pc_1^{curv} and pc_2^{curv}, so every point $p \in pc_1^{curv}$ has $\Phi_{ac} \cdot n(pc_1^{cut})$ related points (The same counts for the points in pc_2^{curv}) as shown in equation 2.25. During further explanations we will mostly talk about processing pc_1^{curv}, but it also applied to pc_2^{curv} but with different indexation (2 instead

of 1). The linked points are the nearest neighbours to the related grasping point, as they describe the surface of the object close to the grasping point. It is not useful to analyse the whole object surface, instead, to increase calculation speed, we just check the local surface (figure 2.17).

$$pc_{1,k}^{ac} = \{p \in pc_2^{cut} \mid \|p(x,y,z) - pc_1^{curv}(k)\|_2 < d_{\Phi_{ac}}\} \quad \text{with } k = \{1,2,3,...,n(pc_1^{curv})\} \tag{2.24}$$

The equation to find the nearest neighbours is shown in equ. 2.24. Φ_{ac} defines the border as a quantile. Every distance between the points $p^{pc_2^{cut}}$ and $p^{pc_1^{curv}}$ which is under the quantile limit $d_{\Phi_{ac}}$ is linked to the new PC pc_1^{ac}, leading to the vector $\mathbf{v}_1^{ac}$ (2.25).

$$\mathbf{v}_1^{ac} = \begin{pmatrix} pc_{1,1}^{ac} \\ pc_{1,2}^{ac} \\ \vdots \\ pc_{1,n}^{ac} \end{pmatrix} \tag{2.25}$$

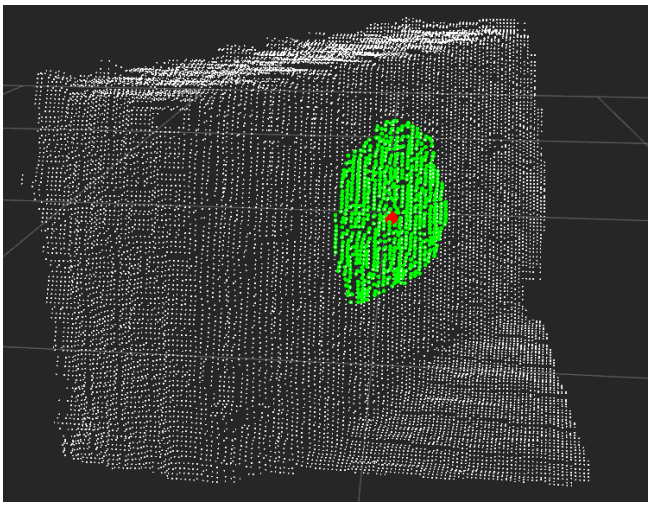


FIGURE 2.17: Points from pc_2^{cut} (green dots) which are the nearest neighbours around one point $p_1^{curv} \in pc_1^{curv}$ (red dot).

We define a random vector $\mathbf{v}_{rand}(x,y,z)$. As every entry from $\mathbf{v}_1^{ac}$ (2.25) is a point cloud, we receive a vector $\mathbf{v}_1^{\alpha}$ for every entry, so we obtain a matrix $\mathbf{M}_1^{\alpha} \in K^{n(v_1^{ac}) \times n(pc_1^{ac})}$ and $K = \{x \mid 0 \leq x < \pi\}$

$$\alpha(v_1, v_2) = acos \frac{v^1 \cdot v^2}{\|v^1\| \|v^2\|} \tag{2.26}$$

2.26 is used to calculate the angle between general vectors, 2 describes the algorithm to gain the set of angle vectors. A visual description is found in 2.18.

Calculate angles (pc_1^{curv}, pc_1)
create random vector $\mathbf{v}_{rand}(x, y, z)$
forall *point clouds i in* v_1^{ac} **do**
  forall *points j in* $v_1^{ac}(i)$ **do**
    $pc_1 = \mathbf{v}_1^{ac}(i)$
    $\mathbf{v}_{angle} = pc_1(j) - pc_1^{curv}(j)$
    $\mathbf{v}_{1,i}^{\alpha}(j) = 2.26(\mathbf{v}_{rand}(x, y), \mathbf{v}_{angle}(x, y))$
  end
  $\mathbf{M}_1^{\alpha}(i) = \mathbf{v}_1^{\alpha T}$
end
return $\mathbf{M}_1^{\alpha}$

Algorithm 2: The algorithm calculates the angle between a random vector and vectors spanned by one grasping point $p \in pc_1^{curv}$ and the nearest points from pc_1^{ac}. The output is the matrix $\mathbf{M}_1^{\alpha}$.

The main idea behind the angle calculation is a parameter to describe the distribution of points in correlation with obliqueness. As seen in figure 2.20, the obliqueness seen from one direction changes the surface from this viewpoint and therefore changes the spatial appearance probability. Analysing the angles in $\mathbf{M}_1^{\alpha}$ introduces a useful parameter for obliqueness characterization.

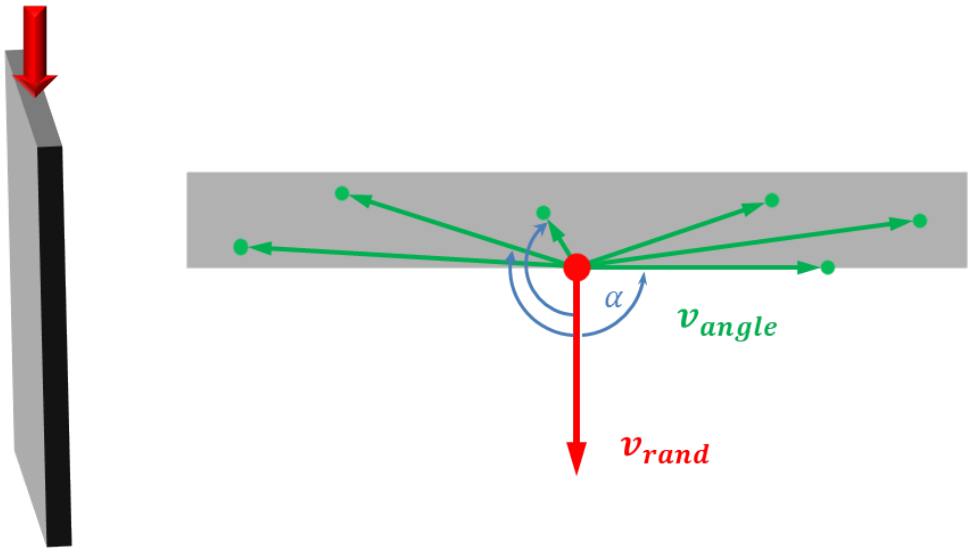


FIGURE 2.18: The angles α (blue arrows) are calculated between $\mathbf{v}_{rand}$ (red arrow), the random vector, and several vectors $\mathbf{v}_{angle}$ spanned between $p_1^{curv} \in pc_1^{curv}$ (red point) and $p_1^{cut} \in pc_1^{cut}$ (green points). The viewpoint is the red arrow (object is seen from above).

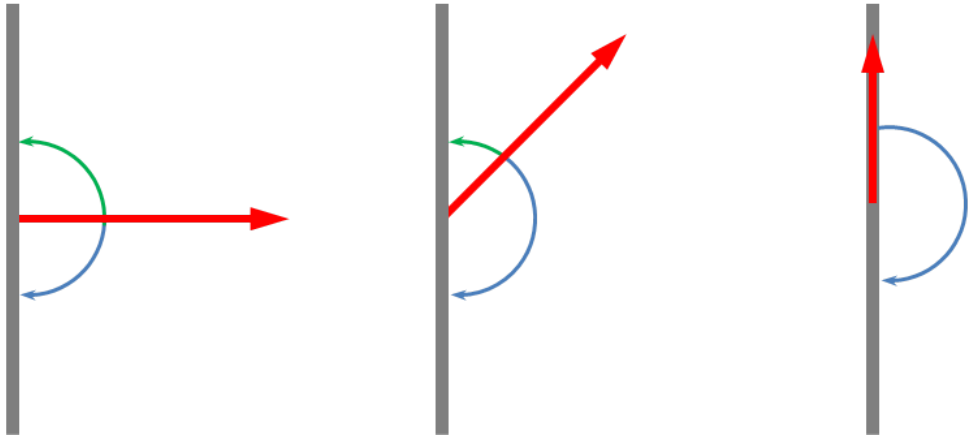
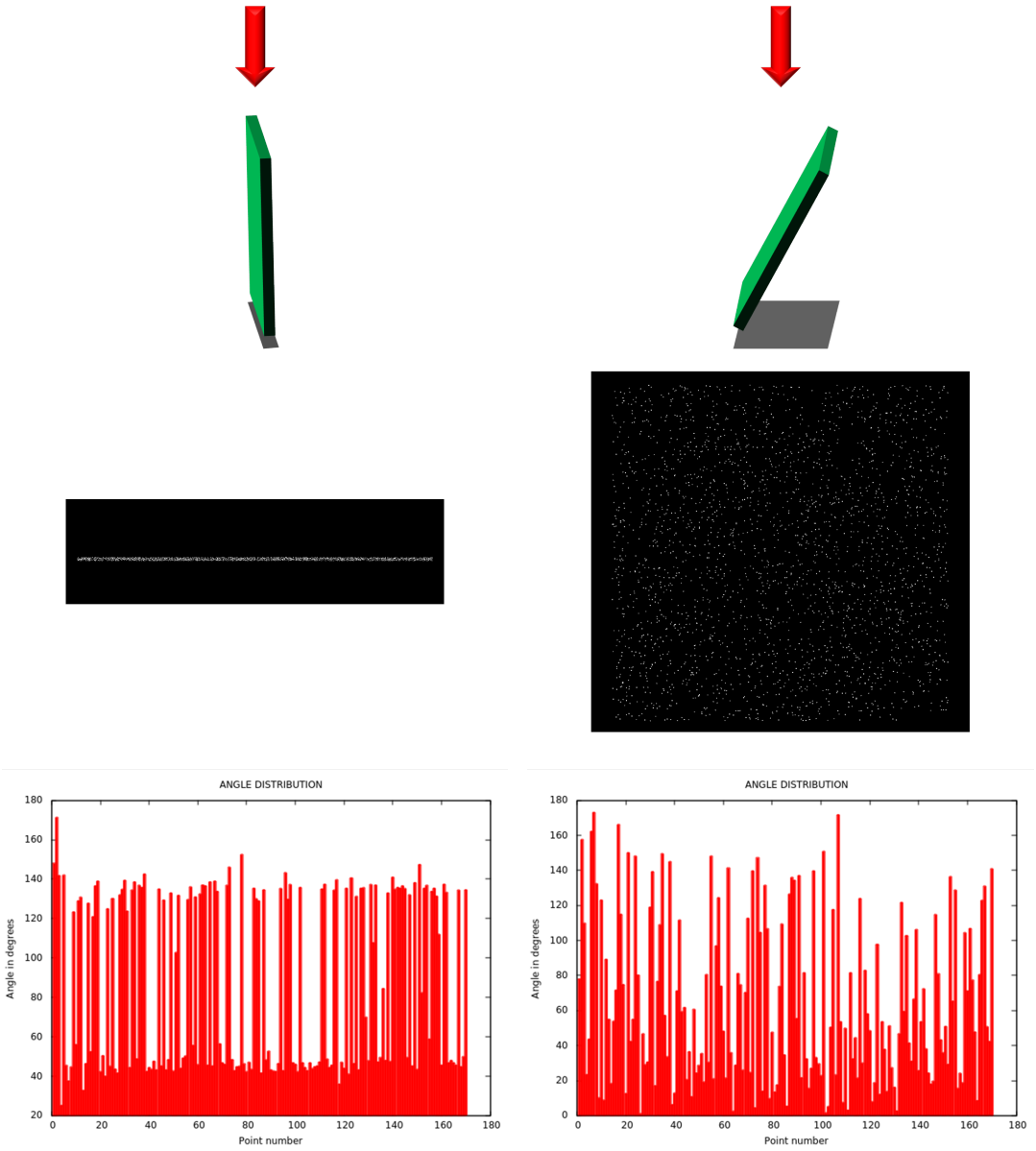


FIGURE 2.19: Depending on $\mathbf{v}_{rand}$ the resulting angles lie between 0 and π. In the left figure, both angles (green and blue arrows) between the red arrow and the grey plane have the value $\frac{\pi}{2}$. The middle figure the green arrow has a lower angle than the blue arrow. The left figures result is 0 since the red arrow and grey plane are parallel. The blue arrow results in π as they are anti parallel. We exploit the feature that the sum of both angles (green and blue) is π, if the surface is straight as in figure 2.18 and deviates more and more with rising obliqueness or the local surface describes a kink.



(A) Due to the low obliqueness based on the viewpoint (red arrow), the resulting points are distributed along two small corridors. The angle calculation leads to a clustering near two angle values.

(B) As the green area is obliqued, the resulting shadow (grey area under the green area) is expanded, the points are distributed in a wider area, angles are not clustered in two specific values, rather the values approach dispersed.

FIGURE 2.20: Based on the obliqueness the resulting distribution of points differs. The effect is visible in the appearance of the calculated angle values in $\mathbf{M}_1^\alpha$.

The next steps 2.27 to 2.40 are using one row from M_1^α to receive $v_1^\alpha = \mathbf{M}_1^{\alpha T}(i)$, but all rows from the Matrix must be calculated to gain the full quality matrix $\mathbf{M}_1^Q$ 2.41. In addition all rows from $\mathbf{M}_2^\alpha$ are processed similar and leads to $\mathbf{M}_2^Q$.

$$\bar{x}_c = \frac{1}{n(\mathbf{v}_1^\alpha)} \sum_{i=1}^{n(\mathbf{v}_1^\alpha)} \mathbf{v}_1^\alpha(i) \tag{2.27}$$

With $\bar{x}_c$ as our separator $\mathbf{v}_1^\alpha$ is separated into a upper and lower angle vector ($\mathbf{v}_{1,u}^\alpha$ 2.28, $\mathbf{v}_{1,l}^\alpha$ 2.33).

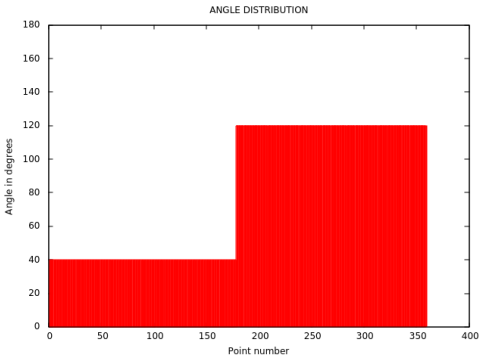
$$\mathbf{v}_{1,u}^{\alpha} = \{\alpha \in \mathbf{v}_1^{\alpha} \mid \alpha > \bar{x}_c\} \tag{2.28}$$

Using $\mathbf{v}_{1,u}^{\alpha}$ as input for 2.29, we calculate the arithmetic mean value for the upper part of our angle distribution.

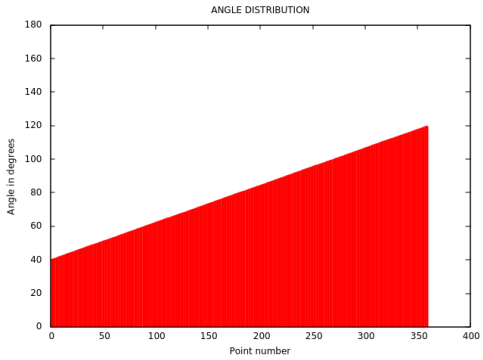
$$\bar{x}_{1,u} = \frac{1}{n(\mathbf{v}_{1,u}^{\alpha})} \sum_{i=1}^{n(\mathbf{v}_{1,u}^{\alpha})} \mathbf{v}_{1,u}^{\alpha}(i) \tag{2.29}$$

Beside the arithmetic mean value the empirical variance is a necessary value for calculating the quality of the grasping point (2.30). It is needed to have an additional value to prove the obliqueness, only the mean might cause several angle distributions to be weighted wrong (figure 2.21).

$$\tilde{s}_{1,u} = \sqrt{\frac{1}{n(\mathbf{v}_{1,u}^{\alpha})} \sum_{i=1}^{n(\mathbf{v}_{1,u}^{\alpha})} (\mathbf{v}_{1,u}^{\alpha}(i) - \bar{x}_{1,u})^2} \tag{2.30}$$



(A) Angles as they occur as the constellation is like 2.19, middle figure. In this case, the mean values $\bar{x}_c = 80°$, $\bar{x}_{1,u} = 120°$ and $\bar{x}_{1,l} = 40°$.



(B) A different constellation, the mean value $\bar{x}_c = 80°$, $\bar{x}_{1,u} = 120°c$ and $\bar{x}_{1,l} = 40°$ is the same as in figure (A)

FIGURE 2.21: While the means of both A and B is the same and there is no way to distinguish between them only by its mean values, the obliqueness is different, as shown in 2.8. To distinguish between these occurring angle distributions the empirical variance are suitable parameter, since in A it is 0 while in B it differs from 0.

As we have two mean values, $\tilde{s}_{1,u}$ and $\tilde{s}_{1,l}$, it is useful to calculate the empirical coefficient of variation, $v_{1,u}$ (2.31) and $v_{1,l}$ (2.36) in order to achieve a better comparison and be able to cover scenarios as seen in figure 2.21 and incorporate them in our calculations.

$$v_{1,u} = \frac{\tilde{s}_{1,u}}{\bar{x}_{1,u}} \tag{2.31}$$

Finally, to compare both, the mean and empirical coefficient of variation, the $v_{1,u}$ as well as $v_{1,l}$ are normed, so they can only occur between 0 and 1 ($v_{1,u}^{*} = 2.32$, $v_{1,l}^{*} = 2.37$).

$$v_{1,u}^{*} = \sqrt{\frac{v_{1,u}}{n(\mathbf{v}_{1,u}^{\alpha}) - 1}} \text{ with } 0 \leq v_{1,u}^{*} \leq 1 \tag{2.32}$$

Equations 2.33 to 2.37 do the same calculations as 2.28 to 2.32 but only the angles underneath the separator value $\bar{x}_c$.

$$\mathbf{v}_{1,l}^{\alpha} = \{\alpha \in \mathbf{v}_1^{\alpha} \mid \alpha < \bar{x}_c\} \tag{2.33}$$

$$\bar{x}_{1,l} = \frac{1}{n(\mathbf{v}_{1,l}^{\alpha})} \sum_{i=1}^{n(\mathbf{v}_{1,l}^{\alpha})} \mathbf{v}_{1,l}^{\alpha}(i) \tag{2.34}$$

$$\tilde{s}_{1,l} = \sqrt{\frac{1}{n(\mathbf{v}_{1,l}^{\alpha})} \sum_{i=1}^{n(\mathbf{v}_{1,l}^{\alpha})} (\mathbf{v}_{1,l}^{\alpha}(i) - \bar{x}_{1,l})^2} \tag{2.35}$$

$$v_{1,l} = \frac{\tilde{s}_{1,l}}{\bar{x}_{1,l}} \tag{2.36}$$

$$v_{1,l}^{*} = \sqrt{\frac{v_{1,l}}{n(\mathbf{v}_{1,l}^{\alpha}) - 1}} \text{ with } 0 \leq v_{1,l}^{*} \leq 1 \tag{2.37}$$

The closer the sum of $\bar{x}_{1,u}$ and $\bar{x}_{1,l}$ relies to π (see figure 2.19) the less the area is oblique. In another formulation, to find the straightest surface $\pi - \bar{x}_{1,u} + \bar{x}_{1,l}$ must be as close as possible to 0. For a better comparison the value is normed (2.38, $max(\bar{x}_{1,u} + \bar{x}_{1,l}) = 2\pi$, $min(\bar{x}_{1,u} + \bar{x}_{1,l}) = 0$).

$$\Delta\bar{x} = \frac{|\pi - (\bar{x}_{1,u} + \bar{x}_{1,l})|}{\pi} \text{ with } 0 \leq \Delta\bar{x} \leq 1 \tag{2.38}$$

Equation 2.39 unites the two normed variant coefficients ($v_{1,l}^{*}, v_{1,l}^{*}$) with equal weights.

$$v_1^{*} = 0.5(v_{1,u}^{*} + v_{1,l}^{*}) \tag{2.39}$$

Finally, Q_1 is calculated as the quality score (or quality value) of the analysed surface relied to the grasping point (2.40). Our assumption is that the probability of a scenario like in figure 2.21 is low and the variance is sensitive to noise, so we weight v_1^{*} with $\frac{1}{3}$ and $\Delta\bar{x}$ with $\frac{2}{3}$ (The noise may be also affect the mean value but negative noise values might annihilate positive, but the squaring in the variance (2.30) dissolves such effects).

$$Q = \frac{v_1^{*} + 2 \cdot \Delta\bar{x}}{3} \tag{2.40}$$

For every grasping point p_1^{curv} in pc_1^{curv} the quality scores are calculated. As a result we obtain a quality matrix $\mathbf{M}_1^{Q}$ (2.41) with $\mathbf{M}_1^{Q}(1,1) \geq \mathbf{M}_1^{Q}(2,1) \geq \mathbf{M}_1^{Q}(3,1) \geq ... \geq \mathbf{M}_1^{Q}(n,1)$, so the Matrix is sorted ascending using Q. The function f_q describes all equation steps from 2.24 to 2.40 with the points of pc_1^{curv} respective pc_2^{curv} and pc_2^{cut}.

$$\mathbf{M}_1^{Q} = \begin{pmatrix} Q_1 & p \in pc_1^{curv} \mid f_q(pc_1^{curv}(1), pc_2^{cut}) = Q_1 \\ Q_2 & p \in pc_1^{curv} \mid f_q(pc_1^{curv}(2, pc_2^{cut})) = Q_2 \\ \vdots & \vdots \\ Q_n & p \in pc_1^{curv} \mid f_q(pc_1^{curv}(n, pc_2^{cut})) = Q_3 \end{pmatrix} \tag{2.41}$$

Like $\mathbf{M}_1^Q$, $\mathbf{M}_2^Q$ is calculated similar but with the input Matrix, $\mathbf{M}_2^{ac}$. The current Matrix $\mathbf{M}_1^Q$ contains p_1^{curv} and $\mathbf{M}_2^Q$ p_2^{curv}, which are the corresponding points for the quality scores in the matrices.

Furthermore we need to prove the distance between the remaining grasping points and the first estimation, p_1^{tip} and p_2^{tip} (figure 2.7) in order to satisfy our weight requirement (grasp our object as close to the center of weight as possible). To incorporate this requirement to our current quality scores, we calculate the euclidean distances between p_1^{tip}, our first estimation, and the current remaining grasp points (equ. 2.42).

$$\mathbf{v}_{Q,1}^{dst} = \begin{pmatrix} \|\mathbf{M}_1^Q(1,2)(x,y,z) - p_1^{tip}(x,y,z)\| \\ \|\mathbf{M}_1^Q(2,2)(x,y,z) - p_1^{tip}(x,y,z)\| \\ \vdots \\ \|\mathbf{M}_1^Q(n,2)(x,y,z) - p_1^{tip}(x,y,z)\| \end{pmatrix} \tag{2.42}$$

To norm this distances the minimum (dst_1^{min}, 2.43) and maximum (dst_1^{max}, 2.44) occurring values in $\mathbf{v}_{Q,1}^{dst}$ are extracted.

$$dst_1^{min} = min(\mathbf{v}_{Q,1}^{dst}) \tag{2.43}$$

$$dst_1^{max} = max(\mathbf{v}_{Q,1}^{dst}) \tag{2.44}$$

This two values combined with $\mathbf{v}_{Q,1}^{dst}$ creates $\mathbf{v}_{Q,1}^{dst^*}$ (2.45), a vector within all distances are normed to weights between 0 and 1.

$$\mathbf{v}_{Q,1}^{dst^*} = \begin{pmatrix} \frac{\mathbf{v}_Q^{dst}(1) - dst_1^{min}}{dst_1^{max} - dst_1^{min}} \\ \frac{\mathbf{v}_Q^{dst}(2) - dst_1^{min}}{dst_1^{max} - dst_1^{min}} \\ \vdots \\ \frac{\mathbf{v}_Q^{dst}(n) - dst_1^{min}}{dst_1^{max} - dst_1^{min}} \end{pmatrix} \tag{2.45}$$

The vector values are added to the current quality matrix $\mathbf{M}_1^Q$ to obtain a enhanced matrix $\mathbf{M}_1^{Q^*}$ where the obliqueness and the distance to the estimated finger tip p_1^{tip} are combined as a final score for the grasp pose estimation (equ 2.46). The distances are weighted half of the obliqueness as we assume the distances from the object weight center is easier to compensate with a gripper than the obliqueness of the surface. Figure 2.22 shows the resulting quality scores based on the grasping points of p_1^{curv}.

$$\mathbf{M}_1^{Q^*} = \begin{pmatrix} 0.5\mathbf{v}_{fin,1}^{dst^*}(1) + \mathbf{M}_1^Q(1,1) & \mathbf{M}_1^Q(1,2) \\ 0.5\mathbf{v}_{fin,1}^{dst^*}(2) + \mathbf{M}_1^Q(2,1) & \mathbf{M}_1^Q(2,2) \\ \vdots & \\ 0.5\mathbf{v}_{fin,1}^{dst^*}(n) + \mathbf{M}_1^Q(n,1) & \mathbf{M}_1^Q(n,2) \end{pmatrix} \tag{2.46}$$

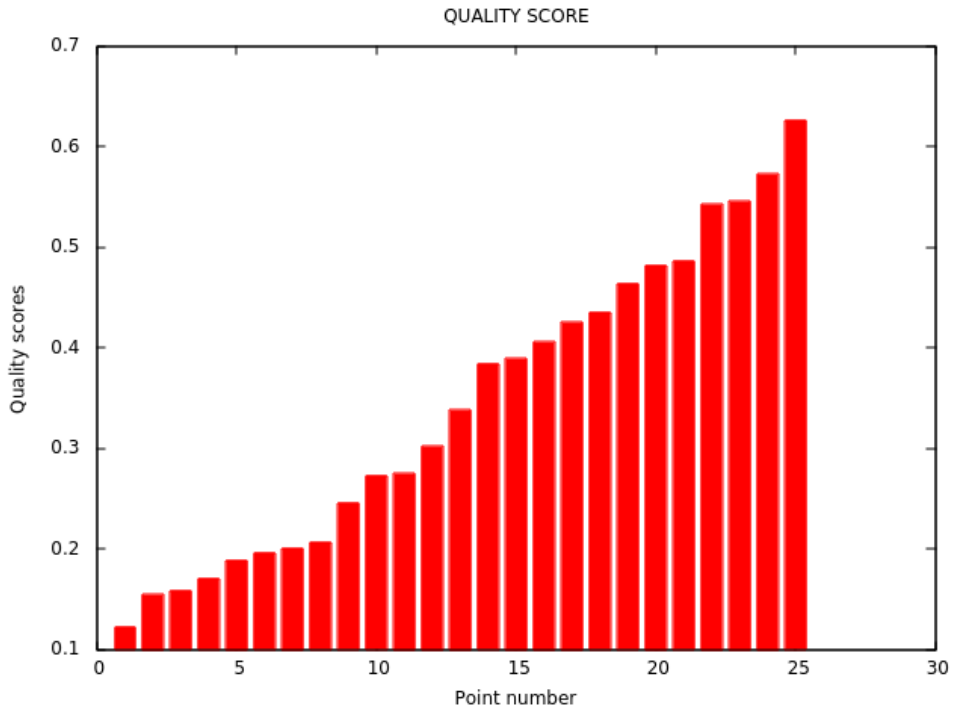


FIGURE 2.22: The quality scores based on their grasping points from pc_1^{curv}. For every $p \in pc_1^{curv}$ the quality score is calculated and is a combination of the difference between upper and lower mean values, empirical coefficients of variation and the distance to p_1^{tip}. The lower the scores the better the point is graspable, preventing erroneous grasping.

A final filter reduces the amount of remaining grasp points to n_{max}^{grasps}, so the the result are the new point clouds pc_1^{spe} based on $\mathbf{M}_1^{Q^*}$ and pc_2^{spe} based on $\mathbf{M}_2^{Q^*}$ with $n(pc_1^{spe}) = n(pc_2^{spe}) = n_{max}^{grasps}$. It might be possible, that $n_{max}^{grasps} > n(\mathbf{M}_1^{Q^*})$. In this case $n_{max}^{grasps} = n(\mathbf{M}_1^{Q^*})$ (same counts for $n(\mathbf{M}_2^{Q^*})$). All points in pc_1^{spe} are the grasp points stored in $\mathbf{M}_1^{Q^*}$ ($\mathbf{M}_1^{Q^*}(2,1), ..., \mathbf{M}_1^{Q^*}(2,n)$) with the lowest n_{max}^{grasps} quality scores in $\mathbf{M}_1^{Q^*}$ ($\mathbf{M}_1^{Q^*}(1,1), .., \mathbf{M}_1^{Q^*}(1,n)$). For an example, if $n_{max}^{grasps} = 10$ the 10 points are stored which are linked with the lowest 10 quality scores in $\mathbf{M}_1^{Q^*}$.

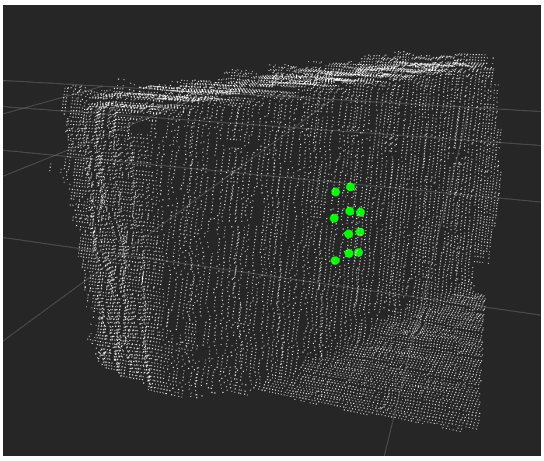


FIGURE 2.23: The remaining n_{max}^{grasps} grasp points (green dots) in pc_1^{spe} filtered by the quality score from $\mathbf{M}_1^{Q^*}$.

Before the grasp points are evaluated and validated if the gripper can reach them, the remaining n_{max}^{grasps} amount of grasp points is aligned to the surface of pc_2^{cut}, resulting in new grasp points (p_e^{TCP}, $p_{1,e}^{tip}$, $p_{2,e}^{tip}$). The requirements for this alignment are a certain distance d_{tar} to the surface in order to have space for the fingers and the vector of the new resulting grasp points (vector between $p_{1,e}^{tip}$ and $p_{2,e}^{tip}$) must be parallel to the vector spanned between p_1^{tip} and p_2^{tip} (algorithm 3).

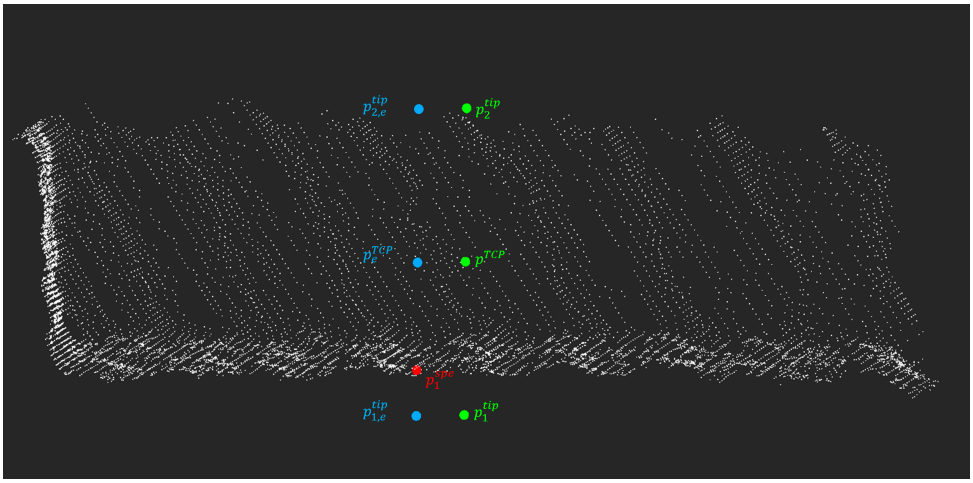


FIGURE 2.24: The adjustment of p^{TCP}, p_1^{tip} and p_2^{tip} (green points). Based on algorithm 3, the initial points p_1^{tip}, p^{TCP}, p_2^{tip} are adjusted to the surface by shifting the point p_1^{spe} (where p_1^{spe} is one random point out of pc_1^{spe}) by a certain distance d_{tar} in the vector direction of $\mathbf{v}^{gripper}$ to obtain p_e^{TCP}, $p_{1,e}^{tip}$ and $p_{2,e}^{tip}$

Align grasping points to surface $(p_1^{tip}, p^{TCP}, p_2^{tip}, pc_1^{spe}, pc_2^{spe})$
forall *points i in* pc_1^{spe} **do**
  $p_{1,e}^{tip} = pc_1^{spe}(i) + (p_1^{tip} - p^{TCP})\Big(\frac{d_{tar}}{\|p^{TCP}(x,y) - p_1^{tip}(x,y)\|} + 1\Big)$
  $p_e^{TCP} = p_{1,e}^{tip} + p^{TCP} - p_1^{tip}$
  $p_{2,e}^{tip} = p_{1,e}^{tip} + 2(p^{TCP} - p_1^{tip})$
  $\mathbf{M}^{spe}(i,1) = p_{1,e}^{tip}$
  $\mathbf{M}^{spe}(i,2) = p_e^{TCP}$
  $\mathbf{M}^{spe}(i,3) = p_{2,e}^{tip}$
end
forall *points i in* pc_2^{spe} **do**
  $p_{1,e}^{tip} = pc_2^{spe}(i) + (p_2^{tip} - p^{TCP})\Big(\frac{d_{tar}}{\|p^{TCP}(x,y) - p_2^{tip}(x,y)\|} + 1\Big)$
  $p_e^{TCP} = p_{2,e}^{tip} + p^{TCP} - p_2^{tip}$
  $p_{2,e}^{tip} = p_{2,e}^{tip} + 2(p^{TCP} - p_2^{tip})$
  $\mathbf{M}^{spe}(n(pc_1^{spe}) + i, 1) = p_{1,e}^{tip}$
  $\mathbf{M}^{spe}(n(pc_1^{spe}) + i, 2) = p_e^{TCP}$
  $\mathbf{M}^{spe}(n(pc_1^{spe}) + i, 3) = p_{2,e}^{tip}$
end
return $\mathbf{M}^{spe}$

Algorithm 3: This algorithm aligns the estimated points $p_1^{tip}, p_{TCP}, p_2^{tip}$ to the calculated points in pc_1^{spe}, pc_2^{spe}, where d_{tar} is the security distance between the new grasping point and the surface. M^{spe} is the output, a matrix with the needed points to describe the gripper pose. See figure 2.24 for a overview of the points. i is the iteration index.

The matrix $\mathbf{M}^{spe}$ (equ. 2.47) contains the grasping points which are necessary to describe the TCP and finger tips, therefore the information needed for the gripper pose. The previous calculations were affected by a distinguish between p_1^{tip} and p_2^{tip}, $\mathbf{M}^{spe}$ now contains $n(pc_1^{spe}) + n(pc_1^{spe})$ entries.

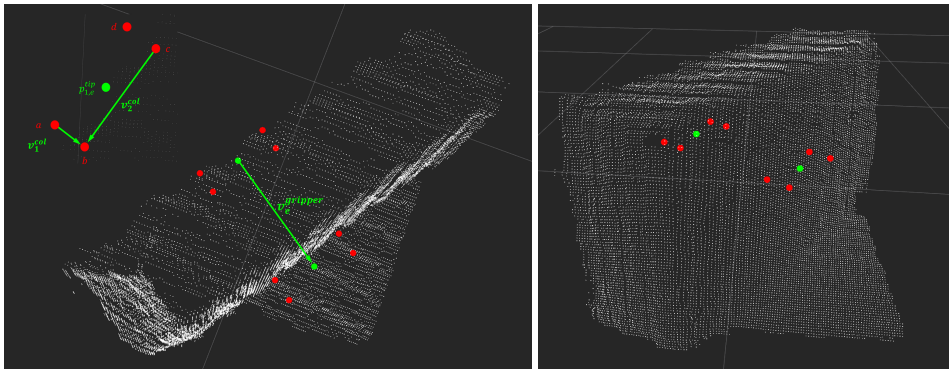
$$\mathbf{M}^{spe} = \begin{pmatrix} p_{1,e}^{tip}(1) & p_e^{TCP}(1) & p_{2,e}^{tip}(1) \\ p_{1,e}^{tip}(2) & p_e^{TCP}(2) & p_{2,e}^{tip}(2) \\ \vdots & \vdots & \vdots \\ p_{1,e}^{tip}(n) & p_e^{TCP}(n) & p_{2,e}^{tip}(n) \end{pmatrix} \tag{2.47}$$

This matrix is used by the grasp pose evaluation.

2.2.5 Grasp Pose Evaluation

The grasp point evaluation is the last task which is necessary to prove if the calculated grasp points are valid. For example, it might be possible that the point is a very good grasping point based on the quality scores, but the antipodal gripper can't grasp the object because it's sparse extension exceeds y_{g_max} (figure 2.1). Or the top of the object as not considered in the surface grasp pose estimation (figure 2.8 (A) green and white dots) might consist of outstanding edges which would cause an collision if the gripper moves from up above to the grasp point. Out of this reasons we need to check if the object is small enough to be grasped and if there are disturbing parts.

To validate the grasping points $\mathbf{M}^{spe}(1,1),...,\mathbf{M}^{spe}(1,n))$ and $\mathbf{M}^{spe}(3,1),...,\mathbf{M}^{spe}(3,n))$ we create a collision area $\square_1^{col}, \square_2^{col}$ around the finger tip grasping points (figure 2.25) with its area corner points $p_{\square 1} = a, p_{\square 2} = b, p_{\square 3} = c, p_{\square 4} = d$ (named a, b, c, d in 2.26). A grasp is not valid, if there is a point $p(x,y) \in pc_1^{cut}$ within the collision area. We use the formulas 2.50, 2.49, 2.51 to calculate the collision area size and if there is any point inside. If there is a point found inside the collision area, a height check in z-direction is done, thus the collision area checking only regards the x,y direction.



(A) (B)

FIGURE 2.25: Created collision area (green dots) around the grasping points $p_{1,e}^{tips}, p_{2,e}^{tips} \in \mathbf{M}^{spe}$ (red dots) and the vector $v_e^{gripper}$ spanned between the grasping points. The points on the left upper corner in A is a close up of one security point from A where the parameters d_1^{col} and d_2^{col} are the half distance of v_1^{col} and v_2^{col}.

The security points are calculate by 2.48, where d_1^{col} and d_2^{col} are parameters to adjust the size of the collision area, with $d_1^{col} = 0.5\|v_1^{col}(x,y)\|$ and $d_2^{col} = 0.5\|v_2^{col}(x,y)\|$. v_1^{col} is parallel to $v_e^{gripper}$ and v_1^{col} perpendicular to it (figure 2.25 A, upper left corner).

$$\begin{aligned} a,b,c,d(p) = p(x,y) &\pm v_e^{gripper}(x,y)\frac{d_1^{col}}{\|v_e^{gripper}(x,y)\|} \\ &\pm v_e^{gripper}(-y,x)\frac{d_2^{col}}{\|v_e^{gripper}(x,y)\|} \end{aligned} \tag{2.48}$$

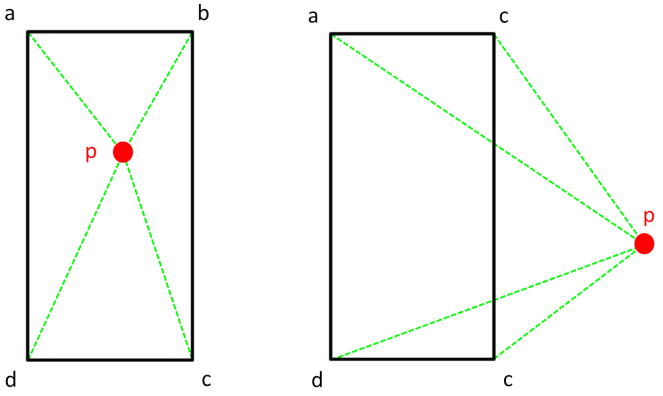
Based on the four points (a,b,c,d) of the rectangle we can prove if a point $p \in pc_1^{cut}$ in x,y direction is outside the rectangle. In this case the sum of the triangle areas (figure 2.26 and equ. 2.49) is greater than the area of the rectangle (equ. 2.50), otherwise it is equal.

$$A_{sum}(a,b,c,d,p) = A_{\Delta}(a,b,p) + A_{\Delta}(a,d,p) + A_{\Delta}(b,c,p) + A_{\Delta}(c,d,p) \tag{2.49}$$

$$A_{rect} = \overline{ab} \cdot \overline{ad} \tag{2.50}$$

$$\begin{aligned} A_{\Delta}(a,b,c) &= \sqrt{s(s-\overline{ab})(s-\overline{ac})(s-\overline{bc})} \\ \text{and } s &= \frac{\overline{ab}+\overline{ac}+\overline{bc}}{2} \end{aligned} \tag{2.51}$$

Algorithm 4 shows how the procedure to find the verdict is executed. Every grasp point in $\mathbf{M}^{spe}$ (expect p_e^{TCP}) is checked if there is any disturbing point from pc_1^{cut} in its collision area, leading to a uncertain grasping position. The z-direction check canvasses if the point which is inside the collision area has a higher z value than the grasp point minus a security threshold Δ_{col} to compensate uncertainties in grasp execution. If the value is lower the gripper would not reach the point in pc_1^{cut} and the grasp is valid despite its localization in the collision area in x/y dimension. The heigh of the object which would allow a collision with the p_e^{TCP} was prevented by filtering pc_{init} to pc_1^{cut}.



(A) If the point p is inside the rectangle, the sum of the triangles is the same as the area of the rectangle.

(B) Otherwise the sum of the areas is greater than the rectangles area.

FIGURE 2.26: Based on 2.51 we can calculate and check if a point is within the collision area.

1 **validate grasping points** $(pc_1^{cut}, \mathbf{M}^{spe})$
2 **forall** *rows i in* $\boldsymbol{M}^{spe}$
3 **do**
4 | **forall** *points j in* pc_1^{cut}
5 | **do**
6 | | a, b, c, d =2.48$(\mathbf{M}^{spe}(i, 1))$
7 | | A_{rect} = 2.50(a, b, c, d)
8 | | $A_{\Delta 1}$ = 2.51$(pc_1^{cut}(j), a, b)$
9 | | $A_{\Delta 2}$ = 2.51$(pc_1^{cut}(j)), b, c)$
10 | | $A_{\Delta 3}$ = 2.51$(pc_1^{cut}(j)), c, d)$
11 | | $A_{\Delta 4}$ = 2.51$(pc_1^{cut}(j)), d, a)$
12 | | $A_{sum} = A_{\Delta 1} + A_{\Delta 2} + A_{\Delta 3} + A_{\Delta 4}$
13 | | **if** $A_{sum} = A_{rect}$ *and* $\boldsymbol{M}^{spe}(i, 1)(z) < pc_1^{cut}(j)(z) - \Delta_{col}$
14 | | **then**
15 | | | go to line 2
16 | | **end**
17 | | a, b, c, d =2.48$(\mathbf{M}^{spe}(i, 3))$
18 | | A_{rect} = 2.50(a, b, c, d)
19 | | $A_{\Delta 1}$ = 2.51$(pc_1^{cut}(j), a, b)$
20 | | $A_{\Delta 2}$ = 2.51$(pc_1^{cut}(j), b, c)$
21 | | $A_{\Delta 3}$ = 2.51$(pc_1^{cut}(j), c, d)$
22 | | $A_{\Delta 4}$ = 2.51$(pc_1^{cut}(j), d, a)$
23 | | $A_{sum} = A_{\Delta 1} + A_{\Delta 2} + A_{\Delta 3} + A_{\Delta 4}$
24 | | **if** $A_{sum} = A_{rect}$ *and* $\boldsymbol{M}^{spe}(i, 3)(z) < pc_1^{cut}(j)(z) - \Delta_{col}$
25 | | **then**
26 | | | go to line 2
27 | | **end**
28 | **end**
29 | return $\mathbf{M}^{spe}(i, 1)$, $\mathbf{M}^{spe}(i, 2)$ and $\mathbf{M}^{spe}(i, 3)$
30 **end**
31 return *objectnotgraspable*

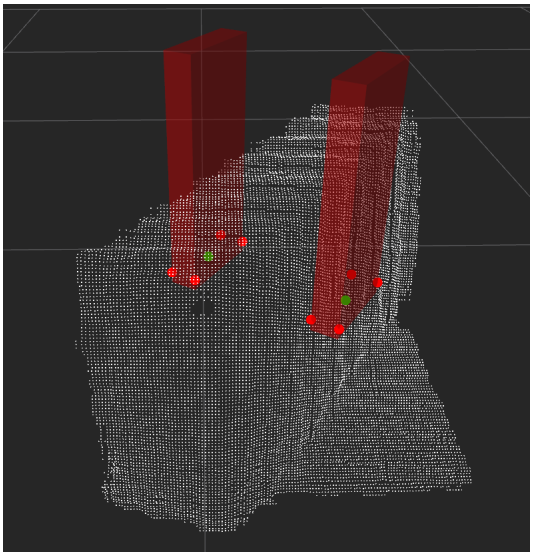
Algorithm 4: The algorithm to validate the grasping points. If one point from pc_1^{cut} is inside the collision area described by a, b, c, d of both grasping points in $\mathbf{M}^{spe}(i, 1)$ as well as $\mathbf{M}^{spe}(i, 3)$ and is within a certain depth border plus a collision threshold Δ_{col}, the grasping point receives a false verdict, this grasp would not b e save to grasp with the gripper. In this case the next row of $\mathbf{M}^{spe}$ would be checked. If all rows yields a false verdict, no grasp is possible and the final verdict is object not graspable. i, j are the iteration indexes.

If the verdict is true, the algorithm stops and the grasping points are used for gripping and a grasp execution starts. Figure 2.27 show a the collision areas with

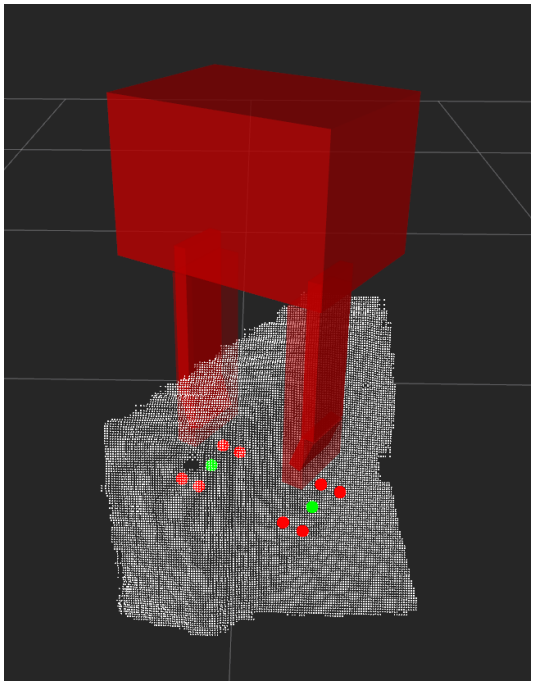
its z-extension in which no points should be (A), a grasp with verdict false (B) and a verdict true, where no disturbing points were found (C). If there is no valid grasp point found in the whole matrix $\mathbf{M}^{spe}$, no grasp execution will be launched.

2.3 Conclusion

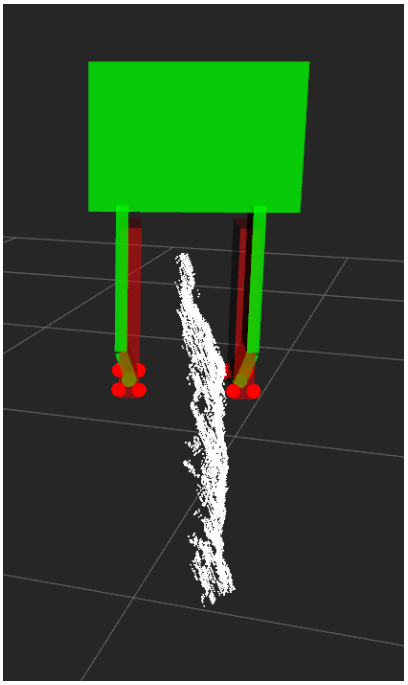
This chapter described the steps done for a grasp pose estimation and evaluation. Starting with the geometrical grasp pose estimation from the incoming point cloud to find the centre of gravity of the object and a first grasp estimation, it distinguish between general and special viewpoint alignments of the object. If supporting points were found, the geometrical grasp point estimation was launched, using curvature, obliqueness and the distance to the centre of gravity as filter parameters. Finally the grasp points were checked for collisions cases. If the verdict was true, a grasping execution will be launched.



(A) The collision area with expansion in z-direction (red blocks over the collision area points).



(B) A false verdict leads to an invalid grasp position (displayed by a red coloured gripper model). The grasping points are discard.



(C) In this case there is no collision, the verdict is true and a grasp is possible (green coloured gripper model).

FIGURE 2.27: This figures shows the spatial extension of the collision zone (A) and two different verdicts of the grasp pose evaluation (B,C).

3 Scheduler and Grasp Execution

Previously the pipeline was only an input driven passive algorithm with a straight calculation order and no possibility to interact with the environment. Extending the pipeline with the possibility to interact with objects the approach develops to an active system, leading to new challenges. One challenge is to find a usable schedule in order to prevent components inferences and provides a smooth work flow.

3.1 Scheduler

Figure 3.3 show a complete time line of the scheduler. First of all the object extraction and the grasp pose detection are running while the manipulation note waits for a steering signal. If an object is tracked by the object extraction and a valid grasp pose was detected, steering signals are sent to the manipulation node (the yellow block in the send line. The color of the block describes the target task to which the signal is sent) and both tasks change the status to idle, waiting for new signals. The data of the steering signals contain the gripper target pose from grasp pose detection and only a execution permission from object extraction.

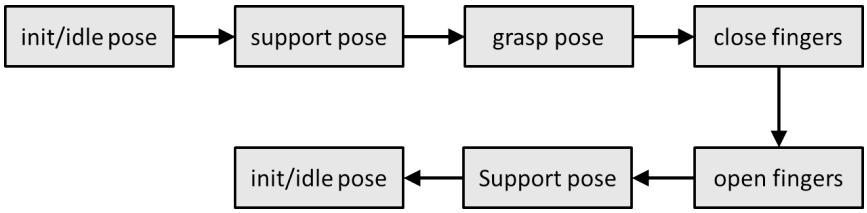


FIGURE 3.1: The first manipulation order is only to grasp the object to establish stable grasps in further manipulation executions and is similar to figure 3.2, only the rotating part (E to G) is missing.

The first manipulation is necessary as the first grasp might change the position of an object. As an example, the cup is grasped with one finger tip on the surface, the other is inside the cup (figure 4.25). If the gripper are closed symmetrically, the cup is shifted by the finger tip on the surface until the finger tips are closed. The resulting new position is a stable grasp pose and will not be shifted inadvertent as the object position is adjusted to the grasp pose. After the execution the node sends a signal to the object extraction in order to check if the object adjustment might lead to an erroneous state (upset the object or shift it from the edge of a table, for example). The execution queue is shown in figure 3.1. If the object is still tracked a steering signal reaches the 3D-mesh-assembly with the tracked point cloud.

The 3D-mesh-assembly stores the point cloud. While the manipulation task launches it's execution, the 3D-mesh-assembly node changes to idle mode. If the manipulation task finished successfully, a signal is sent to object extraction enables gaining

a new point cloud, while the manipulation waits for further orders. The object extractions sends the point cloud to 3D-mesh-assembly and is idle again. As the PC is received and stored, the manipulation task is woken up again by the 3D-mesh-assembly and launches a new manipulation procedure. This is repeated until the desired point clouds are stored which are necessary to generate the 3D-mesh (n times). In this case, the 3D-mesh-assembly sends steering signals to object extraction and grasp pose detection to start a new time line and goes idle itself. The data flow is shown in 3.4.

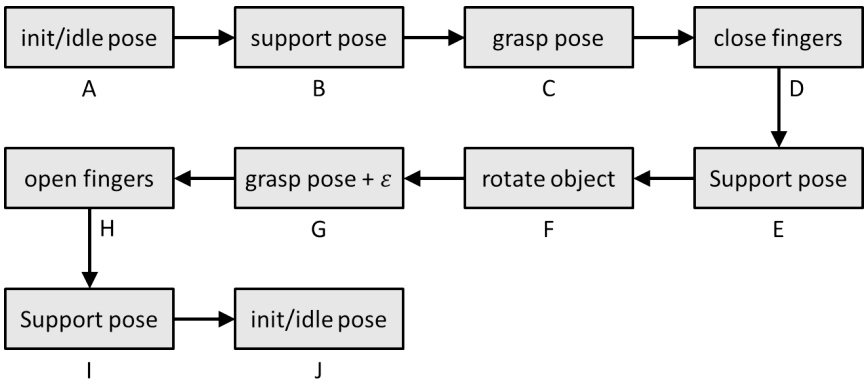


FIGURE 3.2: The different poses for one grasping sequence, beginning with the idle pose. The letters A to J are similar to 3.7 (C and D are combined in 3.7 to D).

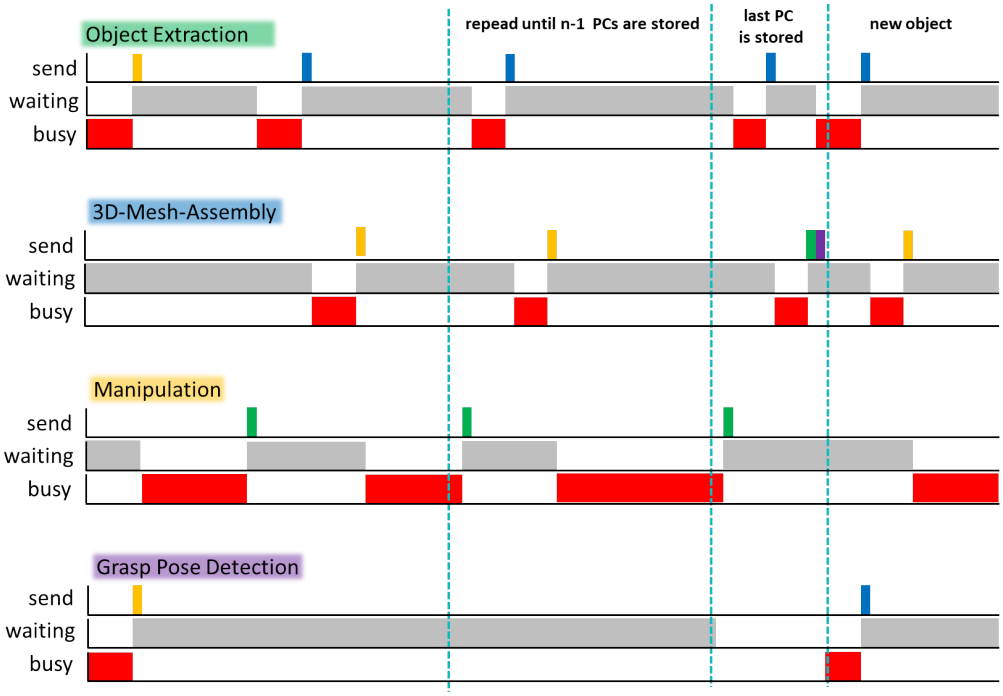


FIGURE 3.3: The scheduler for the project. Due to its larger explanation, the description is shown in the text

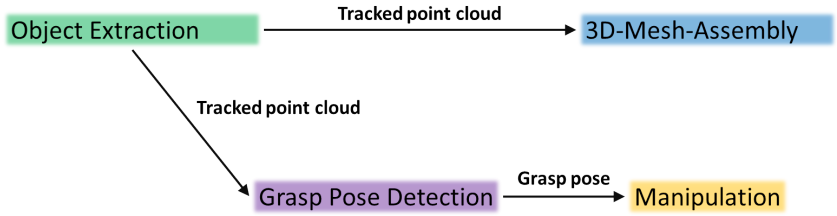
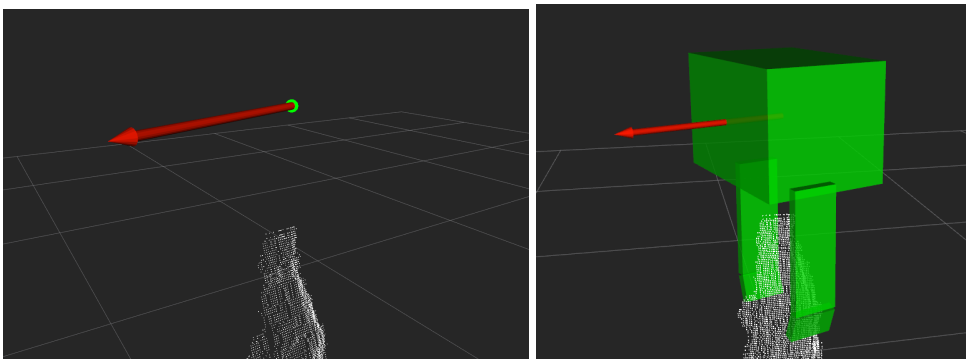


FIGURE 3.4: Data flow of the scheduler. Object tracking sends point clouds to 3D-mesh-assembly and grasp pose detection. 3D-mesh-assembly stores the point cloud if the steering signal reaches the node. Grasp pose detection calculates the grasp pose for the gripper.

3.2 Grasp Execution

Receiving the signal which contains the TCP position in in x, y, z as well as the gripper rotationin quaternion (x, y, z, w) launches the grasp execution (3.5). The whole grasp execution is divided in subsequences as seen with a real robot in figure 3.7 and as a schematic in 3.2. It starts with the idle or initial frame, the robot arm is stretched out. As the PC is caught, the 3D-mesh-assembly signals its state switching to manipulation, the gripper moves to the previously validated position plus a threshold, Δz_{sup}^{TCP}, to define a support point p_{sup}^{TCP} (3.1) It allows reaching the final grasp position only by moving in z direction.

$$p_{sup}^{TCP} = \begin{pmatrix} x = p^{TCP}(x) \\ y = p^{TCP}(y) \\ z = p^{TCP}(z) + \Delta z_{sup}^{TCP} \end{pmatrix} \tag{3.1}$$



(A) point and orientation of the gripper (B) orientation and the gripper as a scheme (green geometry)

FIGURE 3.5: A validated grasp position (green dot) and its alignment (red arrow), describing the rotation in quaternions.

If p_{sup}^{TCP} is reached, the gripper moves along the z-axis with its destination p^{TCP} and the finger tips are closed in order to grasp the object when p^{TCP} is reached. The grasped object is lifted and enables the rotation procedure afterwards. The object is rotated around the z-axis for $\alpha = \frac{2\pi}{n}$ where n are the desired different poses and therefore the stored snapshots of point clouds. After rotating around α the object is moved back to the tabletop (p^{TCP} plus a small z offset to prevent squeezing the

object when reaching the tabletop $p^{TCP}(z) + \epsilon$) and the gripper releases the object. As an interim step the arm moves back to p^{TCP}_{sup} and finally to the initial pose in order to give a free sight for the camera, prevent an occlusion of the object by the robot arm (3.6).

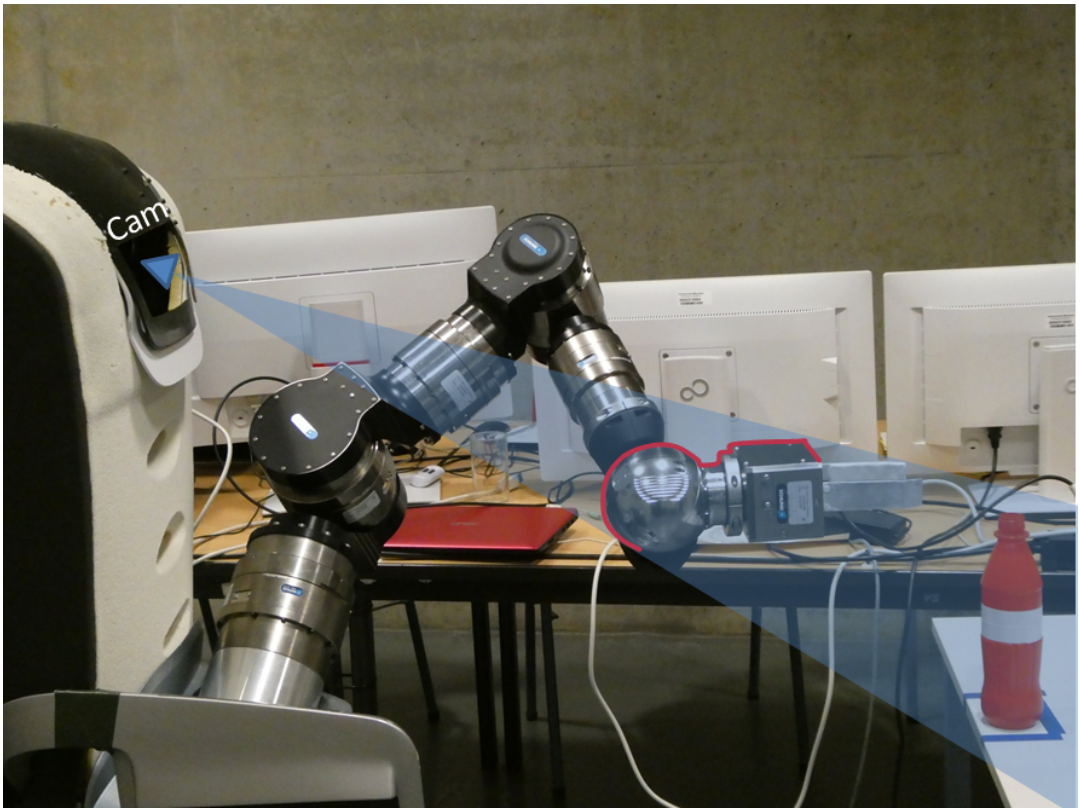
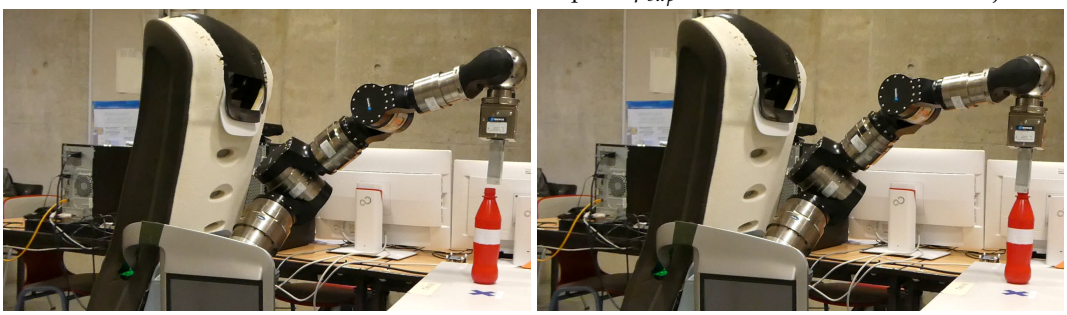


FIGURE 3.6: It is not guaranteed that the pose of the robot arm in order to grasp the object does not cover the vision of the camera. To ensure a free vision to take a new PC snapshot of the rotated object the robot arm is moved to the initial state (3.7 J).

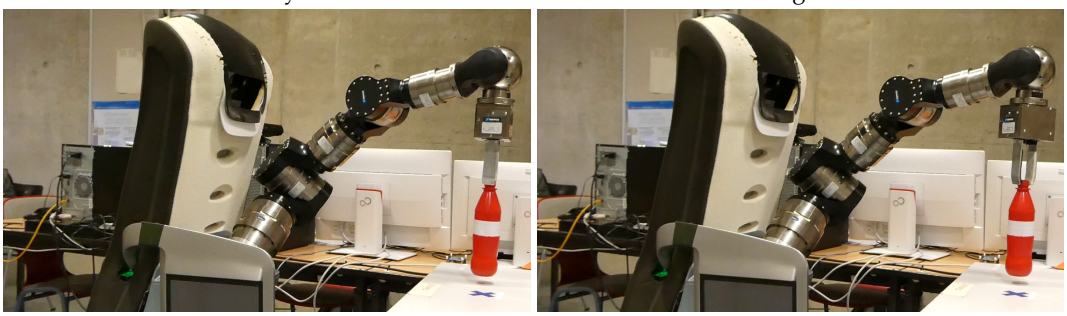
(A) idle pose

(B) moving to grasp pose + z threshold = support point p_{sup}^{TCP} in order to move over the object



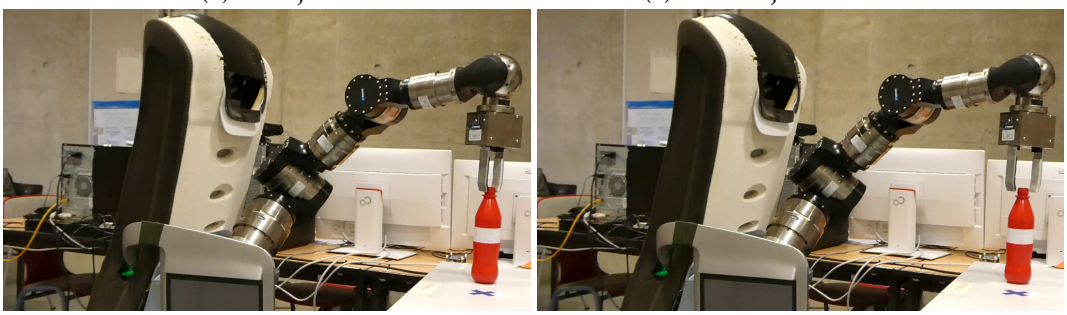
(C) manual adaptation due to calibration uncertainty

(D) moving gripper to grasp position and close fingers



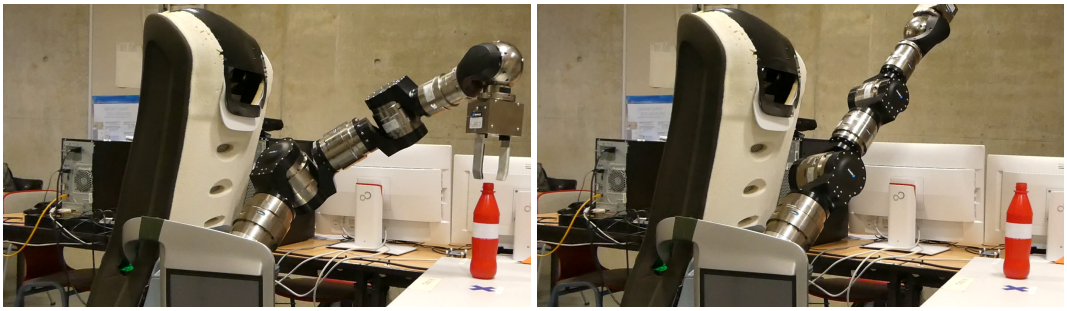
(E) lift object

(F) rotate object around z-axis



(G) lower object

(H) open gripper finger



(I) move back to p_{sup}^{TCP}

(J) go back to idle pose in order to allow PC snapshots

FIGURE 3.7: The first grasping sequence which is used for object rotation. The subsequent grasping sequences does not contain step C. It is only needed to compensate calibration uncertainty.

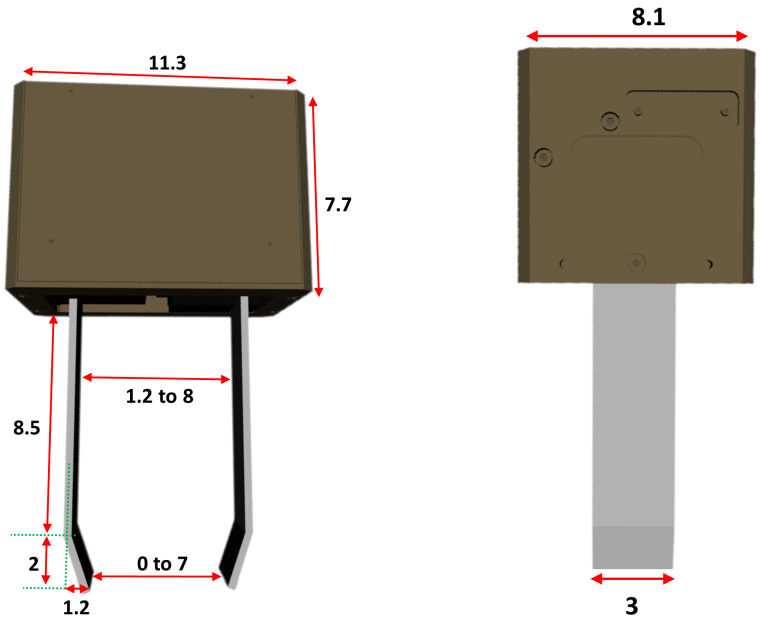
4 Results - Grasp Pose Estimation

In this section we will demonstrate the results of the approach. The system specification on which the computations are running are a Intel Core i7-4790 CPU with 3.60 GHz base clocking, 16 GB of RAM with Ubuntu 16.04 as the operating system. The Care-o-Bot version 3.9, developed by the Fraunhofer Institute for Manufacturing Engineering and Automation, is used as information gathering and manipulation execution system equipped with a vision sensor and a robot arm with gripper. A X-Box 360 Kinect is used as the vision sensor for creating the point clouds mounted on the top of the Care-O-Bot. The robot arm itself is a schunk system using several universal servos with a schunk PG+70 2-finger-parallel gripper on the TCP (dimensions of the gripper are shown in figure 4.1). We use the following items for tests cases.

- Bottle
- Box
- Can
- X-Box controller
- Cup
- Paper cup
- Stanford bunny

Some object has several viewpoints so a total of 12 different configurations (objects and viewpoints) are tested and analysed. We set the parameters as followed and they stay the same for every configuration:

- Grasp Width $= 7$ cm
- $\Delta_{cut} = 1.5$ cm
- $\Phi_{sup} = 0.75$
- $\Phi_{nn} = 0.7 = 70\%$
- $\Phi_{curv} = 0.8 = 80\%$
- $\Phi_{ac} = 0.15 = 15\%$
- $n_{max}^{grasps} = 10$
- $d_1^{col} = 0.25$ cm
- $d_2^{col} = 1.5$ cm



(A) Dimensions of the gripper, front view (B) Dimensions of the gripper, side view

FIGURE 4.1: The dimensions of the gripper, in centimetre (cm). The values for 0 to 7 and 1.2 to 8 are the given ranges the gripper can be closed or opened for grasping.

The reference algorithm is the HAF Grasp Pose Detection [19] as it has the idea of a grasping vector in common with our idea. The area which is scanned and used for grasp pose estimation and the centre of this area are two parameters we can set in HAF. For the center coordinates we use the point p^{TCP} of Geometrical Grasp Pose Estimation (figure 2.7) and the used area dimensions are $x = 18$ cm and $y = 30$ cm.

By virtue of inaccuracy between the input sensor (X-Box Kinect) and the manipulation system (the robot arm and the gripper) we are not able to test the full approach as described in this thesis as one complete pipeline without external intervention. If the gripper reaches the estimated grasping points as calculated, the reached position in reality differs in a range about 6 cm to the position where the system means to be based on the intrinsic measured values by given sensors (4.2). Therefore an engagement is necessary to adjust the grippers position in order to execute a real grasp. This means our section only shows the results of the grasp pose estimation. The manipulation execution itself is proved by figure 3.7 which is working if the grasp point was corrected.

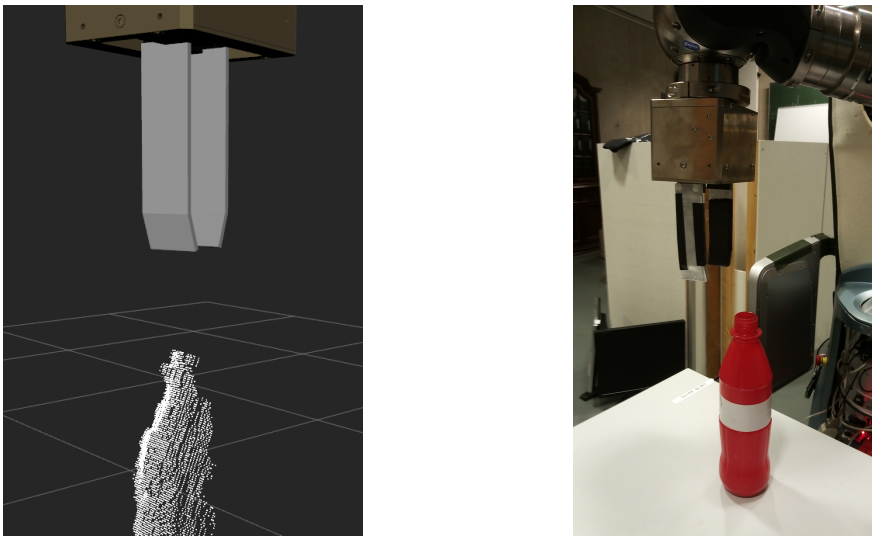
We measured the execution time of the manipulation based on 3.2 (All sequence steps from (A) to (J)). The total time for seven test runs was 758.46 seconds with an average of 108.35 seconds for every execution. The time depends strongly on the calculated trajectory the robot arm uses for the movement from start to the goal position. Test Nr. 7 needs 122.37 seconds due to a non optimal trajectory calculation (4.1).

In addition a full video of the execution is in the added storage drive. In this case the gripper grasp width (between 0 to 7 cm) was measured before as the gripper has no tactile sensors which is a second constraint to test the grasp in real as the grasp

Measurement Nr	Time in seconds		
1	106.87	calculation time (sec)	
2	101.48	0.645643	
3	106.22	0.114081	
4	106.76	7.63854	
5	105.61	5.45948	
6	109.15	0.317514	
7	122.37	2.69163	
Total	758.46	Average	108.35

TABLE 4.1: Measured time for one regular manipulation execution.

width must be known a priori. The mesh assembly was proven in [1].



(A) The estimated gripper position. (B) The real position of the gripper.

FIGURE 4.2: The estimated gripper position based on the intrinsic information (A) and the real gripper position (B). In this case the x/y distance is 6 cm. The gripper position is p_{sup}^{TCP}, thus the estimated and real gripper position is located above the object.

The objects configurations are separated into different heading, the content framework stays the same. The first figure shows the information of the vision sensor as the robot sees the current environment (A) and the resulting point cloud the grasp pose estimation receives (B). The second figure gives an overview about the calculated curvatures according to the grasping points p_1^{tip} (A) and p_2^{tip} (B) and the quality scores for the points from pc_1^{curv} (C) and pc_2^{curv} (D). One exception is the heading 4.0.2 Box - front view where the special view point alignment comes into account leading to non existing support points and therefore no surrounding points for curvature or quality score evaluation. The last figure shows the calculated grasping points, from a default view (A) and top view (B). Green points in the figure are the points for the finger tips calculated by our approach, the red points by the HAF grasp pose estimation. The green or red sheer geometries are a visual model of the gripper (TCP body + finger tips).

Tables 4.2 and 4.3 show a summary of the test results. The calculation time is measured between the incoming of a point cloud pc^{init} and the grasp point output as the verdict ($p_{1,e}^{tip}$, p_e^{TCP}, $p_{2,e}^{tip}$). It has shown that our approach always needs a lower calculation time than the HAF Grasping Detection.

Own approach				
object	view	calculation time (sec)	quality score	verdict
Bottle		0.645643	0.0142225	0
Box	front	0.114081	*1	+
	side	7.63854	0.0338669	0 +
	default	5.45948	0.0625476	+
Can		0.31751	0.0133974	+
X-Box Controller	default	2.69163	0.128934	0
	back	2.63445	0.144856	0
Cup		2.14741	0.0318229	0 +
Paper Cup		1.16634	0.138421	+
Stanford Bunny	front	1.52609	0.0225819	0 +
	back	2.39068	*2	-
	side	3.27662	*2	-

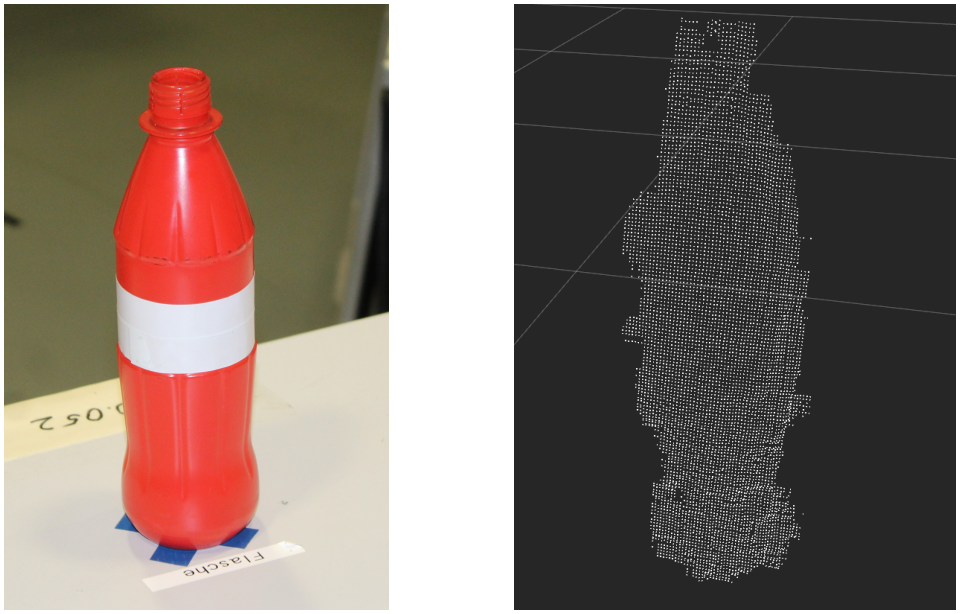
TABLE 4.2: Results of our approach. The listed objects with the viewpoint as seen in the figures in the corresponding headings. The calculation time is the time needed to calculate the final grasp points, starting from the incoming of the point cloud. The quality score counts for valid grasp points. Verdicts represents a visual subjective examination and are segmented into + (high chance of successful grasping), 0 + (possible, but might fail), 0 (high chance of failure) and - (not graspable due to collision or wrong estimation). *1: no quality score due to special viewpoint alignment. *2: no quality score due to no available grasps (all possible grasps leads to collisions).

HAF			
object	view	calculation time in sec	verdict
Bottle		3.989938	-
Box	front	5.624886	0
	side	7.919145	0
	default	7.761026	0
Can		4.083223	0 +
X-Box-Controller	default	6.703462	-
	back	6.418317	-
Cup		5.588885	-
Paper Cup		4.937268	-
Stanford Bunny	front	6.658354	-
	back	5.372491	-
	side	6.684438	-

TABLE 4.3: Results of the HAF-Grasping approach

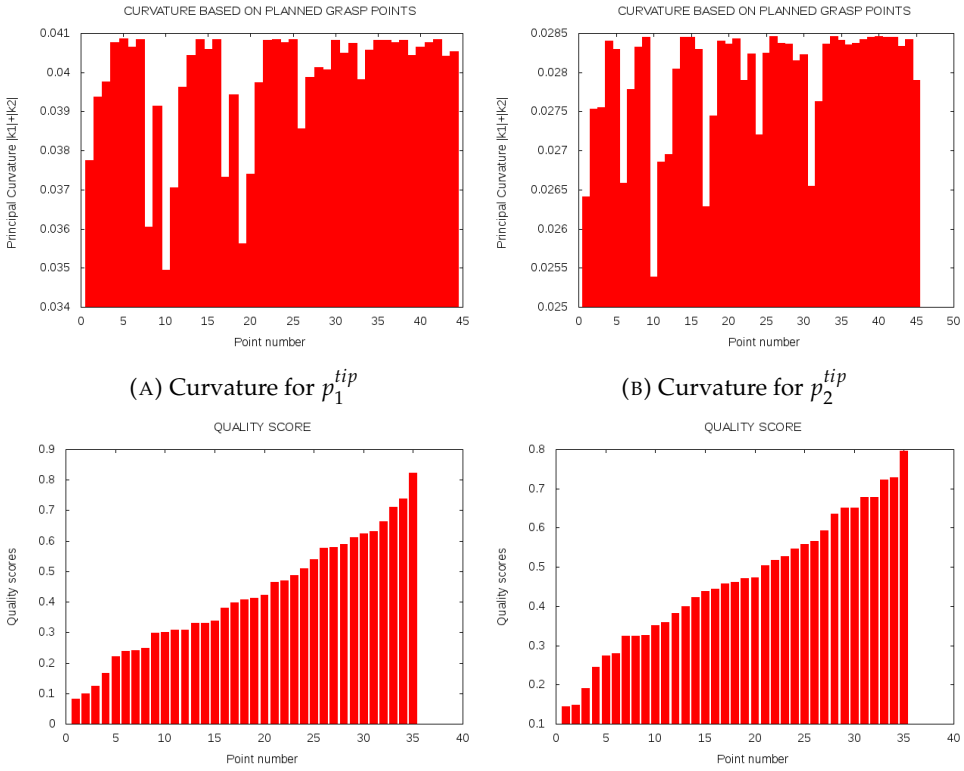
4.0.1 Bottle

With a quality score of 0.0142225 and calculation time of 0.645643 seconds, our algorithm is 517.97 % faster than the HAF computation time of 3.989938 seconds. In addition our grasp position reaches a subjective result of 0. It is graspable as the surface is flat (lies in the area of the white label (4.3 A), but it might fail due to the unknown back geometry of the bottle. The gripper grasping width could be too small for the bottle size. HAF returns a grasp estimation which is not possible as one grasp point is located inside the bottleneck. Figure 4.5 B shows not the top view, it is another default view.



(A) (B)

FIGURE 4.3: Bottle, real model and resulting PC.



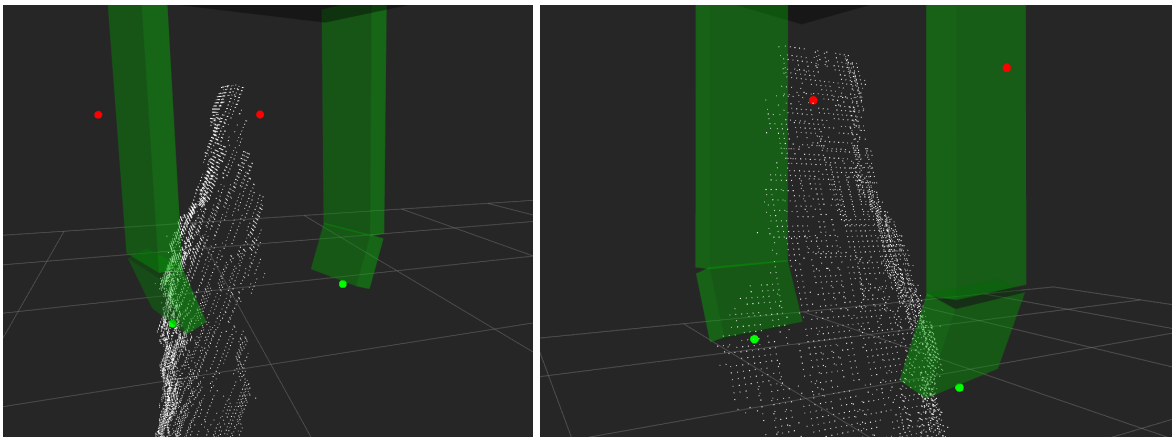
(A) Curvature for p_1^{tip}

(B) Curvature for p_2^{tip}

(C) Quality scores for p_1^{tip}

(D) Quality scores for p_2^{tip}

FIGURE 4.4: Calculated curvatures and quality scores for p_1^{tip} and p_2^{tip} using the bottle PC.



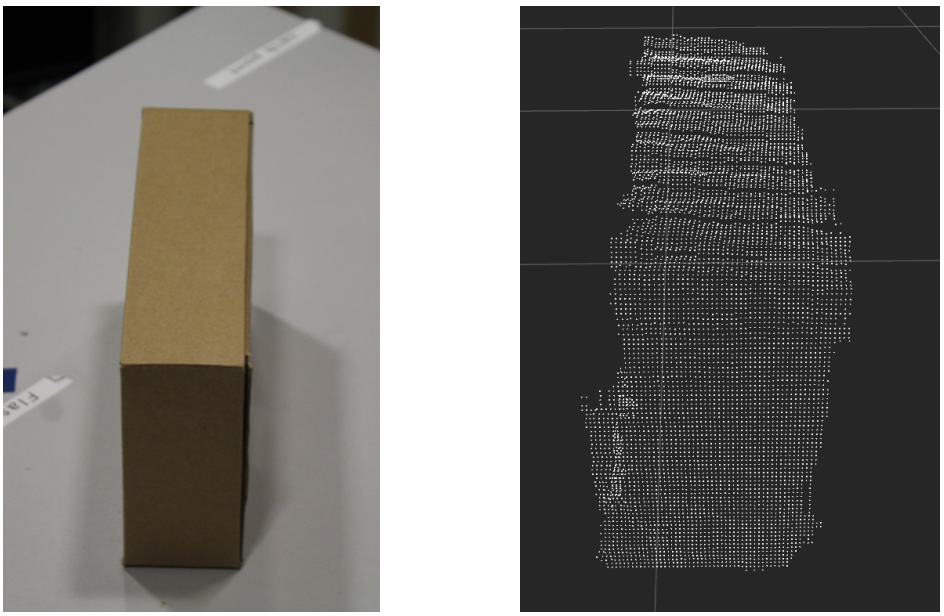
(A)

(B)

FIGURE 4.5: The calculated grasp using the bottle PC.

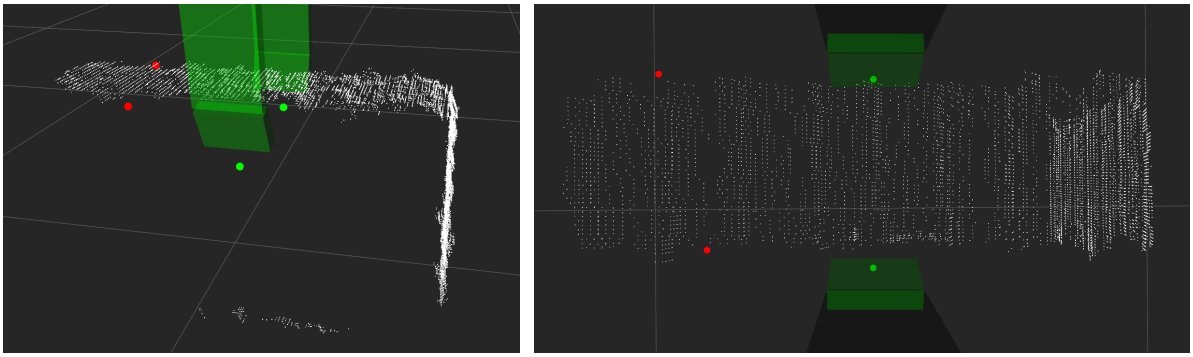
4.0.2 Box - front view

With a calculation time of 0.114081 seconds, our algorithm is 4851.66% faster than the HAF computation time of 5.624886 seconds. In addition our grasp position reaches a subjective verdict of +. This configuration is a special case as there are no support points near the geometrical estimated grasp points. In this case the first grasp estimation is used for grasping. The grasp pose is reachable, the rotation is perpendicular to the surface and very close to the center of weight. The HAF estimation lacks in orientation as it would rotate the object (which is not a severe problem as the first manipulation compensates it). The deferral to the center increases the probability of an erroneous manipulation (figure 2.8).



(A) (B)

FIGURE 4.6: Box, real model and resulting PC, front view.

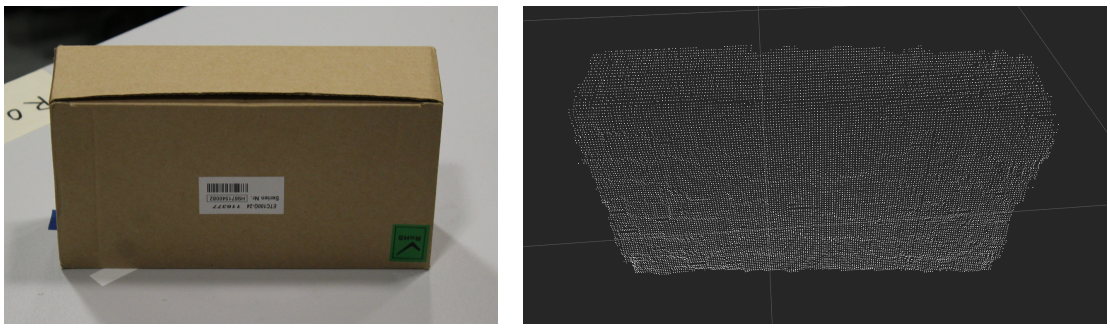


(A) (B)

FIGURE 4.7: The calculated grasp using the box PC, front view.

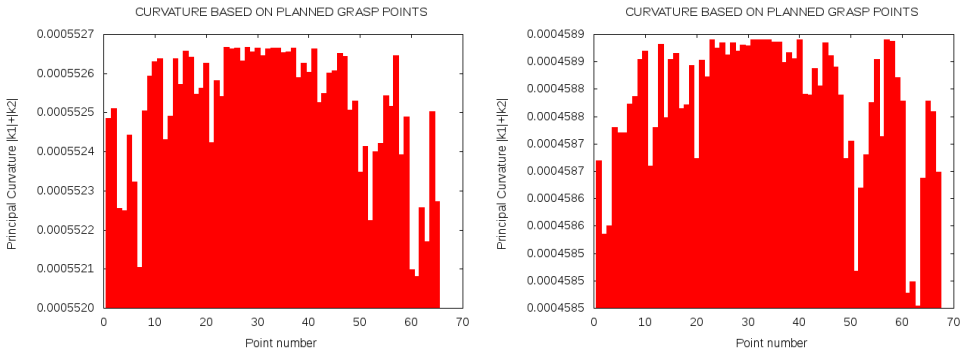
4.0.3 Box - side view

With a quality score of 0.0338669 and calculation time of 7.63854 seconds, our algorithm is 3.67 % faster than the HAF (7.919145 seconds). The verdict for our approach is 0 +, because the back grasp point is very close to the edge (upper red point in 4.10 B). If the point cloud is noisy, a collision might occur, but the probability is low. Like Box - front view, the orientation as well as the distance to the center is in a appropriate scale. The HAF results are similar to Like Box - front view, but with a better rotation estimation. Even more, the grasping is very close to the upper edge. It could be possible that the edge is not graspable if it is a elastic material.

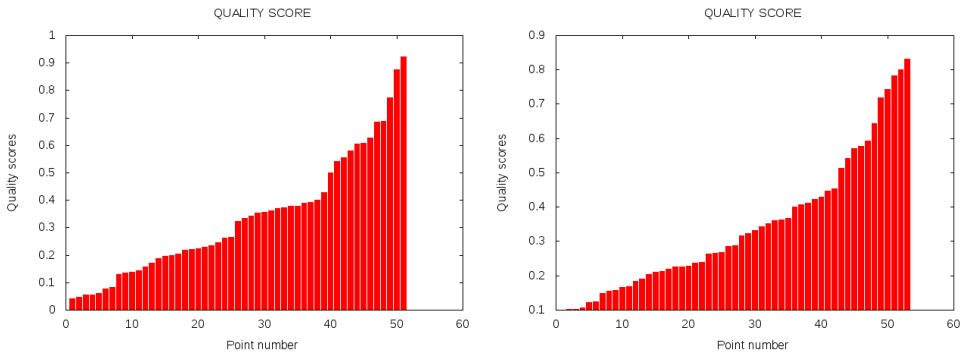


(A) (B)

FIGURE 4.8: Box, real model and resulting PC, side view.

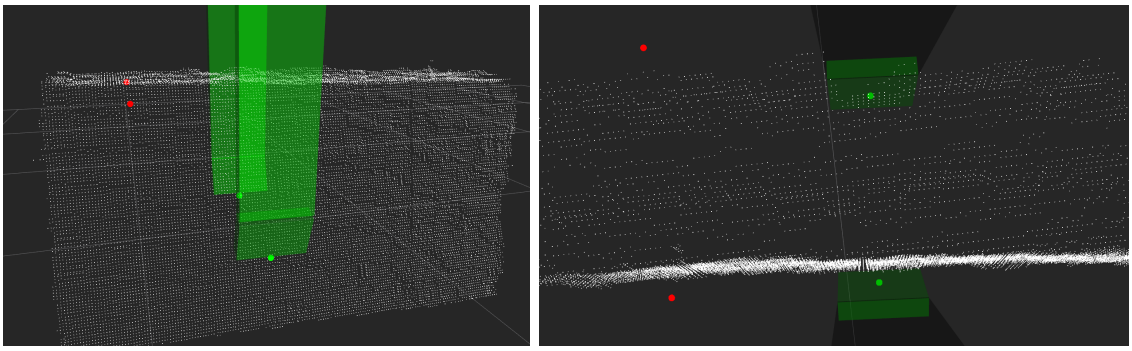


(A) Curvature for p_1^{tip} (B) Curvature for p_2^{tip}



(C) Quality scores for p_1^{tip} (D) Quality scores for p_2^{tip}

FIGURE 4.9: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ for the box, side view.

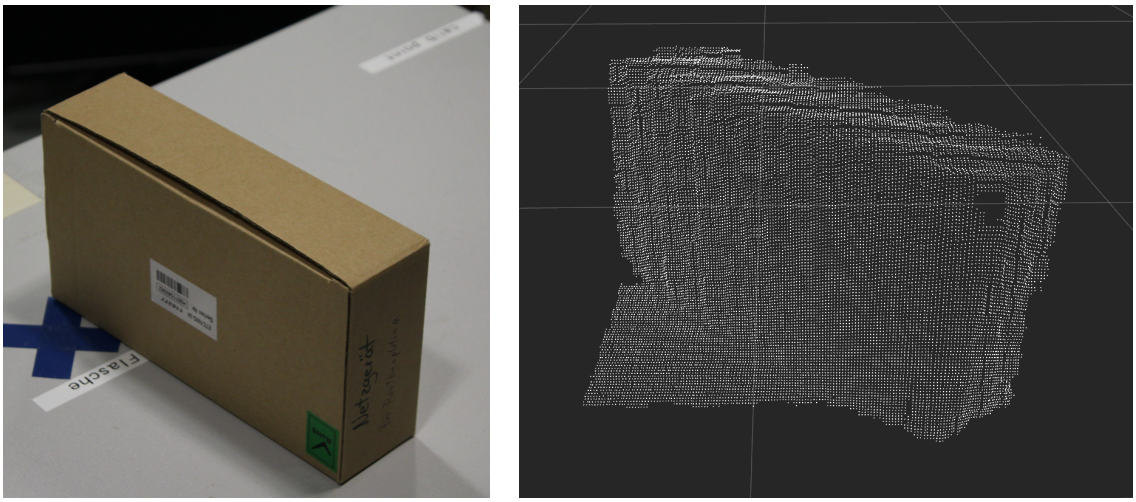


(A) (B)

FIGURE 4.10: The calculated grasp for the box, side view.

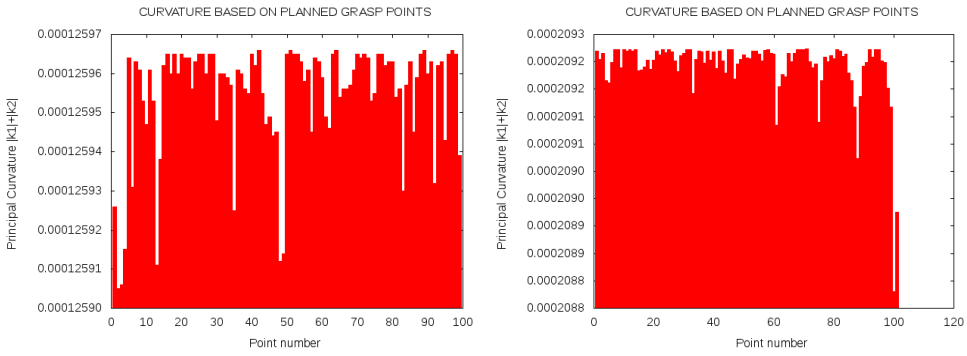
4.0.4 Box - default view

With a quality score of 0.0625476 and calculation time of 5.45948 seconds, our algorithm is 1589.19 % faster than the HAF computation time of 7.761026 seconds. The verdict for our approach is +. The results are similar to Like Box - front view and Box - side view.

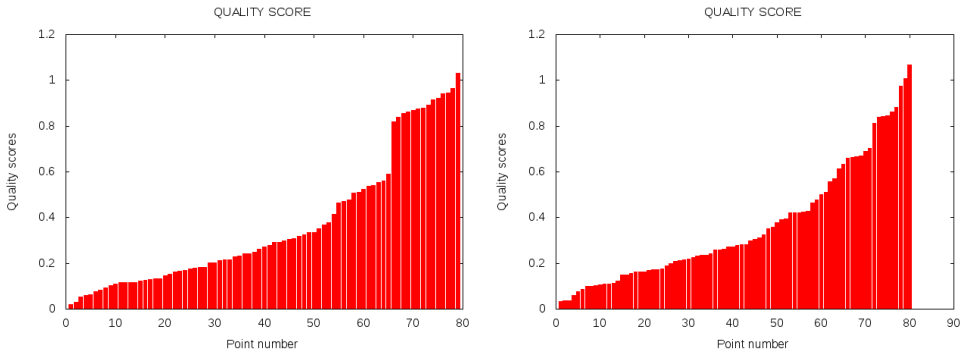


(A) (B)

FIGURE 4.11: Box, real model and resulting PC, default view.

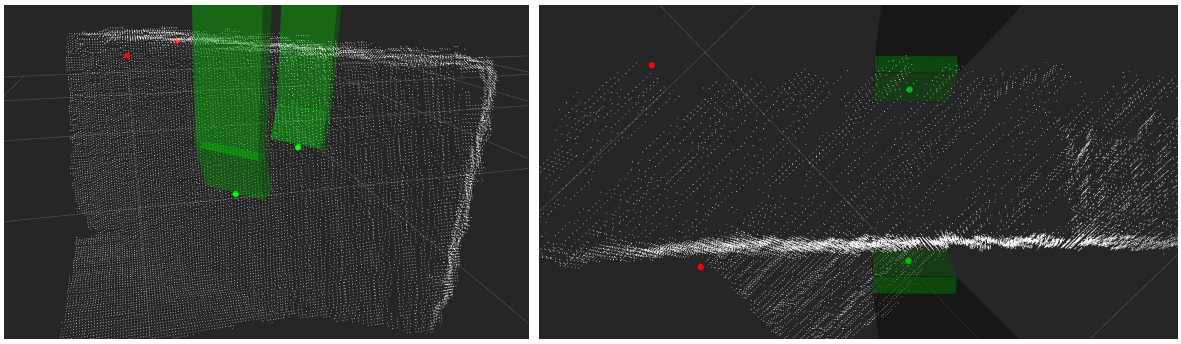


(A) Curvature for p_1^{tip} (B) Curvature for p_2^{tip}



(C) Quality scores for p_1^{tip} (D) Quality scores for p_2^{tip}

FIGURE 4.12: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ for the box, default view.

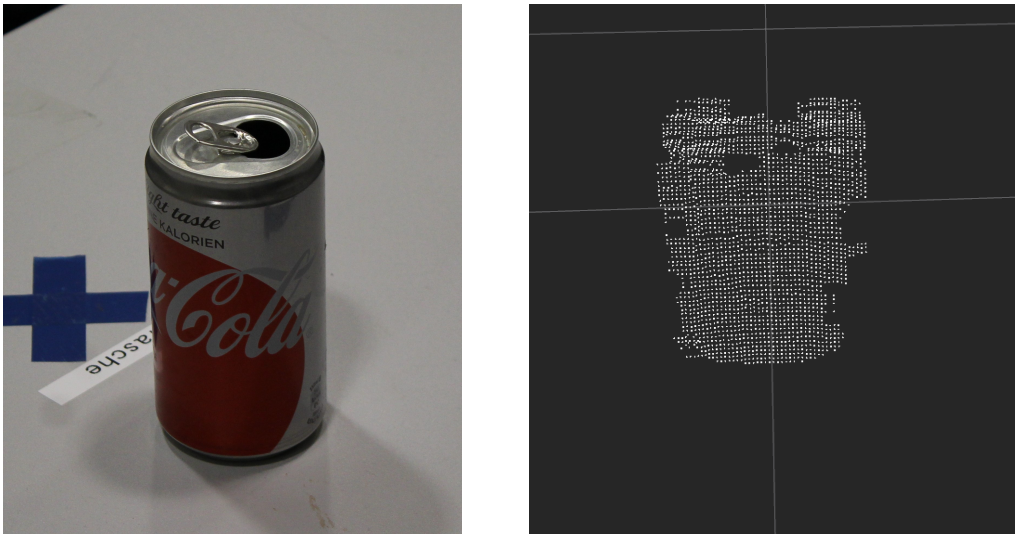


(A) (B)

FIGURE 4.13: The calculated grasp using the box PC, default view.

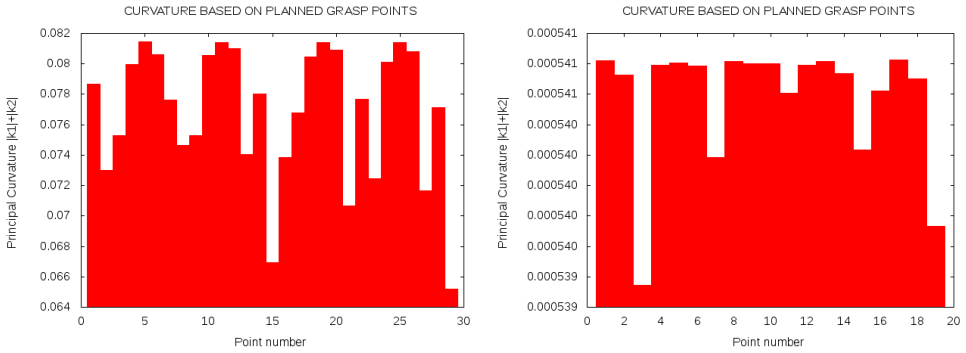
4.0.5 Can

With a quality score of 0.0133974 and calculation time of 0.3175 seconds, our algorithm is 1186.05 % faster than the HAF computation time of 4.083223 seconds. The verdict of our approach is +. Both approaches provide good results, in orientation and distance to the center. One HAF grasping point is very close the point cloud (left point), a possible collision cannot rule out.

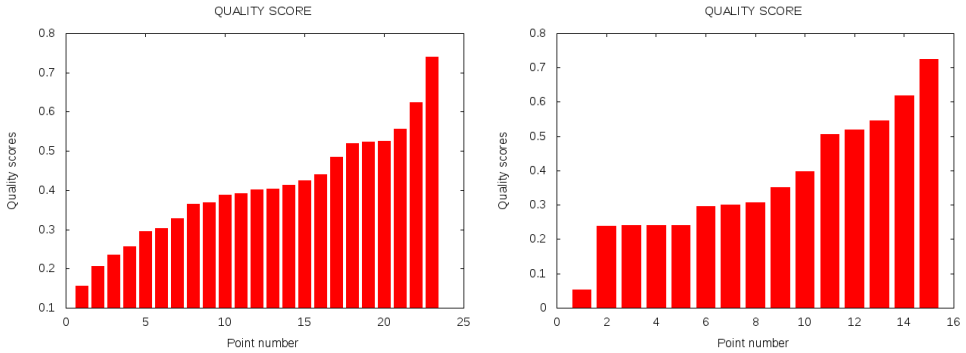


(A) (B)

FIGURE 4.14: Can, real model and resulting PC.

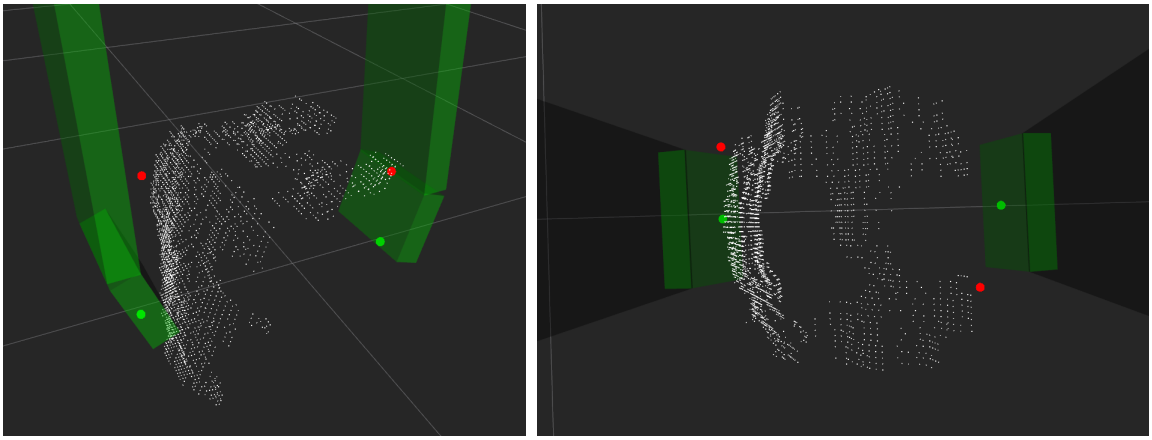


(A) Curvature for p_1^{tip} (B) Curvature for p_2^{tip}



(C) Quality scores for p_1^{tip} (D) Quality scores for p_2^{tip}

FIGURE 4.15: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ using the can PC.



(A) (B)

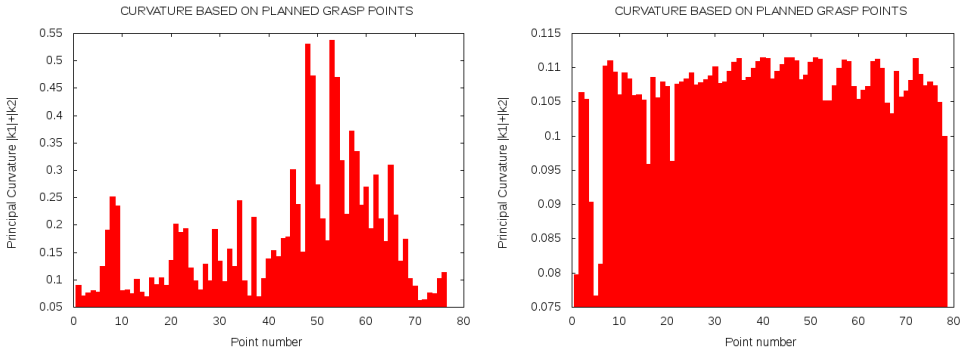
FIGURE 4.16: The calculated grasp using the can PC.

4.0.6 X-Box Controller - default view

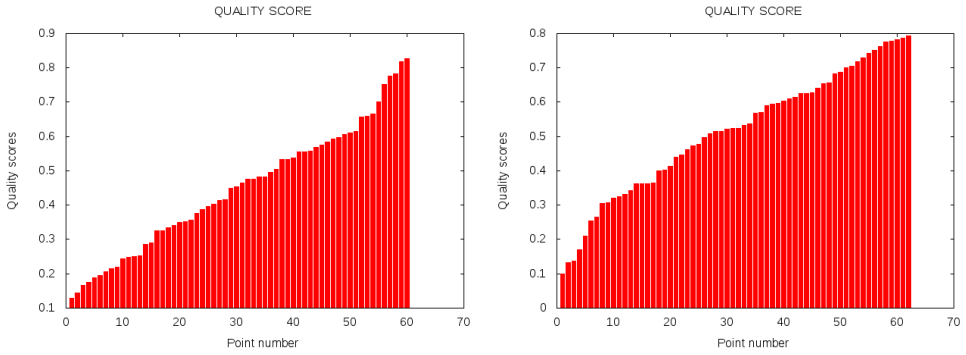
With a quality score of 0.128934 and calculation time of 2.69163 seconds, our algorithm is 149.04 % faster than the HAF computation time of 6.703462 seconds. The verdict of our approach is 0. HAF suggests a grasp over the object itself, so it is not graspable, in addition to a high distance to the center of weight. Our approach calculates a grasp point in a graspable area wit the right orientation, but the height is quite close to the edge, so a grasp might fail.

(A) (B)

FIGURE 4.17: X-Box controller, real model and resulting PC, default view.

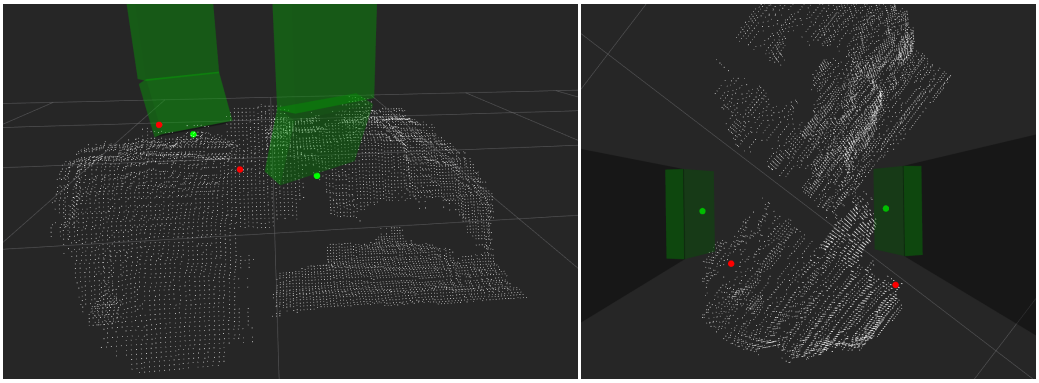


(A) Curvature for p_1^{tip} (B) Curvature for p_2^{tip}



(C) Quality scores for p_1^{tip} (D) Quality scores for p_2^{tip}

FIGURE 4.18: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ using the X-Box controller PC, default view.

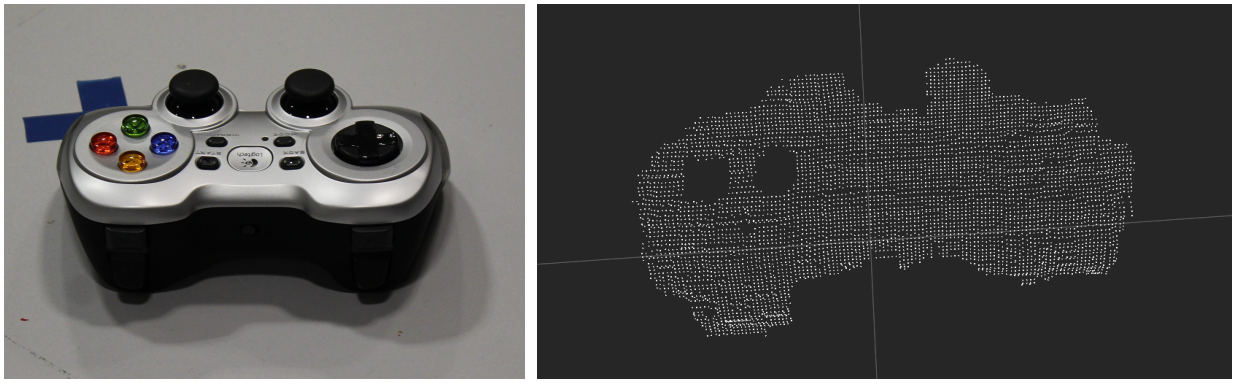


(A) (B)

FIGURE 4.19: The calculated grasp for the X-Box controller PC, default view.

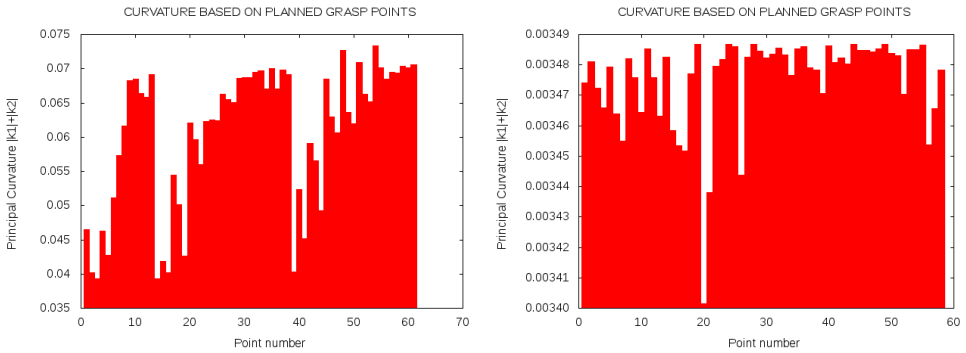
4.0.7 X-Box Controller - back view

With a quality score of 0.144856 and calculation time of 2.63445 seconds, our algorithm is 143.63 % faster than the HAF (6.418317 seconds). The verdict for our approach is 0. HAF provides a grasp over the object, the distance to the center of weight is very high, but the orientation is correct. A central grasp with the right orientation is suggested by our approach, but the grasp is very close to the edge similar to X-Box Controller - default view.

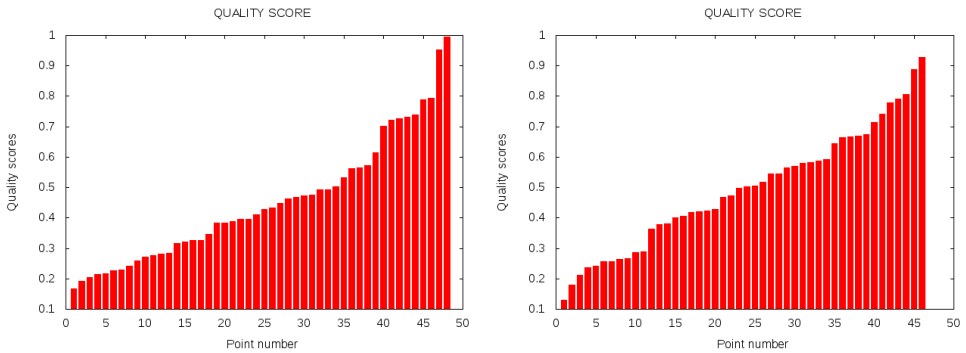


(A) (B)

FIGURE 4.20: X-Box controller, real model and resulting PC, back view.

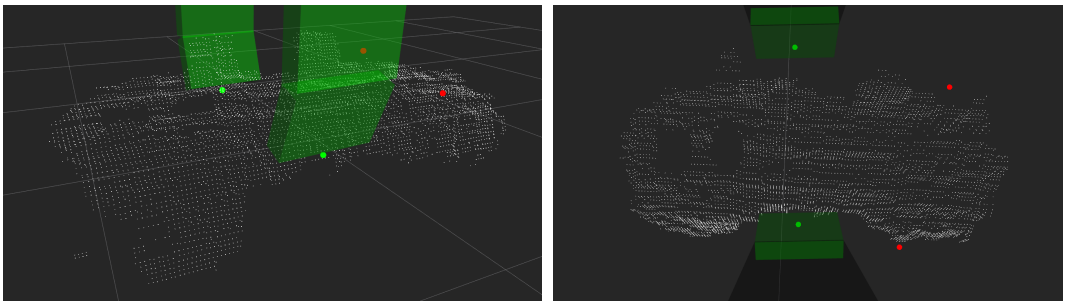


(A) Curvature for p_1^{tip} (B) Curvature for p_2^{tip}



(C) Quality scores for p_1^{tip} (D) Quality scores for p_2^{tip}

FIGURE 4.21: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ using the X-Box controller PC, back view.

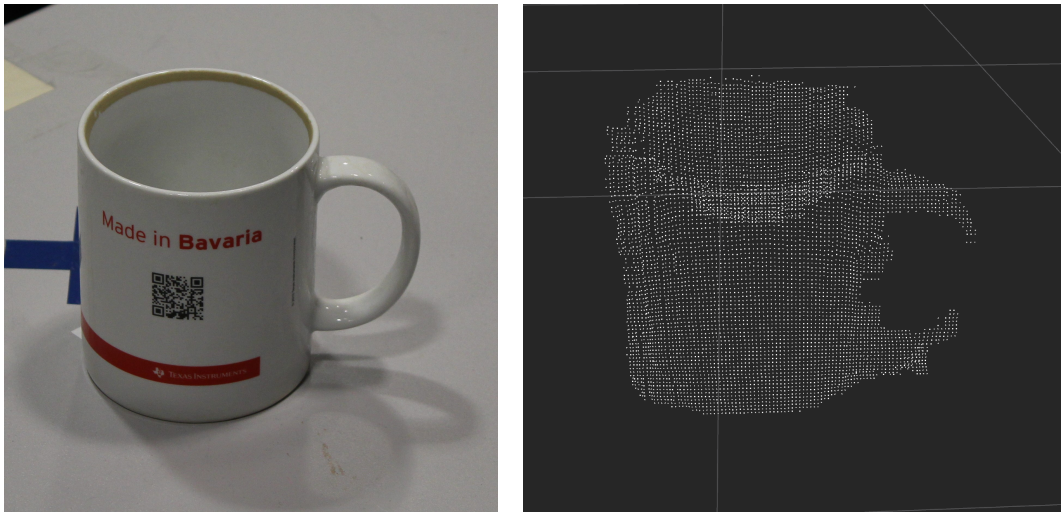


(A) (B)

FIGURE 4.22: The calculated grasp for the X-Box controller, back view.

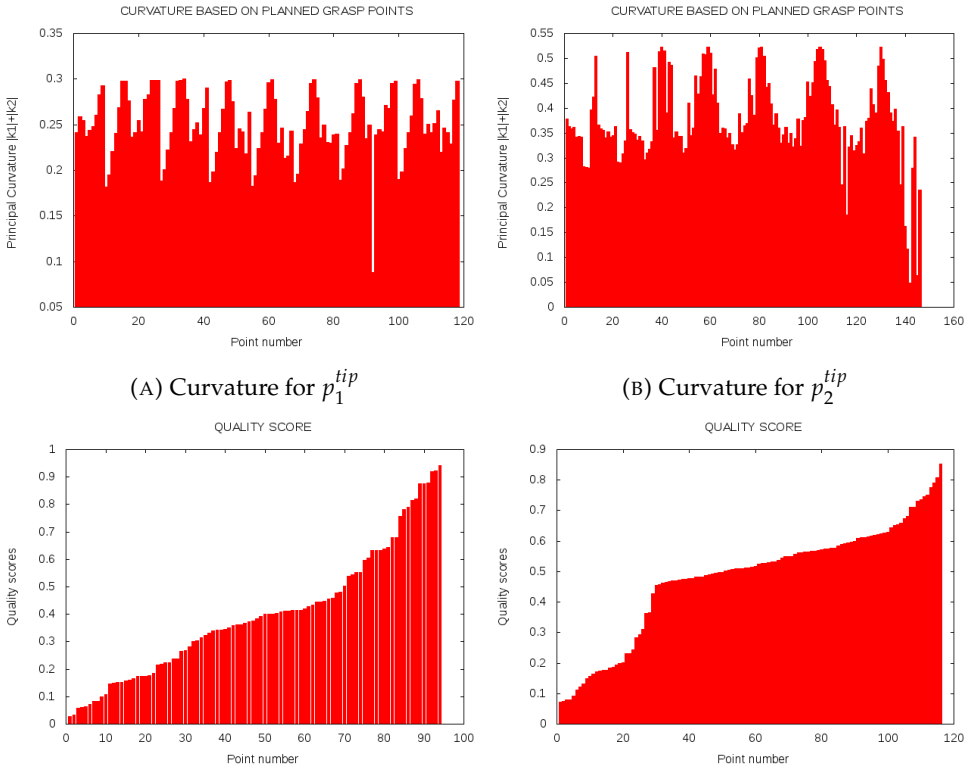
4.0.8 Cup

With a quality score of 0.0318229 and calculation time of 2.14741 seconds, our algorithm is 160.26 % faster than the HAF computation time of 5.58885 seconds. The verdict is 0 +, since the orientation is not ideal and the grasp will shift the cup as mentioned before. On the other hand, the HAF provides a grasp on the handle close to the torso, which has a high failure chance. Adventitious in this case the grasping point are located above the handle, so these points results in an empty grasp.



(A) (B)

FIGURE 4.23: Cup, real model and resulting PC.



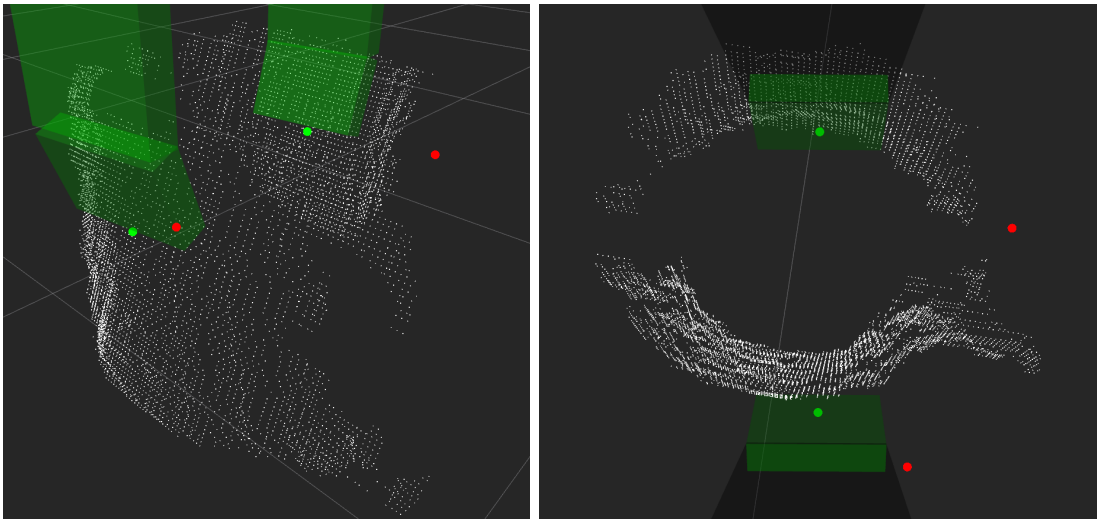
(A) Curvature for p_1^{tip}

(B) Curvature for p_2^{tip}

(C) Quality scores for p_1^{tip}

(D) Quality scores for p_2^{tip}

FIGURE 4.24: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ using the cup PC.



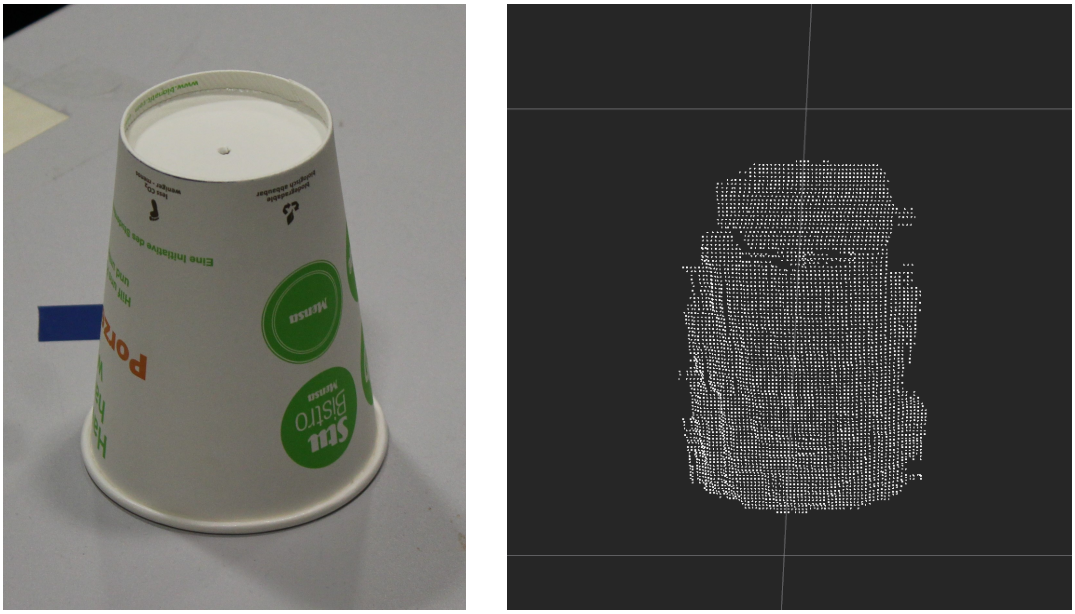
(A)

(B)

FIGURE 4.25: The calculated grasp using the cup PC.

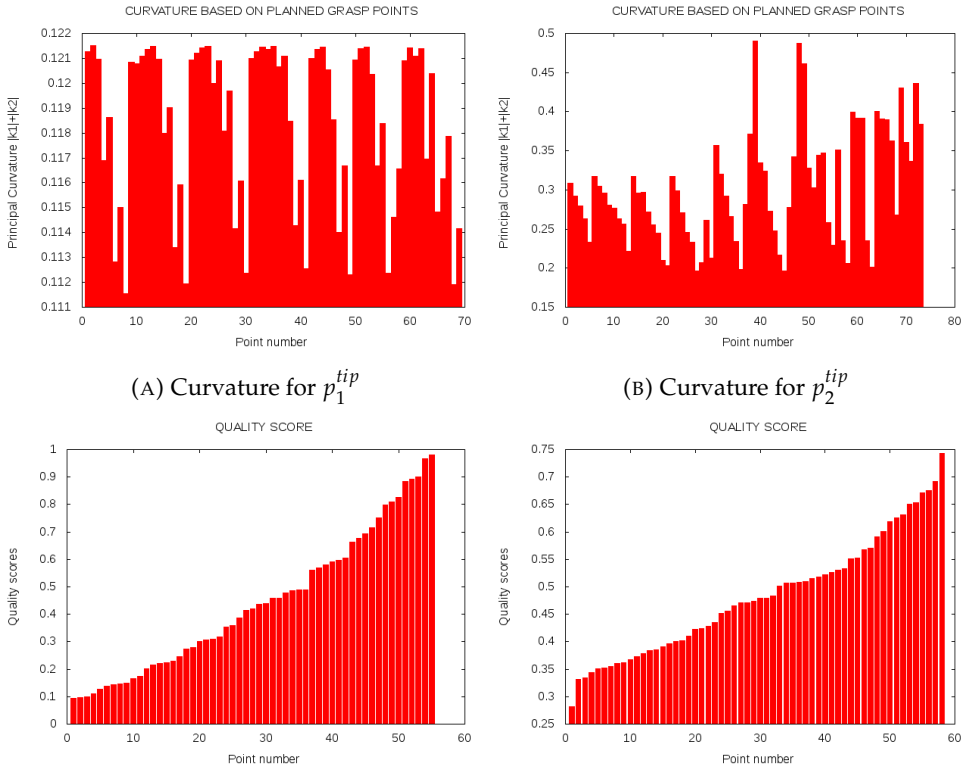
4.0.9 Paper cup

With a quality score of 0.138421 and calculation time of 1.16634 seconds, our algorithm is 323.31 % faster than the HAF computation time of 4.937268 seconds. The verdict of our approach is +, as we have a good orientation and close grasping to the center of weight. The HAF approach suggested grasping points close to the edge and inside the point cloud, so even if there would be no collision, the grasp might fail by sliding off the edge.



(A) (B)

FIGURE 4.26: Paper cup, real model and resulting PC.



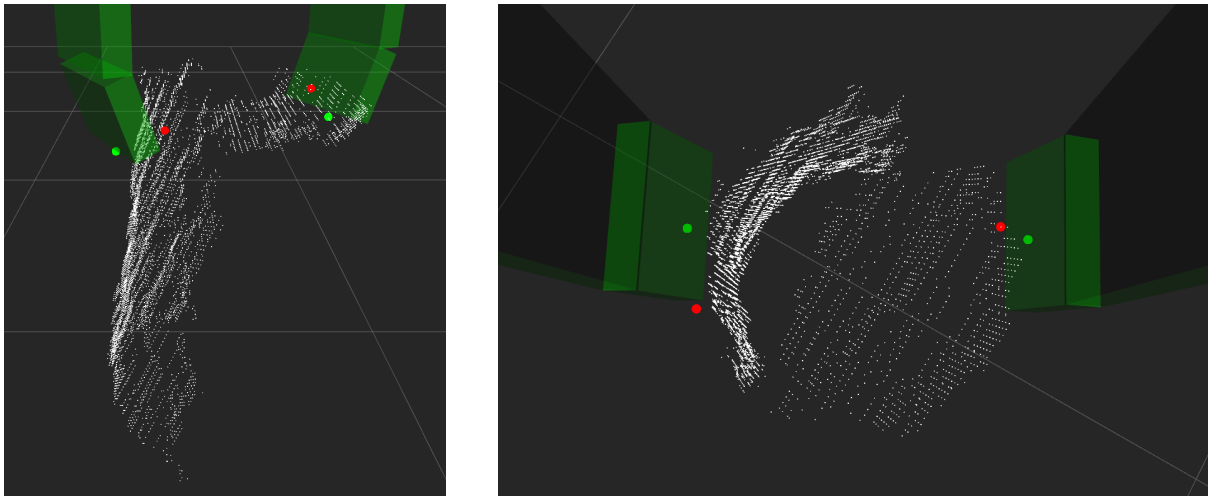
(A) Curvature for p_1^{tip}

(B) Curvature for p_2^{tip}

(C) Quality scores for p_1^{tip}

(D) Quality scores for p_2^{tip}

FIGURE 4.27: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ using the paper cup PC.



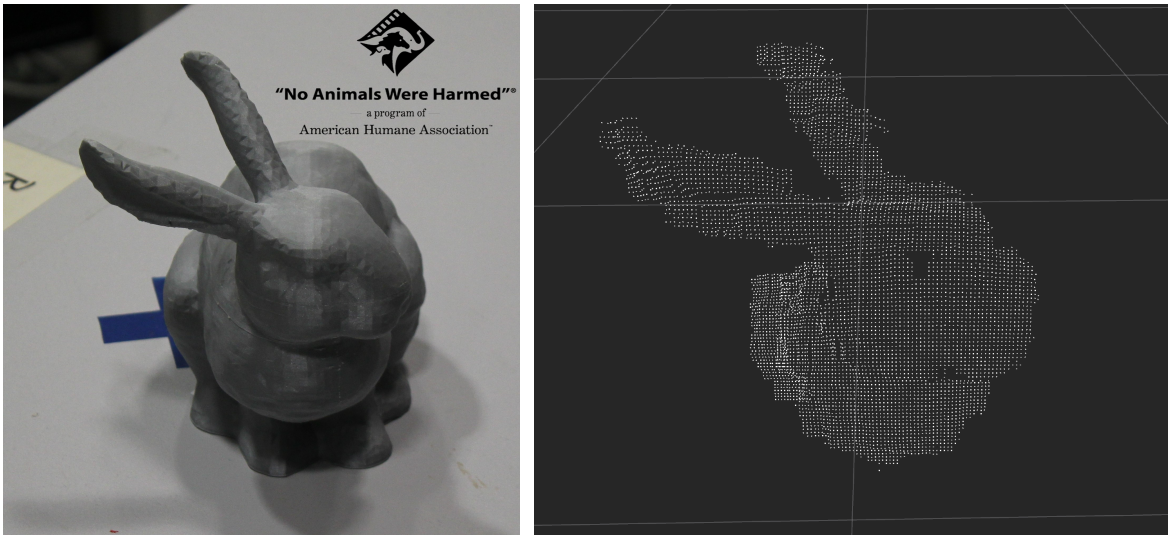
(A)

(B)

FIGURE 4.28: The calculated grasp for the paper cup.

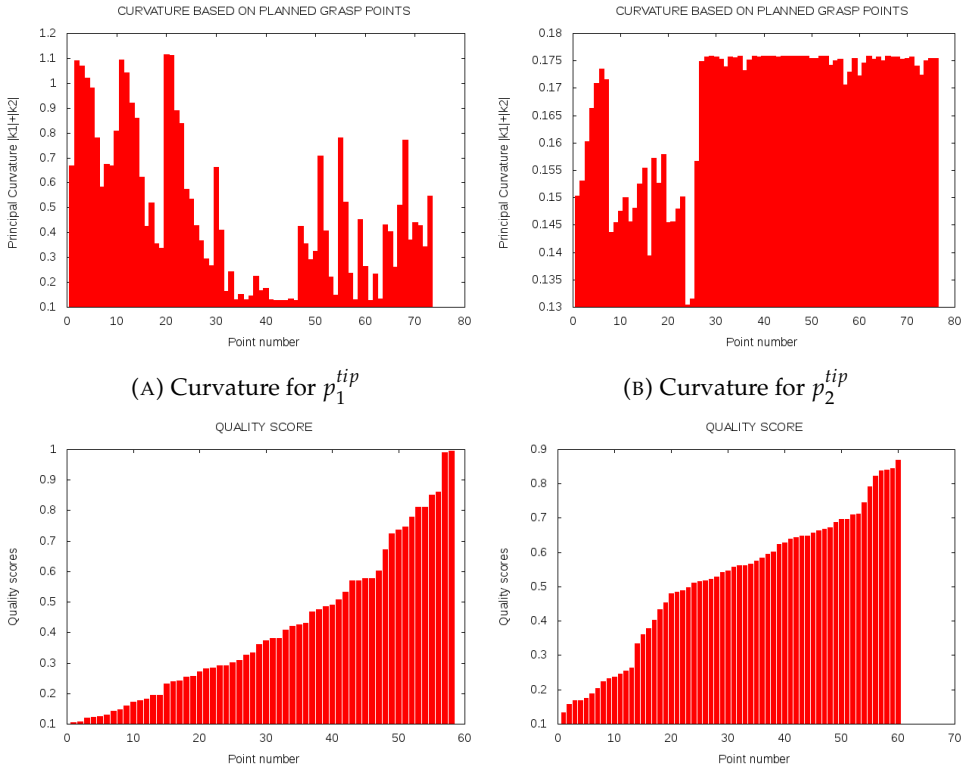
4.0.10 Stanford Bunny - front view

With a quality score of 0.0225819 and calculation time of 1.52609 seconds, our algorithm is 336.3 % faster than the HAF computation time of 6.658354 seconds. The verdict for our approach is 0 +. The algorithm founds a good grasping position, the bunny head. The orientation is suitable but the distance to the center of weight could be a difficulty. The HAF approach is not applicable as one grasp point is in the ear of the bunny and the orientation is inaccurate.



(A) (B)

FIGURE 4.29: Stanford bunny, real model and resulting PC, front view.



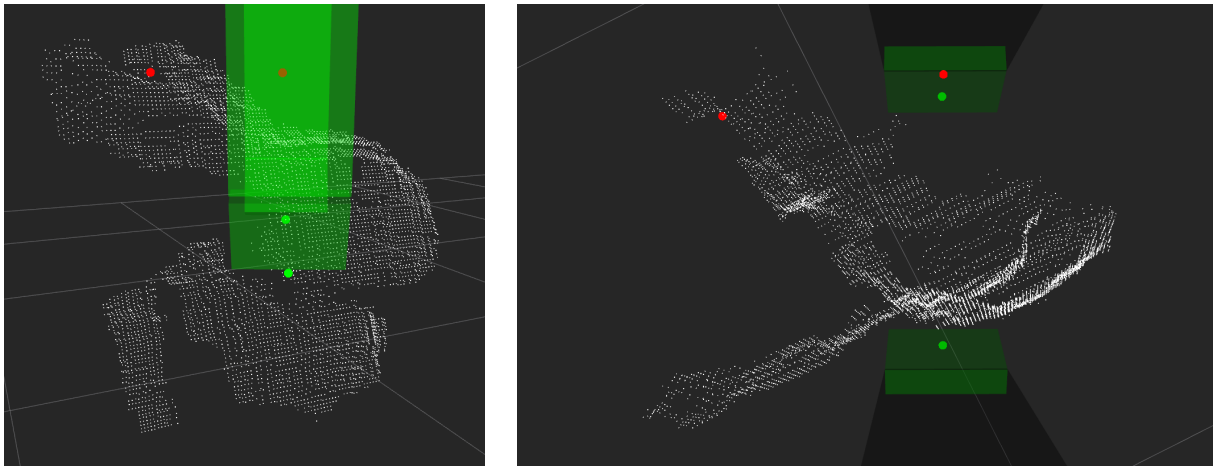
(A) Curvature for p_1^{tip}

(B) Curvature for p_2^{tip}

(C) Quality scores for p_1^{tip}

(D) Quality scores for p_2^{tip}

FIGURE 4.30: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ using the Stanford bunny PC, front view.



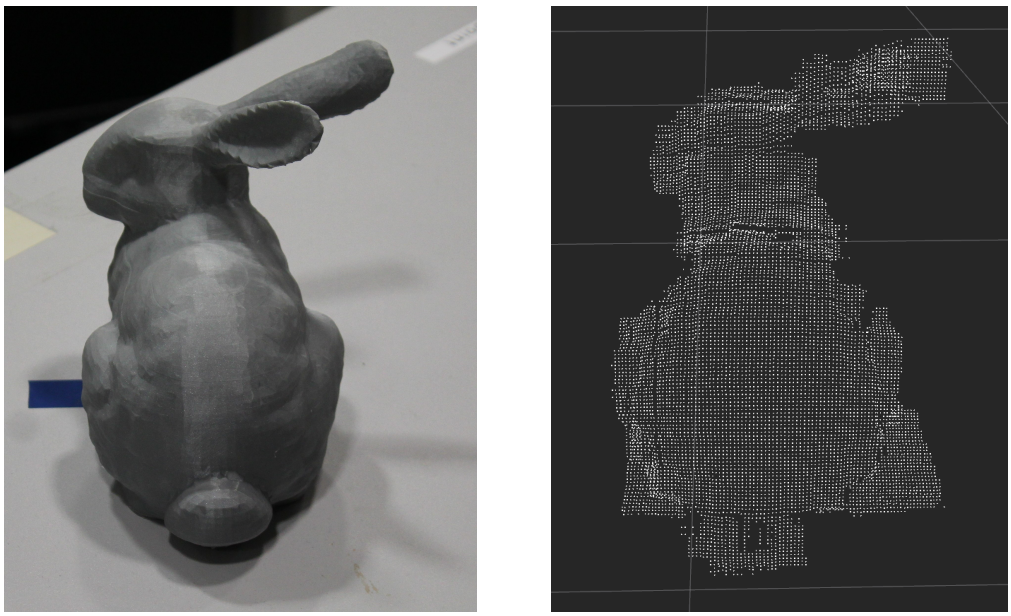
(A)

(B)

FIGURE 4.31: The calculated grasp for the Stanford bunny, front view.

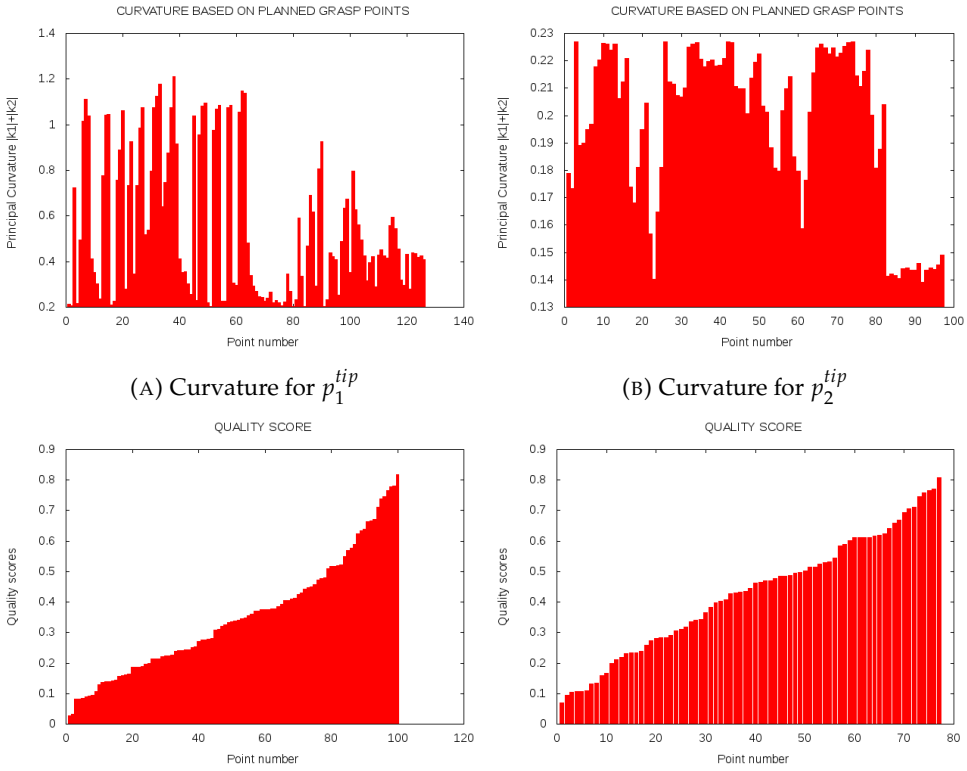
4.0.11 Stanford Bunny - back view

Our approach has a calculation time of 2.39068 seconds while HAV needed 5.372491 seconds, making our approach 124.72 % faster. The verdict based on our approach is -, as there was no collision free grasp possible (the gripper model is red). Depending on the requirements a not graspable object could count as + result, but in this case the subject is to find a executable grasp. The orientation of the HAF result is in the right direction to the ears, but the points are located over the ears, creating an empty grasp if the finger tips would close. Curvature and quality scores are shown for a random points p_1^{tip} and p_2^{tip}.



(A) (B)

FIGURE 4.32: Stanford bunny, real model and resulting PC, back view.



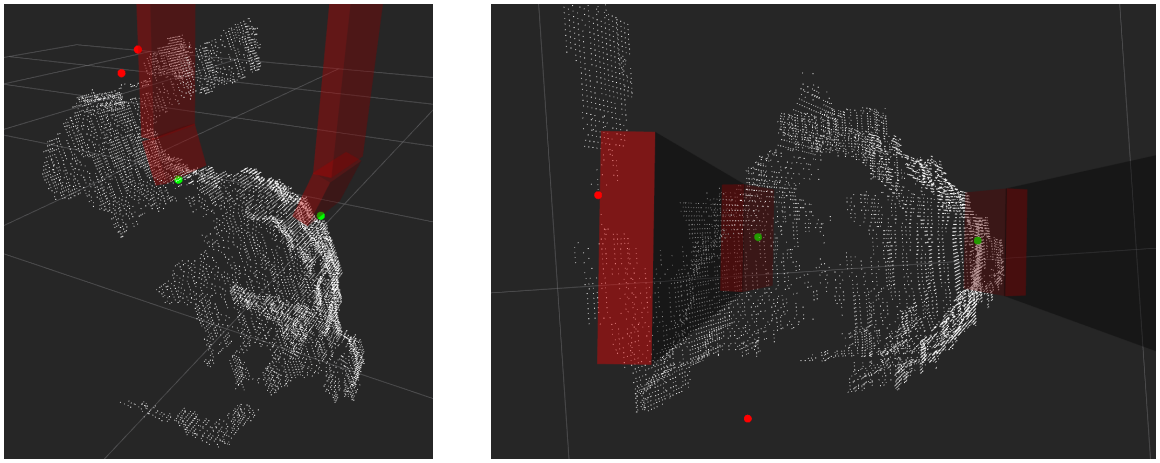
(A) Curvature for p_1^{tip}

(B) Curvature for p_2^{tip}

(C) Quality scores for p_1^{tip}

(D) Quality scores for p_2^{tip}

FIGURE 4.33: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ using the Stanford bunny PC, back view.



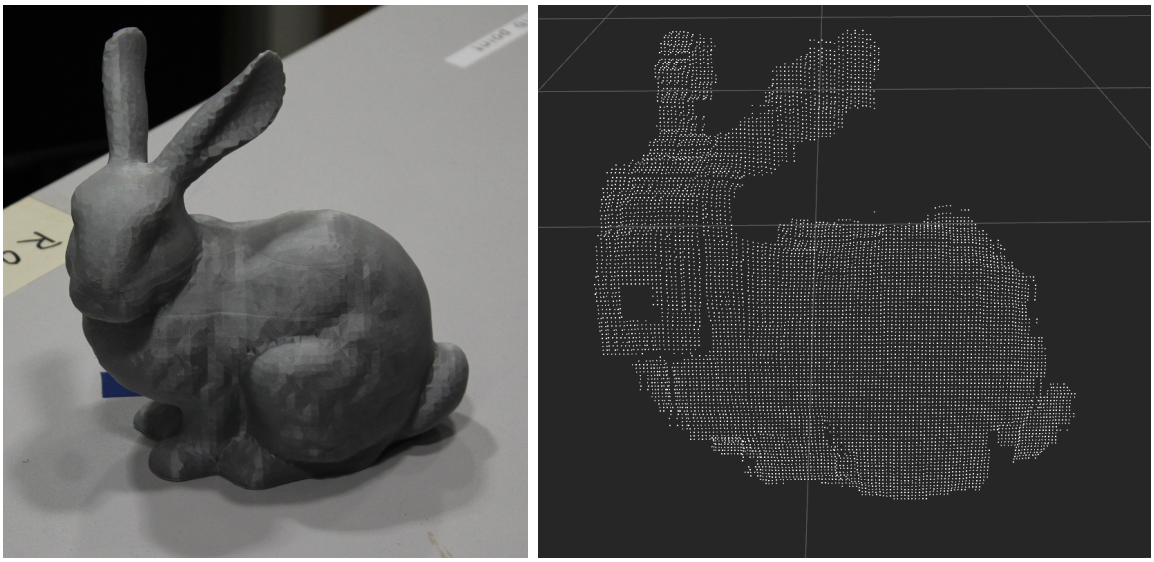
(A)

(B)

FIGURE 4.34: The calculated grasp for the Stanford bunny, back view.

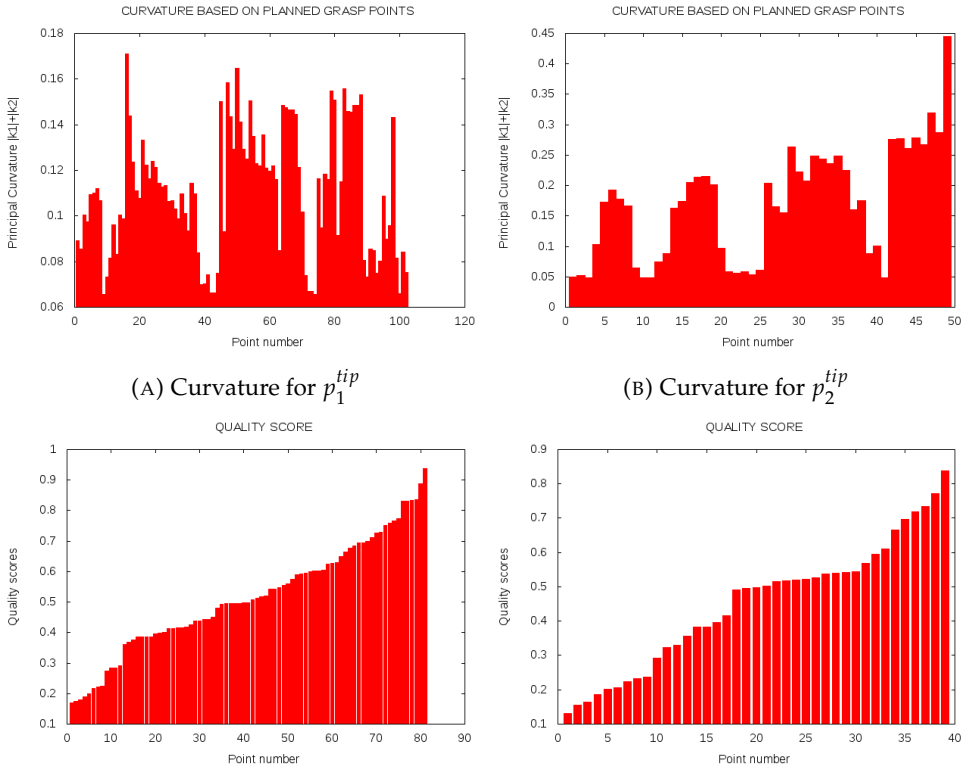
4.0.12 Stanford Bunny - side view

Our approach has a calculation time of 3.27662 seconds while HAV needed 6.684438 seconds, making our approach 104 % faster. The verdict of our approach is -. For all grasp points the grasp point evaluation detected collisions, so no valid grasp point is discovered. HAF lacks in the orientation as well as provokes a collision with the object as one grasp point is located in the point cloud (upper red point, 4.31). Curvature and quality scores are shown for a random points p_1^{tip} and p_2^{tip}.



(A) (B)

FIGURE 4.35: Stanford bunny, real model and resulting PC, side view.



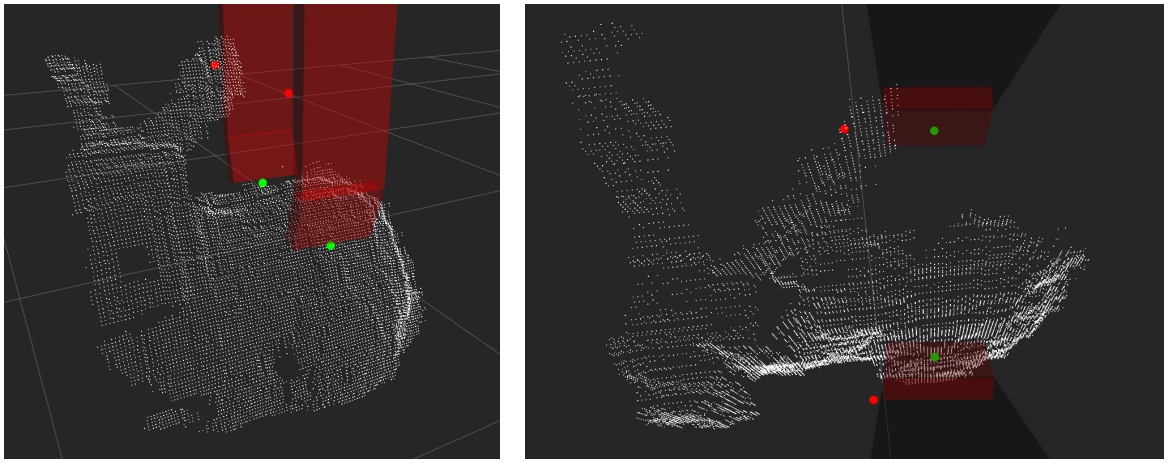
(A) Curvature for p_1^{tip}

(B) Curvature for p_2^{tip}

(C) Quality scores for p_1^{tip}

(D) Quality scores for p_2^{tip}

FIGURE 4.36: Calculated curvatures and quality scores for $p_{1,e}^{tips}$ and $p_{2,e}^{tips}$ using the Stanford bunny PC, side view.



(A)

(B)

FIGURE 4.37: The calculated grasp for the Stanford bunny, side view.

4.1 Conclusion

While the HAF grasping failed in most of our scenario configurations (1 average grasp, 3 risky grasps and 8 not graspable estimations) our approach was able to find 4 high probability grasp chances, 3 possible grasps, 3 change with a medium risk of failure and only 3 grasp were not possible, but this depends on the requirements how to count a non valid grasp verdict. Overall our approach needs less computation time than HAF.

5 Conclusion and Outlook

5.1 Conclusion

In this thesis we introduced our expanded self learning pipeline to achieve an autonomous robot system for 3D-model-reconstruction. The idea was to let the robot extracts objects from the environment information gathered by a vision sensor and interact with the objects to gain more information and finally create a digital representation in form of a 3D mesh.

To fulfil the requirements the incoming point cloud data is preprocessed and the objects of interest are extracted from the point cloud. Out of these objects exactly one is chosen and tracked. Using the point cloud of the object, the grasp pose detection tries to calculate a suitable gripper position in order to grasp the object. The approach uses the visible geometry of the object for a first estimation , curvature and obliqueness as additional features to filter graspable surfaces and validate if these surfaces are reachable without a collision. We have proven that our algorithm outperforms other grasp pose approaches in this kind of scenario.

In order to gain more information, the robot manipulates the object by rotating it and allow a full inspection. Every manipulation is followed by storing the partial point cloud until a full view is gathered. Finally, the partial point clouds are registered to each other and a 3D-mesh is created. This mesh can be used for every task in which a mesh is needed.

5.2 Outlook

As mentioned the mesh is useful for a lot of tasks. One example is the approach of [24] where several time maps are used to calculate the probability if parts of the environment are occupied or not. Beside the time maps an object recognition approach is mandatory for good results. In combination with our pipeline we would expand the time maps with a object map where the poses and sparse extends of objects are stored.

Our test results only relies to a subjective evaluation. To improve our results we need to use a bigger test scenario with hundreds of labeled test data to achieve more expressive values.

The outcomes of the grasp pose estimation are satisfying, but several improvements are possible. The first would be to enhance the rotation of the gripper. As the orientation is only adjusted in the geometrical grasp pose estimation, it might not be the best orientation if a grasp point was found in surface pose estimation. An extension for more general grasp pose estimations not only depending on one given grasp vector would also be a beneficial extension.

Despite all the potential enhancements the project introduced in this thesis is a success for autonomous systems with the need to increase their knowledge about the environment by itself.

Bibliography

[1] Holm, "Self-learning Pipeline for 3D Object Detection, Acquisition and Recognition in Autonomous Systems," *Research Projekt for Masters Programm Applied Research in Engineering Science*, Jan. 2018.

[2] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic Grasping of Novel Objects using Vision," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 157–173, Feb. 2008. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364907087172

[3] M.-H. Mousa, R. Chaine, S. Akkouche, and E. Galin, "Efficient Spherical Harmonics Representation of 3d Objects." IEEE, Oct. 2007, pp. 248–255. [Online]. Available: http://ieeexplore.ieee.org/document/4392735/

[4] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz, "Rotation Invariant Spherical Harmonic Representation of 3d Shape Descriptors," p. 10.

[5] J.-H. Chuang, N. Ahuja, C.-C. Lin, C.-H. Tsai, and C.-H. Chen, "A potential-based generalized cylinder representation," *Computers & Graphics*, vol. 28, no. 6, pp. 907–918, Dec. 2004. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0097849304001426

[6] N. J. Mitra, L. J. Guibas, and M. Pauly, "Partial and Approximate Symmetry Detection for 3d Geometry," p. 9.

[7] D. Terzopoulos, A. Witkin, and M. Kass, "Symmetry-seeking models and 3d object reconstruction," *International Journal of Computer Vision*, vol. 1, no. 3, pp. 211–221, Oct. 1988. [Online]. Available: http://link.springer.com/10.1007/BF00127821

[8] A. Makhal, F. Thomas, and A. P. Gracia, "Grasping Unknown Objects in Clutter by Superquadric Representation." IEEE, Jan. 2018, pp. 292–299. [Online]. Available: http://ieeexplore.ieee.org/document/8329926/

[9] M. Ohuchi and T. Saito, "Three-dimensional shape modeling with extended hyperquadrics." IEEE Comput. Soc, 2001, pp. 262–269. [Online]. Available: http://ieeexplore.ieee.org/document/924449/

[10] C. Rubert, B. León, A. Morales, and J. Sancho-Bru, "Characterisation of Grasp Quality Metrics," *Journal of Intelligent & Robotic Systems*, vol. 89, no. 3-4, pp. 319–342, Mar. 2018. [Online]. Available: http://link.springer.com/10.1007/s10846-017-0562-1

[11] A. ten Pas and R. Platt, "Using Geometry to Detect Grasp Poses in 3d Point Clouds," in *Robotics Research*, A. Bicchi and W. Burgard, Eds. Cham: Springer International Publishing, 2018, vol. 2, pp. 307–324. [Online]. Available: http://link.springer.com/10.1007/978-3-319-51532-8_19

[12] O. Kroemer, S. Leischnig, S. Luettgen, and J. Peters, "A kernel-based approach to learning contact distributions for robot manipulation tasks," *Autonomous Robots*, vol. 42, no. 3, pp. 581–600, Mar. 2018. [Online]. Available: http://link.springer.com/10.1007/s10514-017-9651-z

[13] S. H. Kasaei, J. Sock, L. S. Lopes, A. M. Tomé, and T.-K. Kim, "Perceiving, Learning, and Recognizing 3d Objects: An Approach to Cognitive Service Robots," 2018.

[14] X. Zhou, X. Lan, H. Zhang, Z. Tian, Y. Zhang, and N. Zheng, "Fully Convolutional Grasp Detection Network with Oriented Anchor Box," *arXiv preprint arXiv:1803.02209*, 2018.

[15] F.-J. Chu, R. Xu, and P. A. Vela, "Deep Grasp: Detection and Localization of Grasps with Deep Neural Networks," *arXiv:1802.00520 [cs]*, Feb. 2018, arXiv: 1802.00520. [Online]. Available: http://arxiv.org/abs/1802.00520

[16] OpenAI, "Learning Dexterous In-Hand Manipulation," *arXiv:1808.00177 [cs, stat]*, Aug. 2018, arXiv: 1808.00177. [Online]. Available: http://arxiv.org/abs/1808.00177

[17] A. t. Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp Pose Detection in Point Clouds," *arXiv:1706.09911 [cs]*, Jun. 2017, arXiv: 1706.09911. [Online]. Available: http://arxiv.org/abs/1706.09911

[18] M. Gualtieri, A. t. Pas, K. Saenko, and R. Platt, "High precision grasp pose detection in dense clutter," *arXiv:1603.01564 [cs]*, Mar. 2016, arXiv: 1603.01564. [Online]. Available: http://arxiv.org/abs/1603.01564

[19] D. Fischinger, A. Weiss, and M. Vincze, "Learning grasps with topographic features," vol. 34, 05 2015.

[20] "PCL - voxel grid filter," http://pointclouds.org/documentation/tutorials/voxel_grid.php, online; accessed 02-08-2018.

[21] "Point Cloud Library - principal curvature," http://docs.pointclouds.org/1.0.0/classpcl_1_1_principal_curvatures_estimation.html, online; accessed 02-08-2018.

[22] "MIT - principal curvature description," http://web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/node30.html, online; accessed 02-08-2018.

[23] "surface shape classes based on principal curvatures," http://homepages.inf.ed.ac.uk/rbf/BOOKS/FSTO/node28.html, online; accessed 02-08-2018.

[24] Holm, "SLAM in dynamic Environments," *Research Projekt for Masters Programm Applied Research in Engineering Science*, Feb. 2017.