CHALMERS
UNIVERSITY OF TECHNOLOGY

Grasping with a mobile manipulator using 3D vision

Software Implementation of a Grasp planner for a mobile manipulator using data from RGB-D camera

Master's thesis in Electrical Engineering

CHATELIN TRISTAN

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Grasping with a mobile manipulator using 3D vision

Software Implementation of a Grasp planner for a mobile manipulator using data from RGB-D camera

TRISTAN CHATELIN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2022

Grasping with a mobile manipulator using 3D vision
Software Implementation of a Grasp planner for a mobile manipulator using data from RGB-D camera
TRISTAN CHATELIN

Supervisor and Examiner: Yasemin Bekiroglu, Department of Electrical Engineering,
Co-supervisor: Hadi Hajieghrary, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The mobile manipulator known as DORA on which the experiments were conducted.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Abstract

Robotic grasping and manipulation is a well-studied topic, but often it's implemented using stationary robotic arms. Using a mobile manipulator for grasping and manipulation enables significant increase in the capabilities of a robot. This thesis tackles the challenging task of grasping a given object with a mobile manipulator. Our approach maintains a good trade-off between the arm motion and the base motion. This work is done using the data from a stereo camera which oversees the scene.
Firstly, an instance segmentation algorithm is used to recognise the objects which are available in the scene. Next, one object among the batch of detected objects is selected. Then, the mask of the selected object is used to generate a point cloud which only contains the points which belong to the object. This point cloud is then used as input for a pose estimation algorithm to find the 6D pose of the object. Then, a set of grasp poses is generated based on the detected object pose.
The most appropriate grasp pose is chosen from the set of grasp poses, knowing the actual pose of the gripper based on a defined selection criteria. The motion of all the joints of the robot is computed to follow an end-effector trajectory which is a straight line between the initial pose of the gripper and the selected pose. Our real robot experiments lead to successful grasp execution results with high grasp quality.

Keywords: mobile manipulator, ROS, robotic grasping, depth camera.

Acknowledgements

I would like to thank everyone who has contributed to this thesis.
I would like to thank my tutors Yasemin Bekiroglu and Hadi Hajieghrary, for letting me be a part of a project, as exciting and rewarding as this one. I really appreciated your guiding, helping and constant motivation through my work.

Tristan Chatelin, Gothenburg, July 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CNN	Convolutional Neural Network
GPD	Grasp Pose Detection
ICP	Iterative Closest Point
RANSAC	Random Sample Consensus
RRT	Rapidly Random Tree
SVD	Singular Value Decomposition
TAMP	Task And Motion Planning

Contents

List of Figures

List of Tables

1

Introduction

The aim of the master thesis is to enable a mobile manipulator (figure 1.1) to grasp an object on a cluttered table. The task of grasping an object is studied by many researchers with encouraging results. But the grasping is mostly carried out by fixed robots, which have two main limitations: First, the workspace of the robot is limited and therefore a lot of tasks such as grasping an object on a big table can be challenging due to reachability issues which can be addressed if the manipulator is mobile. The second limitation of using a fixed robot is that the number of possible tasks done by a fixed robot is more limited than with a mobile robot.

Adding a mobile base to robotic manipulator increases the degree of freedom which have the drawback of making the computation of the trajectory harder. Indeed, for the same motion of the hand, there are a lot of different ways to move the different parts of the robot. But this drawback is also an advantage in the sense that it enables to choose between different strategies to move the hand to the goal position.

Before the motion of the robot is computed, the goal position of the hand needs to be known. To do it, a depth camera, which is the ZED2 camera from Stereolabs is overseeing the table. This camera produces a depth image of the environment. Then, the pipeline presented in this report process the depth image and returns a set of grasp poses for the object that is needed to be grasped.

Once the set of grasp poses is determined, a grasp pose to be executed needs to be chosen from the set of grasp poses. Then, the motion of the robot towards the grasp pose is computed. But, the position of the robot needs to be known. In order to calculate the pose of the robot, we rely on markers placed on the robot. The data from the ceiling camera is used to compute the position of the robot from the detected pose of the AprilTag. **The process for grasping and manipulating a set of objects is: 1. The Fixed 3D camera looks at the table and the objects on it, detects and segments the objects. 2. A planning algorithm plans the sequence of picking-moving-placing of the objects on the table, i.e., bases on some optimization cost the robot decides which object it should pick and move. 3. A plan should be drawn to move the robot to the target. The plan includes the motion of the base and the arm. This has to be also considered in item 2, when the plan is drawn. 4. The robot should be able to guide itself (the hand) toward the grasp pose. One way to make this possible is to use the Camera on the wrist. 5. While the robot and its arm is moving toward the object to be grasped**

it can complete the partial point cloud of the object to be grasped. The grasp poses might be refined if needed using the completed point cloud in order to have a more successful grasp pose.

Figure 1.1: The robot used in this work: DORA

The pipeline is split into two parts. The first part generates a complete point cloud and a set of grasps for each object. This part is run only one time for each object. The second part of the pipeline computes the pose of the object in the scene and computes a set of grasp poses in the camera frame.

The following flowchart shows the general architecture of the pipeline:

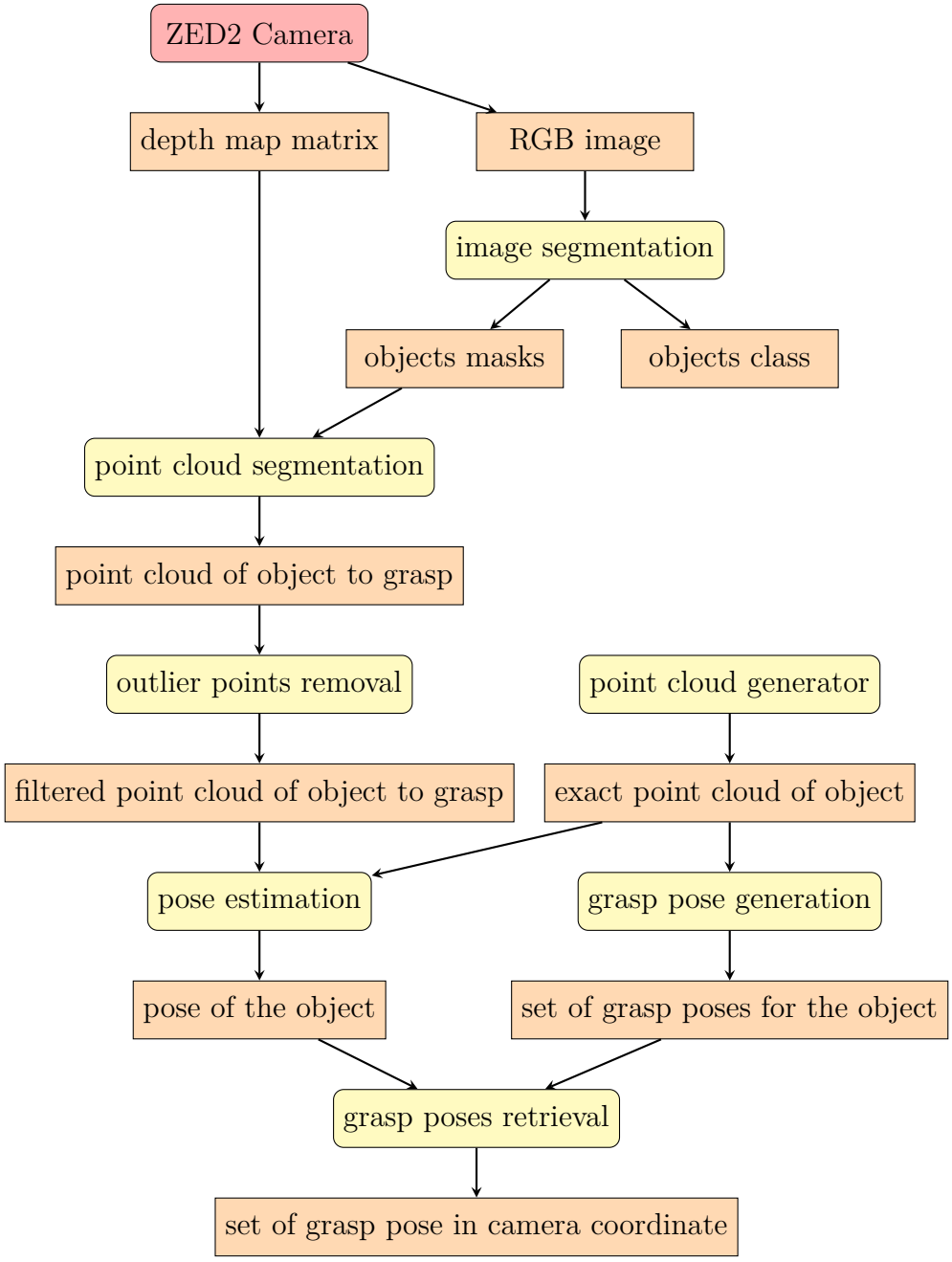

Figure 1.2: General architecture of the pipeline

This gives the general idea of how the pipeline works. The following parts of this report will give more information about each part of the pipeline.

2

Literature review

In this section, we present a brief overview of the previous work on the areas of robot perception, motion planning and control.

2.1 Perception and grasp planning

There are two types of grasp planners. The first type is used for known objects. This type of planner uses the geometry of the objects to generate a set of viable poses. A well-used grasp planner for known objects is Grasp Pose Detector (GPD) [1].
The second type of grasp planner generates grasp poses for unknown objects using a partial point cloud of the objects. A well-used grasp generator for unknown objects is 6-DOF GraspNet [2]. This generator is based on three networks. The first one generates a large set of antipodal poses. A second network assesses the grasps and removes the worst grasps. Then a third network refines the grasps.
Another approach for unknown objects is to find geometric similarities between an object and a set of known objects [3]. Then, the set of grasps generated for the object is based on the grasps for the known objects.
This article is a review about robotic grasping using the data from an RGB-D camera [4]. This review was helpful to determine the overall architecture of the pipeline. It also helped to find the open-source algorithms to perform each stage of the pipeline.

2.2 Path planning

The path planning for the mobile base can be generated using a rapidly random tree algorithm (RRT). This algorithm generates a set of possible trajectories between two points and then choose the trajectory which is the shortest. The paper [5] improves the RRT algorithm by adding the task of grasping an object on a table which is between the initial pose of the robot and the goal position.
This review presents the different algorithms known for path planning. The presented algorithms are compared each other in term of computation time [6]. There are some algorithms which enable to avoid static or moving objects.
This paper tackles the problem of grasping a moving object [7]. The method computes a trajectory of the arm until the object and moves the arm to the object. Then, the pose of the hand is adjusted to the new pose of the object. This paper introduces the idea of heuristic function. The purpose of a heuristic function is to improve the efficiency of the search by guiding it in promising directions. A typical

method for formulating a heuristic is to use the results from a simplified problem, such as from a lower-dimensional version of a similar search problem which enables to give a direction to the algorithm which enables to improve the efficiency of the search algorithm by guiding it in promising directions.

2.3 Whole-body control

The main control objective for a mobile manipulator is to follow the desired motion of the hand. The motion of the mobile base and the joints is then computed to satisfy the target position of the hand. For it, two main methods are used.
The first type is based on reinforcement learning [8]. This method computes the motion of the robot to follow a given hand trajectory. The idea of this method is as follows. Firstly, a set of random motion of both the joints of the arm and the mobile base is generated. Then, the motions which are the closest to the desired hand motion are randomly mutated. This creates a new set of motion which is also mutated, and so on. This algorithm is run until the result is satisfactory. This method enables to don't have to compute analytically the motion of each joint but it requires a high computation time and it needs to run the learning process for each new motion of the hand.
The second method uses control strategy based on the model of the robot. A promising approach is to use a global control strategy to compute the whole-body motion. Then, a local control strategy make sure that each part of the robot follows the desired motion. The global control can be done using model predictive control and the local control can be done using proportional integral derivative control [9, 10]. This method enables to have good follow-up results.
The paper [11] uses a quadratic program based inverse kinematics (IK) controller to generate the global motion of the robot. But it doesn't use a local control for each part. This approach enables to make a trade-off between the trajectory error and the intensity of the command. This work also enables to avoid the obstacles in the path of the robot.

2.4 Task and motion planning

Task and Motion Planning (TAMP) is an interesting field of research. It focuses on planning operations for a robot in environments containing a large number of objects. The operations are discrete events like moving the robot from one position to another or to change the state of the objects. The state of an object is its pose and the fact that it's hold on not by the robot. TAMP is based on discrete events and focuses on how to reduce the execution time or energy consumption to carry out a set of tasks. The review [3] gives a way to formalize the actions and introduce some algorithms about how to address the TAMP problems.

3
Hardware

In this chapter will be introduced the different depth camera technologies, the advantages and disadvantages of each technology. Based on the advantages and disadvantages of each technology, the choice of the depth camera used in this work will be explained.
After, the different parts of the mobile manipulator and their characteristics will be presented.

3.1 Depth Camera

3.1.1 Depth Sensing overview

2D cameras return a planar image of the environment, it's composed of a set of pixels. Each pixel contains a tuple value which is (red, green, blue). A depth camera returns the same amount of pixels, but each pixel contains the tuple (red, green, blue, depth). Where the depth value is the distance between the camera and the pixel in the environment.
Recently, depth measurement has gained importance in many applications, like automation, robotics and autonomous vehicles. There are different technologies to generate depth images. Here will be shown an overview of the most important methods.

3.1.1.1 Stereo Vision

Stereo vision mimics human vision. This technology uses two cameras with a fixed distance between them (baseline in the following scheme). It uses triangulation to know the distance between the camera and the pixel. To do it, the software of the camera looks for the corresponding features in the two images to identify the same pixels in both sides. Each camera capture the scene with a slightly different angle. The farther an object, the most its feature pixels are shifted to the left on the left image. This shift is called disparity and is shown in the following scheme. Then, the depth is computed for every pixel using disparity and the depth map is generated.
There are two types of stereo vision system. The first one is passive which only uses the ambient light to illuminate the objects. The flaw of passive stereo cameras is that it doesn't provides accurate depth map for poor-featured environment like white walls. The active systems project infra-red on a semi-random patterns to the environment. The dots are then used as a feature to find the corresponding pixels.

The added features are recognised by the camera and then enable to improve the quality of the depth. The flaw of active stereo vision is that it's not possible to use multiple cameras since they will infer each other.
The advantage of using a stereo camera is its robustness over changing light conditions. It's also possible to use multi-camera setup (only for passive stereo camera). Two drawbacks of a stereo camera is that the point cloud is noisy and that it don't returns a good depth value for transparent objects. The range of the camera and the accuracy rely on the model of the camera.

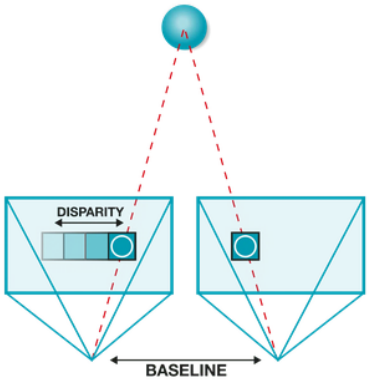

Figure 3.1: Stereo camera conceptual scheme

In this work, a passive stereo camera is used because in the future, two cameras will be used, one on the ceiling and one on the wrist of the robot.

3.1.1.2 Structured light

Structured light is based on a known light pattern which is projected to the environment by a projector. One or more cameras record the environment. The pose of the camera in regard to the projector is known. The difference between the projected pattern and the distorted pattern seen by the camera is used to compute the depth of the environment. This method doesn't give good results for transparent objects, highly reflective surfaces or long range. It cannot work with multiple cameras or when external light sources emit in the same wavelength. This method has the advantages of having a high accuracy within short range and to be low cost in regard to other technologies.

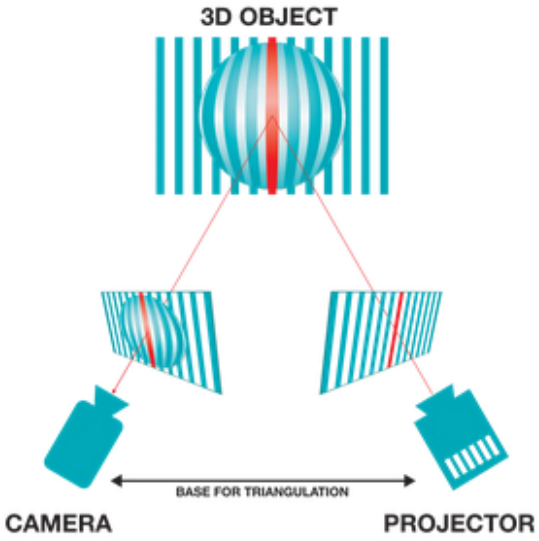

Figure 3.2: Structured light conceptual scheme

3.1.1.3 Time of flight

The time of flight is based on the measurement of the time taken by a light wave to travel from an emitter to an object, to reflect on it and to come back to a receiver. Since the speed of light is known, the distance between the camera and the object is known. Usually, the light is emitted by a LED or a laser in the infrared spectrum. Similarly to structured light, cameras using time of flight can infer each other, so it's not possible to use several time of flight cameras at the same time. The advantage of this technology is the high accuracy of the point cloud.

3.1.2 ZED2 Camera from Stereolabs

For this project, the used cameras is the ZED2 Camera [12] from Stereolabs. This camera is a passive stereo camera. It uses Neural Depth Sensing to improve the accuracy of the point cloud. The camera also embeds some sensors, which are a gyroscope, an accelerometer, a magnetometer, a barometer and a thermometer. The output resolution of the camera can be chosen between (672 x 376), (1280 x 720), (1920 x 1080) and (2208 x 1242). The depth range of the camera is from 0.3 m to 20 m. The field of view is 120 ° in the horizontal axis. The depth accuracy is less than 1 % up to 3m and less than 5 % up to 15m. The following picture shows the ZED2 camera.

Figure 3.3: ZED2 Camera from Stereolabs

The camera returns three matrices which are a RGB image, a depth map and a RGB-D image. On the following picture, the upper left picture shows the RGB image, the bottom left figure shows the depth map and the right image shows the coloured point cloud retrieved with the camera.

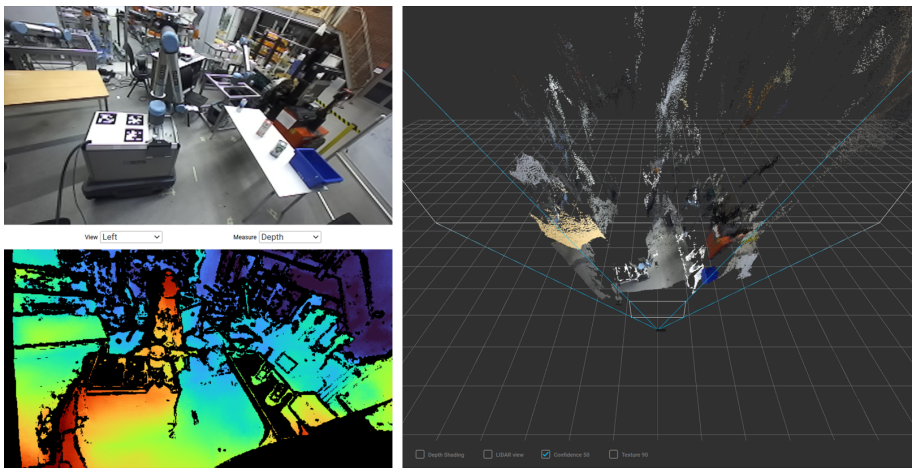

Figure 3.4: RGB image, depth map and coloured point cloud retrieved with the ZED2 camera

The point cloud from the camera is noisy. But this noise can be reduced by using an algorithm implemented by Stereolabs. This algorithm uses a neural network. This algorithm requires much more computation time to retrieve the point cloud, but improves greatly its accuracy.

3.1.3 Pose of the camera

Since the camera has a high accuracy in the field of view of 3 m, I decided to put it near of the table to be close to the objects whose point cloud needs to be accurate. But the camera should be far enough of the table so that the mobile manipulator can be between the fixture of the camera and the table. The camera should also see the top of the robot to see the AprilTag and then find its pose. The following picture shows an appropriate pose of the camera.

Figure 3.5: Pose of the camera with regard to the robot and the objects. The camera is in the upper right corner of the picture

3.2 Robot

The robot used in this work is a mobile manipulator. It combines a mobile base and a robotic arm which is fixed on the top of the robot. At the tip of the arm is hang a gripper. The mobile base is MIR200 from MIR company. The robotic arm is UR10 from Universal Robots. The gripper is RG6 from OnRobot. The next picture shows the complete robot. Each part of the robot will be presented more deeply in the following sections.

Figure 3.6: The mobile manipulator used in this work. The mobile base is MIR200, the arm is UR10 and the gripper is OnRobot RG6

3.2.1 Mobile base: MIR200

The MIR200 is the mobile base for the DORA robot. It can carry 200 kg of payload and pull 500 kg. The mobile base can drive up to 1.1 m/s. Different top modules can be mounted on it. This mobile base contains two laser scanners, two 3D cameras which are Intel RealSense and four ultrasonic sensors. These sensors are useful to understand the environment, as well as motion planning and obstacle avoidance.

Figure 3.7: Mobile base: MIR200

3.2.2 Arm: UR10

The UR10 arm is widely used in the field of robotics, both for fixed robots and mobile manipulators. The UR10 arm can carry a payload of 10 kg. The arm has 6 revolute joints and can reach up to 1.3 m.

3.2.3 Gripper: OnRobot RG6

The gripper used in this work is the RG6 from OnRobot. This gripper enables to pick up a payload of 6 kg in the axis of the gripper and 10 kg in the axis of the fingers. The maximum distance between the two fingers is 16 cm.

Figure 3.8: Gripper: OnRobot RG6

The shape of the gripper produces that when the distance between the fingers vary, the distance between the tip of the fingers and the base of the gripper also vary. The next left figure shows the pose of the tip of the fingers when the gripper is closing. The right figure shows the distance between the tip of the fingers and the base of the hand (h) in regard to the distance between the two fingers (w).

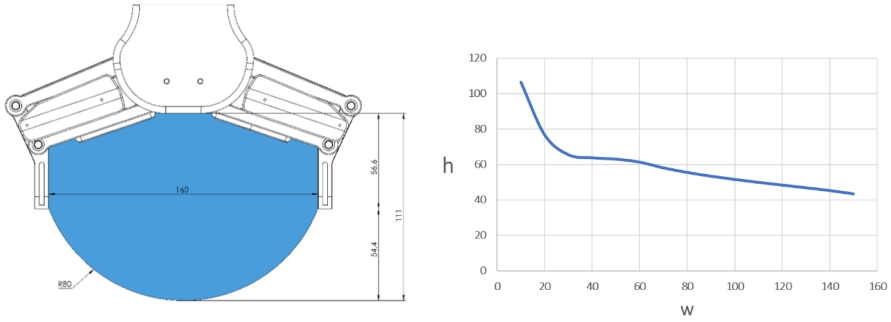

Figure 3.9: Link between the width and the height of the gripper

This curve is not linear and needs to be taken into account later in this work.

4

Point cloud generation and grasp pose generation

The first step which is point cloud generation produces a comprehensive point cloud of an object. Then, in the second step, which is grasp pose generation, the point cloud generated in the first step is used to generate a set of grasp poses for this object.
These two steps are only run once for each object. For each object, two files are generated, the first one contains the point cloud and the other contains the set of grasp poses. The exact point cloud will then be used in the pose detection step. The grasp poses set will then be used in the grasp pose retrieval step.
The main advantage of computing the grasp poses aside is that it's not needed to tackle the issue of computation time. Indeed, the grasp pose generation step is time consuming and requires a long computation time to generate grasps with a high score. Then, by computing the grasp poses aside, the execution time is reduced and the quality of the grasps is improved.

4.1 Point cloud generation

In the file "add_new_object_to_database.py" are two functions "generate_cuboid" and "generate_cylinder" which enable to generate whole-body point cloud for cuboid-shaped and cylinder-shaped objects. These two functions then save the point cloud of the object in the directory "objects". The output of these functions can be replaced with any point cloud in the format (x, y, z) where the coordinates are in meter.
In the library PCL (Point Cloud Library), there is a feature which enables to generate point clouds from 3D mesh models. The point cloud generated with this method can then be used in this pipeline. This will allow to grasp and manipulate YCB objects using this pipeline.

4.2 Grasp pose Detection

A grasp pose detection method is a method that generates a set of poses for the gripper fingers to grasp an object. The grasp pose detection method uses point cloud data to generate the grasp poses.

4.2.1 Theory about GPD

Grasp Pose Detection (GPD) [1] is a grasp detection method which is broadly used in the field of grasping. GPD process point cloud to find grasp poses. When it's used for only one object, it generates the grasp poses in 3 steps:

1. A large set of candidates is generated. These grasp poses are chosen as evenly as possible all around the object.
2. A Convolutional Neural Network (CNN) takes as input the point cloud of the object and generated grasps. The CNN assesses whether or not each candidate grasp is a grasp. A candidate grasp is a grasp if the hand's body doesn't collide with the point cloud and if the hand's closing region contains one or more point from the point cloud. The CNN also return a score for each grasp.
3. The grasp candidates which are not grasps are removed. The grasps with a low score are also removed.

This algorithm can also handle partially occluded and noisy point cloud. It can also manage to generate grasp poses in a densely cluttered scene. This algorithm presents a success rate of 93 % for robotic experiment in a dense cluttered scene.
I chosen to use this algorithm because it enables to generate qualitative grasps for objects of all shapes.

4.2.2 Use of GPD

Then, a grasp pose detection algorithm (GPD) is used to compute the grasp poses from the generated point cloud. This algorithm is run inside the function "generate_grasp_poses" which is inside the file "add_new_object_to_database.py". This algorithm generates a set of grasps for each object. Each grasp is stored in the format:

$$[x,\ y,\ z,\ r_{11},\ r_{12},\ r_{13},\ r_{21},\ r_{22},\ r_{23},\ r_{31},\ r_{32},\ r_{33},\ \mathit{width\ gripper},\ \mathit{success\ mark}]$$

Where x, y and z represent the coordinates of the red cross in the following picture. The r_{ij} represent the rotation matrix of either the red, green or blue arrow. This choice is made in the file "eigen_params.cfg" which is in the directory "GPD_parameters" by changing the parameter "hand_axes" to 0 for the red arrow, 1 for the green arrow and 2 for the blue arrow. The parameter is already set to 2 in the file.

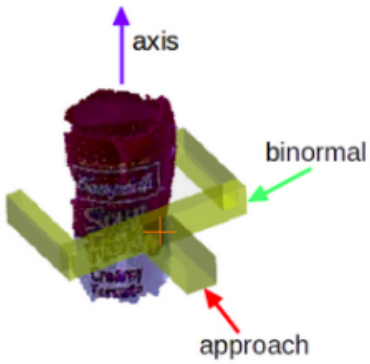

Figure 4.1: GPD Grasp pose referential

In the file "eigen_params.cfg" some parameters can be changed in order to vary the quality of the grasps, the spacing between each grasps and the number of stored grasps. The parameter "num_selected" directly changes the number of lines in the grasp pose file. The file "hand_geometry.cfg" needs to be fed with the geometrical parameters of the gripper. The following picture shows 20 grasps generated for a box:

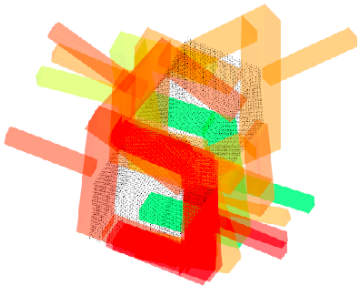

Figure 4.2: Grasp poses generated with GPD for a box

The computation time to generate the grasps vary between several seconds to a few hours for an object relying on the quality of the grasps that is requested. After the point cloud generation and the grasp pose generation is completed, two files "point_cloud_name_object.txt" and "grasps_name_object.txt" are created in the directory "object".

5

Image segmentation and point cloud segmentation

In this chapter will be presented the segmentation algorithm used for this work, the reason why this algorithm was chosen and the results obtained after this stage. After, the point cloud segmentation and outlier point removal stages will be explained and the results of these stages will be presented.

5.1 Image segmentation using Detectron2

5.1.1 Object detection, Semantic Segmentation and Instance segmentation

Object detection, semantic segmentation and instance segmentation are three computer vision techniques which enable to identify and locate the objects in an image. The first one consists of identifying and locating the objects in an image. Semantic segmentation consists of a specific categorisation of each pixel in an image. Instance segmentation consists of detecting and delineating each object in an image.

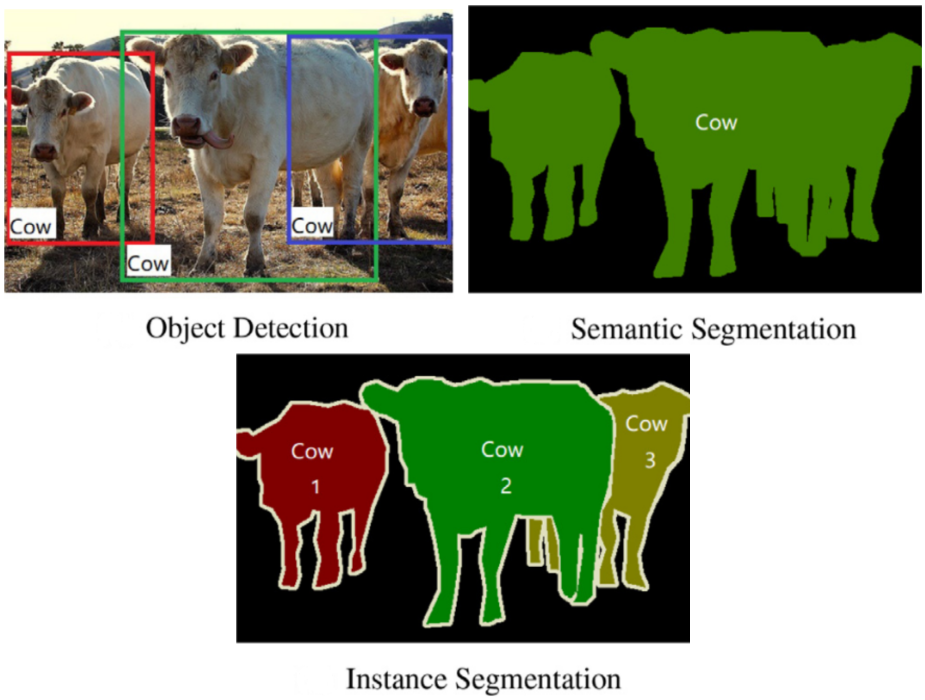

Figure 5.1: Object detection, Semantic segmentation and Image segmentation examples

5.1.2 Detectron2

Detectron2 [13] is a python library which detects and segments the objects which are on an image. It works with a few layers of convolutional neural network. The output of Detectron2 is a list of the objects detected on the scene and for each object a mask and a percentage of certainty is also returned. Detectron2 embeds different projects which enable to carry out different tasks like object detection, instance segmentation, person keypoint detection or panoptic segmentation. In the scope of this work, only object detection and instance segmentation will be interesting. In Detectron2, the object detection is done with Faster-RCNN. The instance segmentation is done with Mask-RCNN. These two sub-parts of Detectron2 embed different pre-trained layers with different output quality and computation time. In this software implementation, the layer "R50-FPN" of Mask-RCNN is used. This layer was chosen because it returns a good detection quality and a good mask quality.
In addition, it's possible to add new objects to the set of objects that Detectron2 can detect, that can be useful for this application. Indeed, it's possible to teach the algorithm to detect and classify several objects which look quite similar like two different mugs and then use the right point cloud and grasp pose set.
It's possible to reduce the computation time of this part of the pipeline. Indeed, in this pipeline, the detection and the segmentation are carry out for every regions of interest. Firstly, Faster-RCNN could be run to carry out object detection. Then run Mask-RCNN to segment only the region of interest that contains an object which belongs to the list of graspable objects. Doing like that, the computation time will be reduced. It could also be possible to improve the quality of the masks if Detec-

tron2 compute them without down-sampling.
The following picture shows a picture with the masks and the class returned by Detectron2 with an instance segmentation layer:

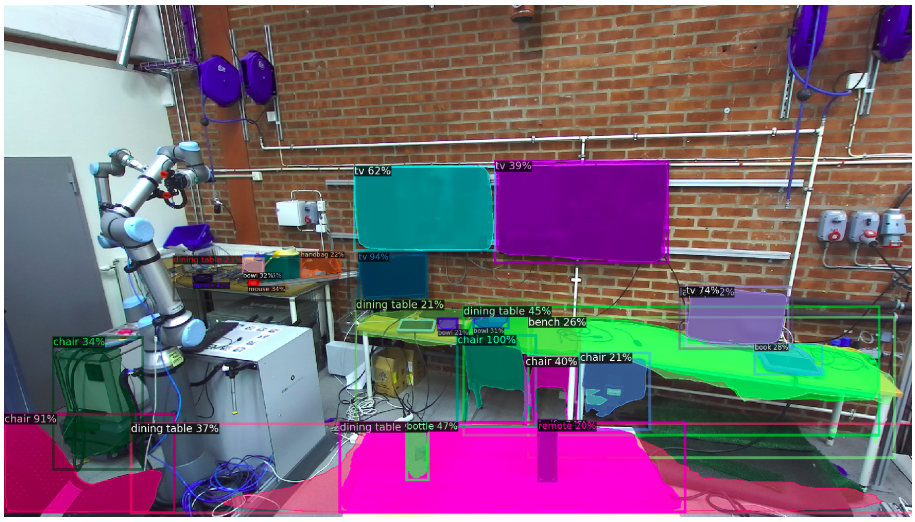

Figure 5.2: Image segmentation generated with Mask-RCNN

One can see on the picture that the objects are not all detected with the right name. This issue can be fixed by retraining the model with the objects which are in the picture.
Detectron2 have two other advantages for this application. The first one is that it can detect the objects in a cluttered environment. The following picture shows the masks generated in a cluttered environment with a low threshold:

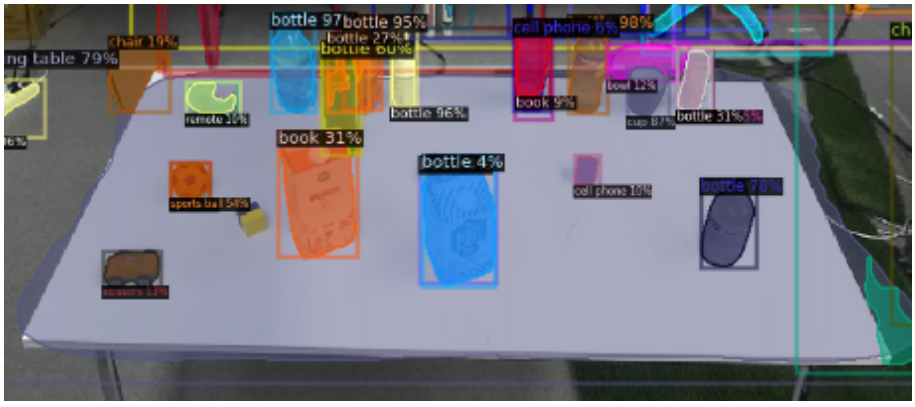

Figure 5.3: Image segmentation generated by Mask-RCNN with a low threshold

Secondly, to add a new object in the used Detectron2 model, with only 100 hand-labeled images, the trained model can retrieve a fairly good mask quality and a high detection success rate. This enables to add a new object to the list of detectable objects without a huge amount of work. Moreover, the new model can be trained rapidly. Indeed a Detectron2 model is trained at the average speed of 60 image/s. The following curves show the quality of a model in regard to the number of training pictures of each object:

Figure 5.4: Plot of the efficiency of the model in regard to the size of the training set

5.2 Point cloud segmentation

Once the mask of the object that is wanted to be grasp is generated, it's needed to generate a point cloud with only the points which belongs to the object. To do it, for every pixel in the mask, the tuple (x, y, z) is fetched in the point cloud. Then, this tuple is added to the list of points which belongs to the object.

The figure on the right shows the mask obtained with Detectron2 for a box. The three following pictures show the result of the point cloud segmentation for this box. The 3D point cloud is plotted from three viewpoints:

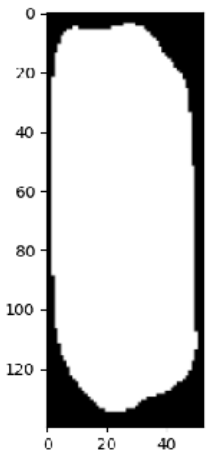

Figure 5.5: Mask of the box

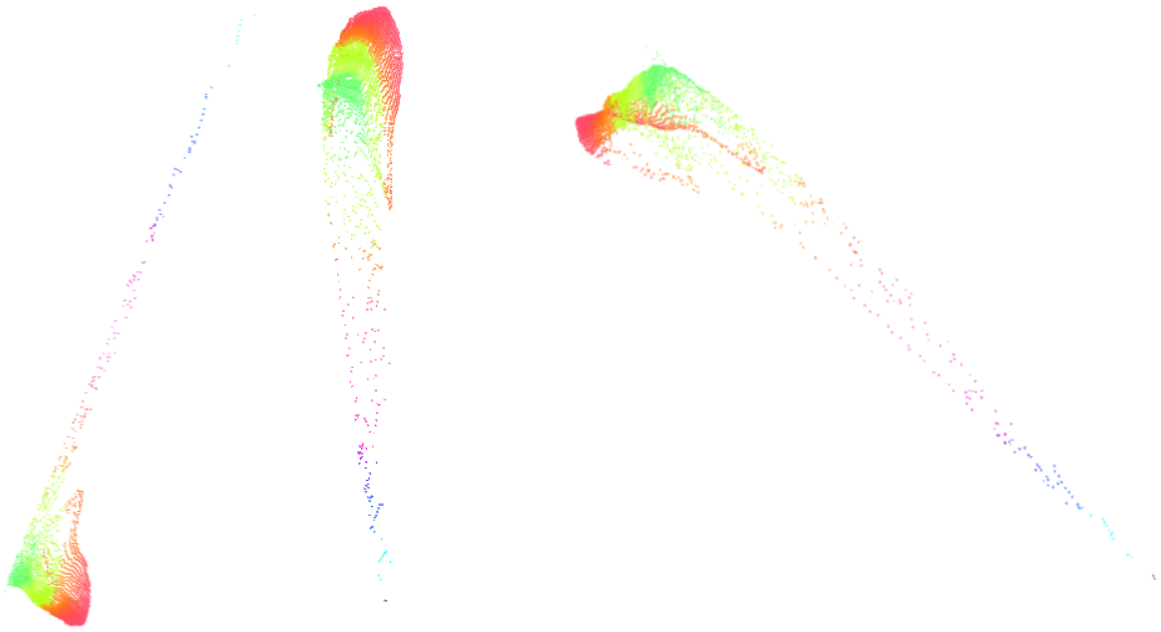

Figure 5.6: Point cloud of the box from different viewpoints

As we can see on these pictures, there is a set of outlier points which belongs to the vicinity of the object which are stored. These outlier points are due to the mask accuracy of Detectron2. These points will be removed in the outlier point removal stage.

5.3 Outlier point removal

To remove the outlier points, a statistical outlier point removal algorithm is used. This algorithm removes the points which are further away from their neighbours compared to the average for the point cloud. This algorithm takes two input parameters which are:

- nb_neighbors: specifies how many neighbours are taken into account in order to calculate the average distance from a given point. In the following picture, this parameter is set to 5.
- std_ratio: allows setting the threshold level based on the standard deviation of the average distances across the point cloud. The lower this number the more aggressive the filter will be. In the following picture, this parameter is set to 0.05.

The following picture shows the point cloud after the outlier points removal step:

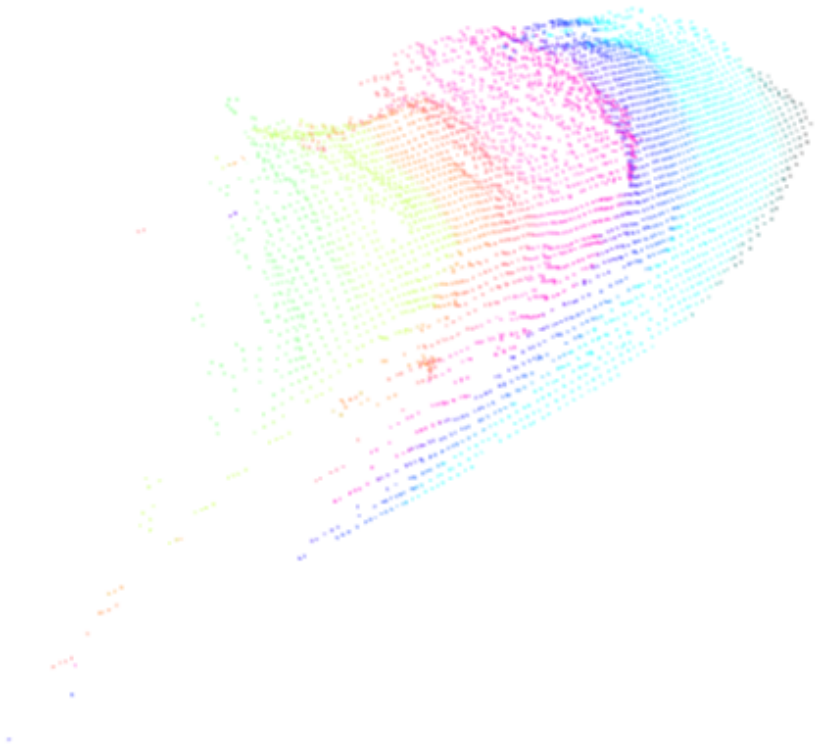

Figure 5.7: Point cloud of the box after outlier point removal step

As we can see in this picture, the point cloud now only contains the points which belong to the box.

6

Pose estimation and grasp pose retrieval

Once the point cloud of the object has been generated, the position of the object can be computed. To achieve this, the whole-body point cloud of the object is loaded from a file in the folder "objects". Then, two algorithms are used to compute the pose of the object. The first one is RANSAC global registration, which takes as input the down-sampled exact point cloud and the retrieved point cloud to generate an approximate pose of the object. Then a point-to-point Iterative Closest Point (ICP) algorithm refines the pose of the object.

6.1 Pose estimation

Iterative Closest Point (ICP) is an algorithm that returns the spatial transformation between two point cloud. It can align 2 or 3 dimensions point cloud. It would be possible to find the spatial transformation using brute-force search, but this method would be extremely time consuming. Then ICP algorithm tackles the problem of aligning point cloud in a more efficient way.
ICP computes the spatial transformation in two steps. Firstly, the two point clouds needs to be roughly aligned. This task is done by the global registration step and is generally done with down-sampled point cloud. The output from this step is an inaccurate spatial transformation between the two point cloud. I have chosen to use RANSAC ICP to carry out this task. The second step refines the spatial transformation, this step is done using the full point cloud. I have chosen to carry out this task using point-to-point ICP. There are other variant of ICP algorithm like point-to-plane ICP or coloured ICP.

6.1.1 Global registration

6.1.1.1 RANSAC ICP

Random Sample Consensus (RANSAC) is a type of ICP algorithm which deals with noisy point cloud. It can find the spatial transformation even if there are outlier points in the point cloud. RANSAC ICP separates the data into inlier and outlier points. The number of iterations of this algorithm is chosen before it starts to iterate. Each iteration of this algorithm contains four steps. The algorithm will be explained with an example, the aim of the example is to fit a 2D line in a 2D point cloud. The following two figures show the results from two different iterations of

the algorithm with the same point cloud and the same model. The four steps are as follow:

1. A sample of points is randomly chosen in the point cloud. The number of chosen points rely on the geometry of the model. In the example, two points are chosen since we want to fit a line. In the following picture, the two blue dots rely to this sample.
2. A model of the shape is generated. In the following picture, it's the black line.
3. The points which are close to the model are inserted in the inlier point set. These points are the green dots in the example.
4. The model is scored, the score is given by the number of points in the inlier set divided by the total number of points in the point cloud.

When all the iterations are done, the model with the highest score is chosen as the result.

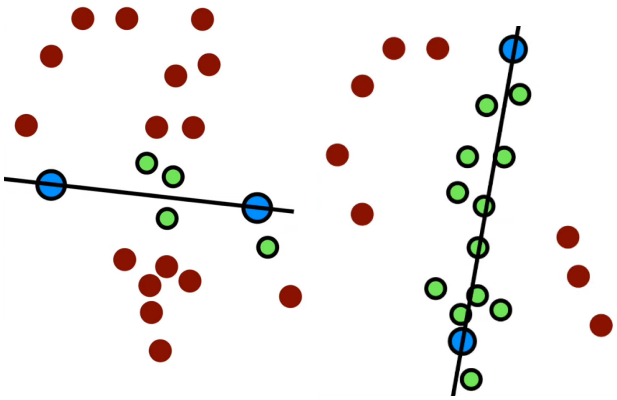

Figure 6.1: Result from two iterations of RANSAC ICP

These two above figures show the results from two different iterations of RANSAC ICP. The left figure contains 4 inlier point over 22 points, so the score of this iteration is 0.18. The right figure contains 12 points, so the score is 0.54.

6.1.1.2 Implementation

The following picture shows the result obtained with the RANSAC global registration for the box. The point clouds are down-sampled to have a voxel size of 1 cm.

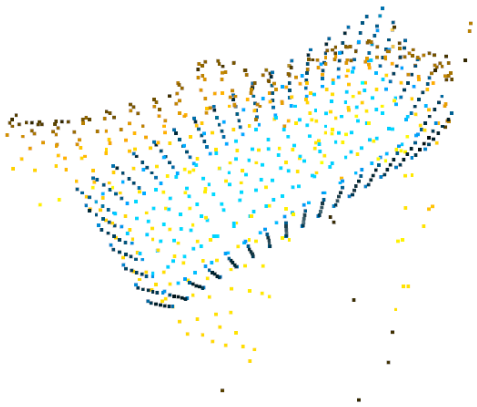

Figure 6.2: Result of the RANSAC global registration step for the box

6.1.2 Local registration

6.1.2.1 Point-to-point ICP

Point-to-point ICP iteratively computes a transformation between two point cloud. It works in two steps. The first one is data association step. The second step computes the transformation based on the data association. The few following figures show how ICP works. The first figure shows the two data set that is wanted to be aligned. The blue point cloud needs to be aligned with the red point cloud.

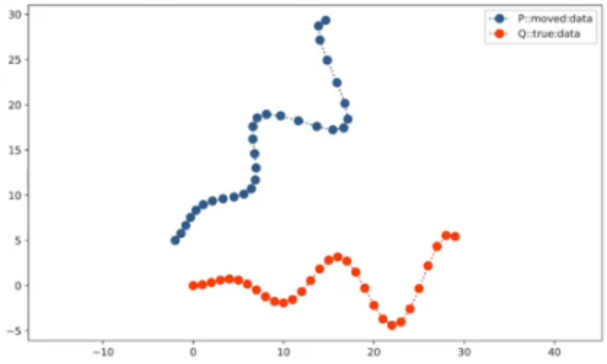

Figure 6.3: The two point cloud that need to be aligned

The first step is data association. This step uses a simple nearest neighbor approach. For every point in the blue point cloud, we look for the closest point the red point cloud. After this step, we retrieve the data association as in the following figure.

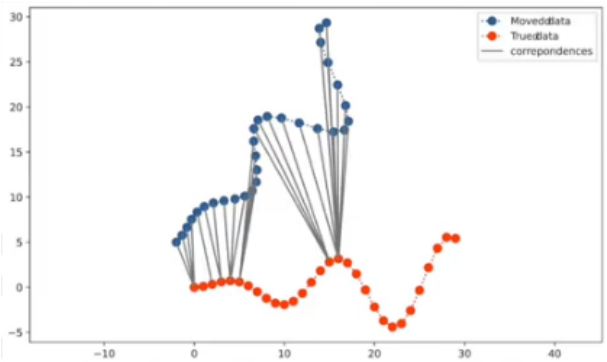

Figure 6.4: Data association step

The second step is the transformation step. This step takes the data association obtained in the first step. Based on this transformation, it tries to find the transformation that minimizes the distance between the point pairs. This is done in two steps.
Firstly, the center of mass of the blue point cloud is computed. Also, the center of mass of a subset of the red point cloud is computed. This subset contains the points which are paired with a point in the blue set. Then, the two point cloud are shifted so that the center of mass are aligned. The following figure shows the result of this step.

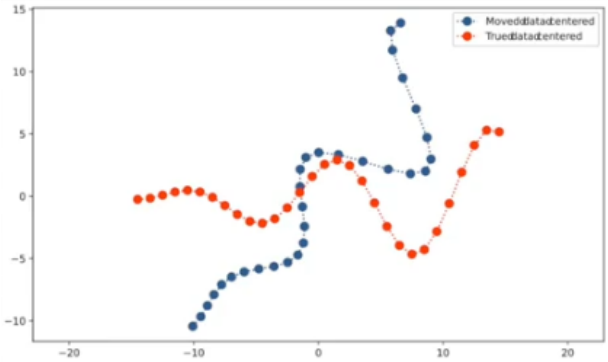

Figure 6.5: Center of mass alignment

Secondly, the best rotation between the two point cloud is computed. This is done using a Singular Value Decomposition (SVD) based approach to align the two set. The following figure shows the result of this step.

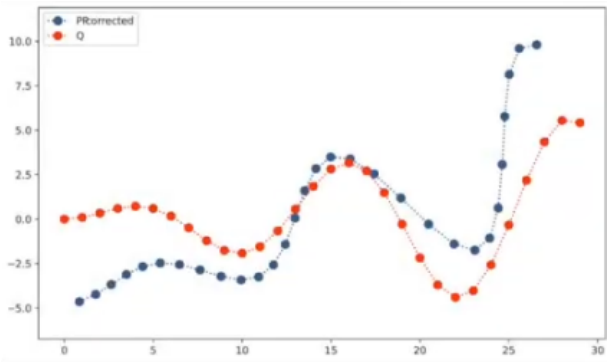

Figure 6.6: Result of the first iteration of point-to-point ICP

As we can see in this figure, the alignment between the two point cloud is not perfect. This is due to the data association step. To improve the result, the above steps are iterated until it converge to a satisfying transformation.

6.1.2.2 Implementation

After trying different ICP methods for the local refinement step, like point-to-point, point-to-plane or robust KERNEL, I have chosen to use point-to-point ICP. I firstly ruled out point-to-plane ICP, because it was giving good results for cubic objects, like milk cartons, but the results were not good for the other shapes of objects. Then I compared the results from point-to-point ICP and robust KERNEL, and point-to-point ICP was giving the best results. This stage refines the pose of the object given by the global refinement with robust KERNEL. The following pictures show the result of pose estimation step obtained for a box and a bottle. The blue point cloud is the exact point cloud of the object, while the yellow point cloud is the retrieved point cloud.

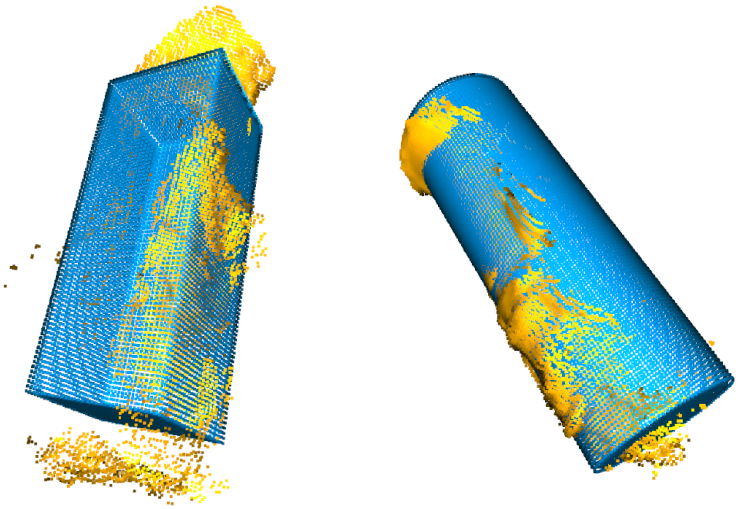

Figure 6.7: Result of the local refinement for a box and a cylinder

Even if the point cloud is not perfect, using robust ICP and point-to-point ICP enable to retrieve an approximation of the pose of the object. However, these two

algorithms have a flaw, it's that the algorithm can converge towards sub-optimal results like shown on the following two pictures.

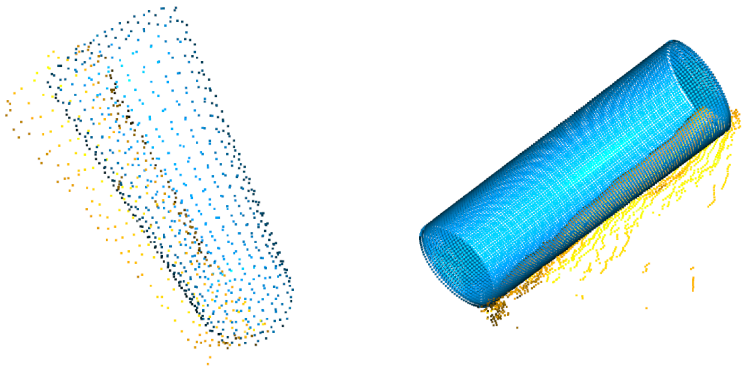

Figure 6.8: Failures of the ICP algorithm

Like shown in the above picture, this strategy can converge to sub-optimal results. Moreover, the pose of the object is not always accurate enough. In order to address the issue of sub-optimal results and to reduce the error in the pose of the object, I chose to use multi-stage point-to-point ICP with varying search distance. It would be possible to use a different approach to improve this stage of the pipeline. The first one would be to use a point cloud completion algorithm before the pose estimation step. Another approach would be to use coloured ICP method, but this method would require coloured point cloud.

6.2 Generation of the grasp poses in camera coordinate

At this stage of the pipeline, the 6D pose of the object in the camera coordinate is known and stored as a numpy array. Also, a set of grasp poses on the object coordinate is stored in a separate file. The grasp poses are then translated in the camera frame and stored in a matrix. The dimension of the matrix is N x 14 where N is the number of grasps. Each line of the matrix contains the data in the following format:

$$[r_{11},\ r_{12},\ r_{13},\ x,\ r_{21},\ r_{22},\ r_{23},\ y,\ r_{31},\ r_{32},\ r_{33},\ z,\ \mathit{width\ gripper},\ \mathit{success\ mark}]$$

Where the r_{ij} correspond to the rotation matrix between the camera frame and the grasp. The (x, y, z) coordinate correspond to the position vector of the center of the grasp in the camera frame.

7

Grasping using the real robot

Once a set of grasp poses for the object to grasp is generated, it's needed to compute the trajectory of the arm to go to the grasp pose. But before, the pose and the state of the robot needs to be known. Then, the motion of the arm is generated and executed. After, the grasping and manipulation tasks can be done.

7.1 Pose estimation of the robot using AprilTags

An AprilTag is a figure as shown in the following picture. It enables to be detected by a specific algorithm. The algorithm enables to return the four coordinates of the four pixels which are at the corners of the AprilTag.

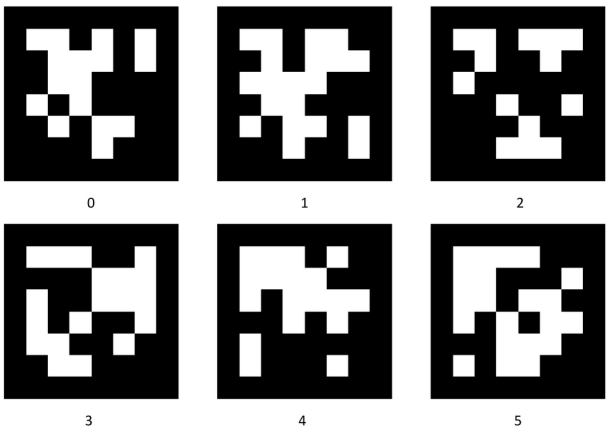

Figure 7.1: Some AprilTags with their ID

To know the pose of the robot, the coordinates of three corners of the AprilTag are used to know the origin and to compute the (x, y, z) axis of the robot. Then, the center of the robot is moved from the corner of the AprilTag to the center of the arm.
The AprilTag recognition algorithm returns the 2D coordinate of the four pixels which are at the four corners of the AprilTag. Then, the 3D coordinate of the corners is fetch in the point cloud. Only three corners are useful to compute the origin and the axis of the robot, the upper left, the upper right and the bottom right. The 3D pose of these three corners in regard to the camera center is given by the following vectors. Where ul stands for upper left, ur for upper right and br for

bottom right.

$$p_{ul} = \begin{bmatrix} x_{ul} \\ y_{ul} \\ z_{ul} \end{bmatrix}, p_{ur} = \begin{bmatrix} x_{ur} \\ y_{ur} \\ z_{ur} \end{bmatrix}, p_{br} = \begin{bmatrix} x_{br} \\ y_{br} \\ z_{br} \end{bmatrix}$$

Then the x and y axis are calculated using the following formulas:

$$\Delta_x = \sqrt{(x_{ur} - x_{br})^2 + (y_{ur} - y_{br})^2 + (z_{ur} - z_{br})^2}$$

$$x_1 = \frac{x_{ur} - x_{br}}{\Delta_x}, x_2 = \frac{y_{ur} - y_{br}}{\Delta_x}, x_3 = \frac{z_{ur} - z_{br}}{\Delta_x}$$

$$\Delta_y = \sqrt{(x_{bl} - x_{br})^2 + (y_{bl} - y_{br})^2 + (z_{bl} - z_{br})^2}$$

$$y_1 = \frac{x_{bl} - x_{br}}{\Delta_y}, y_2 = \frac{y_{bl} - y_{br}}{\Delta_y}, y_3 = \frac{z_{bl} - z_{br}}{\Delta_y}$$

After, the z axis is calculated using the vector product of the x and y axis:

$$z_1 = x_2 \times y_3 - x_3 \times y_2$$

$$z_2 = x_3 \times y_1 - x_1 \times y_3$$

$$z_3 = x_1 \times y_2 - x_2 \times y_1$$

Then, the robot axis in the camera frame are given by the three following vectors:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \; y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}, \; z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

These three vectors are then used to compute the rotation between the camera frame and the robot frame in a quaternion form. The four quaternions are given by the following formulas:

$$q_w = \frac{1}{2}\sqrt{1 + x_1 + y_2 + z_3}$$

$$q_x = \frac{1}{4} q_w (y_3 - z_2)$$

$$q_y = \frac{1}{4} q_w (z_1 - x_3)$$

$$q_z = \frac{1}{4} q_w (x_2 - y_1)$$

The quaternion is used instead of the rotation matrix because it has three main benefits. Firstly, it requires only four numbers to describe a 3D rotation, while 9 are needed with the rotation matrix. Moreover, quaternion requires less computation time and has a better numerical stability.
The above strategy has a drawback which is that the point cloud is noisy. Then, there is an error in the 3D pose of the corners of the AprilTag. This small error can lead to an error of a few centimeters at the gripper, which can lead to an unsuccessful grasp. Moreover, it's possible that the AprilTag is occluded by the arm of the robot. These two flaws led me to use several AprilTags. Indeed, I used three AprilTags,

like in the following two pictures. If one or two of the AprilTag are occluded, then the pose of the robot is calculated only using the coordinates of the corners of the viewable AprilTags. If two or three AprilTag are seen, then the corners used for the axis computation are farther away and then the relative error is reduced.

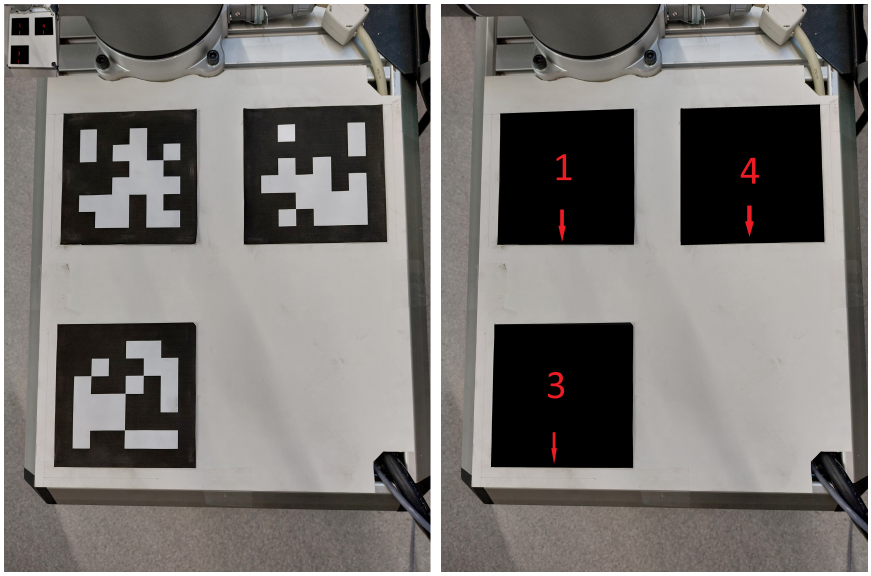

Figure 7.2: Layout of the AprilTags on the top of the robot

In the right above picture, the red number is the ID of the AprilTag and the red arrow is the orientation of the aprilTag.

7.2 TF tree

At this stage, the set of grasp poses for the object is given in the camera frame, also the pose of the robot is given in the camera frame. But the motion of the robot is planned and executed in the robot frame. Then, it's needed to manage the link between every frame.
Fortunately, ROS embed a package which enables to keep track of every frame efficiently. This package is TF and enables to keep track of multiple coordinate frames over time. It maintains the relationship between coordinate frames in a tree structure.
There is a TF structure from the base of the robot to all the parts of the arm. But it's required to know the link between the pose of the camera and the center of the robot. To do it, a TF link is created between the center of the camera and the robot. The following picture shows the links between every frame used in this project.

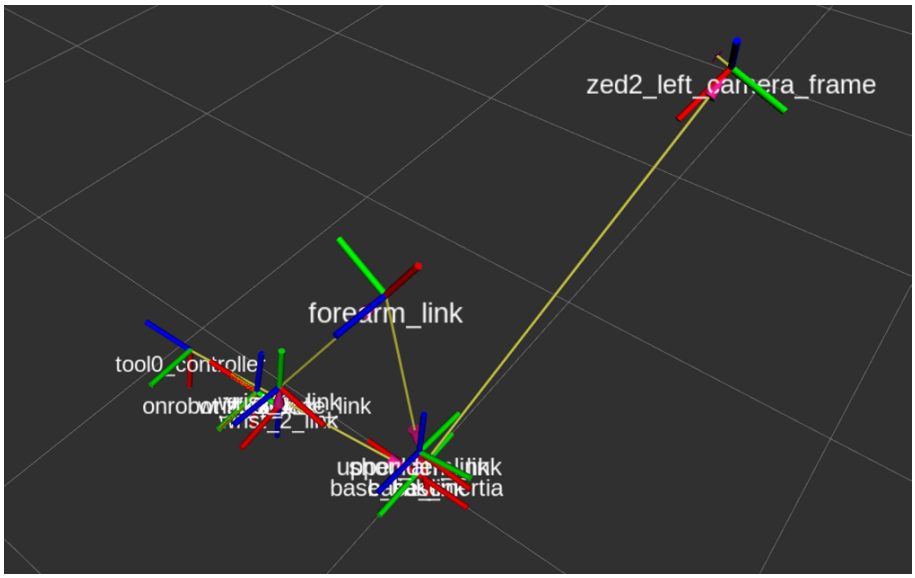

Figure 7.3: TF representation of the camera and the robot

In this picture, every frame is depicted by three poles. The red pole depicts the x axis, the green pole depicts the y axis and the blue pole depicts the z axis. The yellow arrow represents the link between the frames in the TF tree.
Then, to understand more clearly the frames depicted in the above picture, the following picture shows the same frames superimposed with the model of the robot arm.

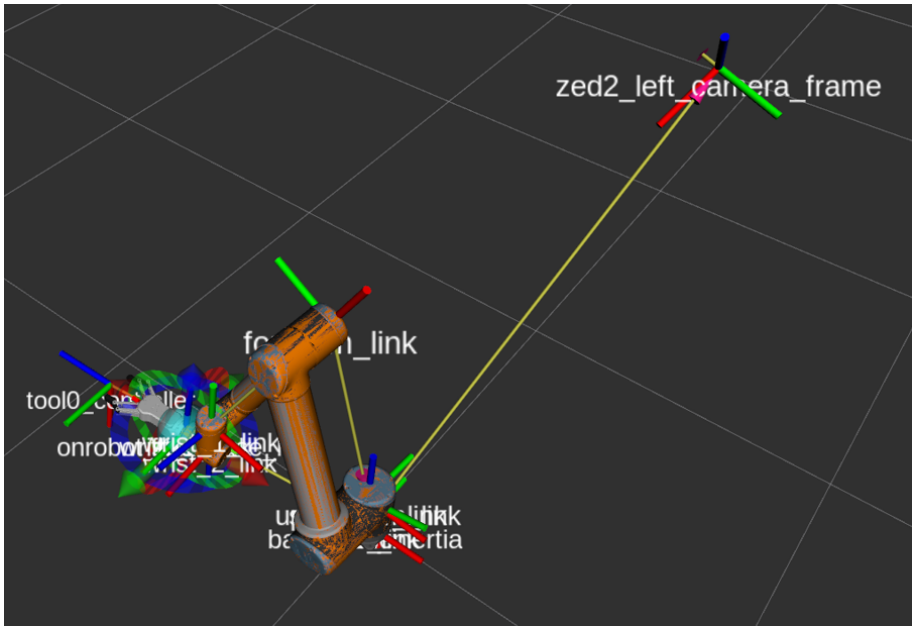

Figure 7.4: TF representation of the camera and the robot superimposed with the arm model

7.3 Choose of a grasp pose

In the above sections of this report was presented the pipeline who generates a set of grasp poses. But at this stage, where the pose of the robot and the pose of any part of the arm is known, it is needed to choose between all the grasp poses the most suitable grasp pose. Since GPD has been trained for a long time for all the objects used in this project, every grasp has a good grasp quality. Then, the score is not taken into account.
The chosen grasp pose is the grasp pose for whom the center of the gripper is the closest to the center of the grasp. To do it, the center of the gripper is collected using TF. Then, for every grasp pose, the distance between the center of the gripper and the grasp pose is computed and added to a vector. Finally, the grasp pose with the lowest distance is chosen.

7.4 Arm motion planning

The motion planing of the arm is done by finding the closest path from the gripper actual pose to the gripper goal pose. The actual pose is retrieved with TF. The goal pose is the grasp pose chosen in the above section. Since there are only two points in the trajectory, the gripper trajectory is a line. Then, when the gripper trajectory is known, inverse kinematics is used to compute the motion of all the joints of the arm at any time of the motion.

7.5 Grasping using the real robot

This work was experimented with four objects which are two milk cartons with different dimensions, a box of medicine and a non-transparent bottle. During the trials of grasping of these four objects, the success rate was promising.
The following picture shows the same experiment from two standpoints in RVIZ. In these pictures is shown the point cloud retrieved with the depth camera. Also, the pose of the center of the robot and the position vector from the center of the camera to the center of the robot is displayed. In these pictures is also displayed the selected grasp pose for the milk carton.

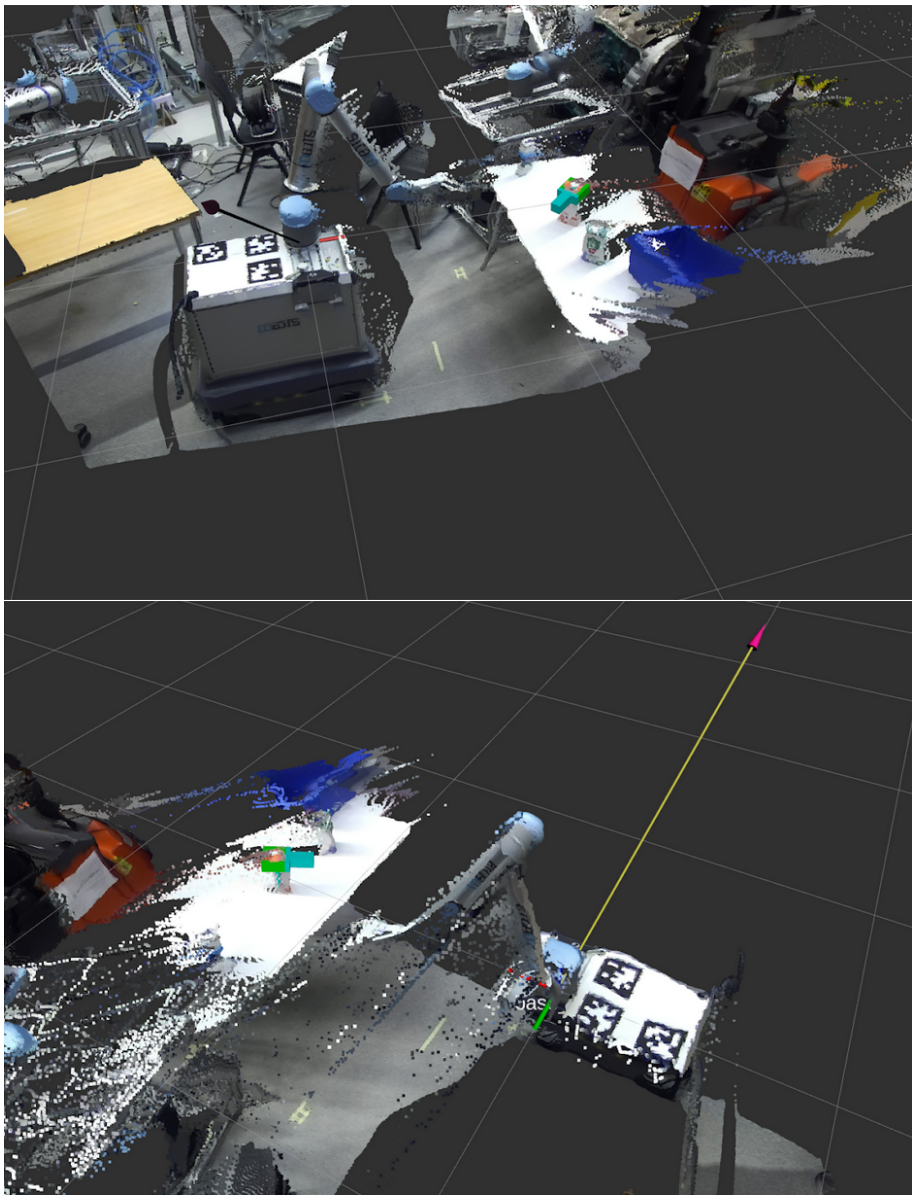

Figure 7.5: View of the grasp pose in RVIZ from two standpoints

We can see in these two pictures that the grasp pose matches the pose of the milk carton. The selected grasp pose is quite far from the center of mass of the bottle, this is due to the fact that the gripper is higher than the bottle at the beginning of the experiment. Since the selected grasp is the closest to the gripper, the selected grasp pose is the closest to the top of the bottle.
The following pictures show the pose of the gripper when he reached the grasp pose.

Figure 7.6: Gripper pose after reaching the grasp pose from two standpoints

We can see in these two pictures that the axis of the gripper is slightly out of the center of the milk bottle, this mismatch will reduce the smoothness of the grasp. This mismatch is due to the error in the pose of the robot and also to a small error in the pose estimation of the object.
The two following pictures show the bottle once the gripper closed its finger on the object.

Figure 7.7: Pose of the fingers when the object is grasped by the robot

As we can see in these two pictures, the mismatch between the axis of the gripper and the axis of the bottle boil down to a grasp which is not smooth. Indeed, the bottom of the bottle is not well aligned with the table after the grasp.
During the experiments, the success rate of the grasping was promising for the four objects. However, the mismatch between the axis of the gripper and the axis of the object sometime led to unsuccessful grasps.
Moreover, in the implementation the offset between the grasp pose and the center of the gripper was constant. Due to the architecture of the gripper, the required offset relies on the grasp width (figure 3.9). Then, the offset needs to be calculated in regard to the width of the grasp. The success rate would be improved by addressing this issue.
In order to measure the success rate of this method I did the following experiment. The experiment is shown in the following picture. The goal is to grasp the object which is on the table. The experiment was carried out with the four non-transparent objects presented before. The initial pose of the gripper was always the same, in front of the table about thirty centimeters away from the edge of the table. The pose and the orientation of the objects was modified for every test. For each object, 20 tests were performed.

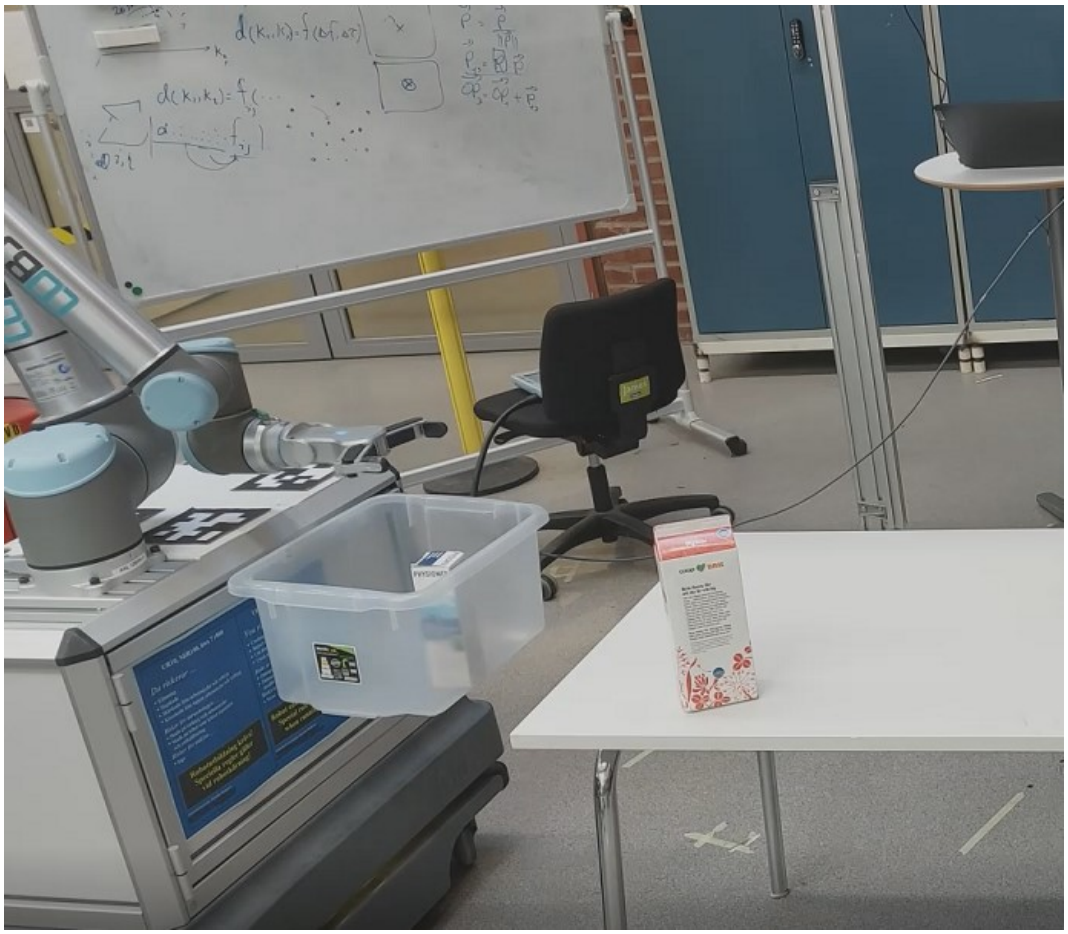

Figure 7.8: Setup of the experiment carried out to measure the success rate of the method presented in this report

The following table shows the results of the experiment.

	failure	non-smooth	smooth
Bottle	6	7	7
Medicine box	8	4	8
Red milk carton	5	9	6
Green milk carton	6	8	6
Average percentage	31.25 %	35 %	33.75 %

Table 7.1: Results of the grasp experiment for 4 objects of different shape

The failure column rely to the cases where the gripper was unable to pick up the object after it finished to close its fingers. The non-smooth column rely to successful grasps but the object was moving while the gripper was closing its fingers. The smooth column rely to the successful grasps where the object did not move when the gripper was closing its fingers.
From the table, we can conclude that the success rate of this method for the four tested objects is 68.75 %. Among the successful grasps, about half of the grasps

were smooth. The success rate could be improved by increasing the accuracy of the robot pose and improving the pose estimation stage.

8

Conclusion

The software produced for this project has been written in Python and it is available in Github via:
github.com/TristanChatelin/Grasp

8.1 Contributions of the pipeline

This pipeline was written with the aim to be used easily by other people. Indeed, it's written in Python. Moreover, it's possible to use another camera than the ZED2 camera, because the program only uses as input an image matrix and a point cloud matrix generated by the camera. It's also possible to change each part of the pipeline because they're each separated in different functions. In the future, the accuracy of the results of this pipeline could be improved by changing the used algorithms with newly published algorithm for the image segmentation, pose detection and grasp generation.
The algorithm to generate new objects point cloud and grasp poses enable to add a large amount of objects in the list of graspable objects. The YCB object set could be added.
The grasp poses for each object are computed only once and not during the use of the pipeline. Computing the grasp poses aside enable to compute high-quality grasp poses. Moreover, it removes the grasp pose computation during a robotic application and therefore speed up the pipeline.

8.2 Applications and possible improvements of the pipeline

This pipeline could be improved in three ways. Firstly, Detectron2 generate a class for each object, but it doesn't make the difference between two objects of the same class. For example, if there are two different mugs on the scene, it will only returns that there are two mugs. By training a new Detectron2 model with hand-labeled pictures of all the objects that needs to be detected, it will be possible to distinguish all the objects which are on the table. It could be done with the objects of the YCB dataset. Then, the exact identity of an object could be used to fetch the right point cloud and grasp poses in the database. If a model is trained with 100 hand-labeled images of each object, Detectron2 gives fairly good results. The quality can be improved by using more pictures, but the quality increase logarithmically with the

number of hand-labeled images.
Secondly, the computation time of the pipeline can be reduced by using another segmentation strategy. A strategy could be to firstly run an image detector algorithm and then to run an image segmentation algorithm only in the areas of interest. The quality of the input image in Detectron2 can also be decreased to reduce the computation time.
Thirdly, the pipeline could be improved by adding a point cloud completion to improve the accuracy of the pose detection.

8.3 Insertion of the pipeline in the robotic application

This pipeline returns a set of grasp poses for various known objects. This work can be continued in two ways.
The first one is about mobile manipulation. The aim would be to manage the motion of the arm and the mobile base together to grasp more efficiently an object. The final purpose of this work is to reduce the execution time and the energy consumption to grasp an object with a mobile manipulator by making a trade-off between the motion of the arm and the motion of the mobile base. Since the mobile base consume more energy than the arm during motion, in order to reach a grasp pose, it's more appropriate to move the arm instead of the base. In order to increase the reachability area of the arm, the mobile base needs to be moved accordingly.
The second work is about Task And Motion Planing (TAMP). The idea would be to improve this pipeline to return a set of grasps for all the objects which are on the environment. Then TAMP tasks would be done using all the generated grasp poses for all the objects. The aim would be to collect all the objects which are on a table efficiently and to use the mobile base to move towards the objects which are too far from the robot.

Bibliography

[1] Andreas ten Pas et al. "Grasp Pose Detection in Point Clouds". In: *The International Journal of Robotics Research* 36 (2017), pp. 1455–1473.

[2] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. "6-DOF GraspNet: Variational Grasp Generation for Object Manipulation". In: *CoRR* abs/1905.10520 (2019). arXiv: 1905.10520. URL: http://arxiv.org/abs/1905.10520.

[3] Jeffrey Mahler et al. "Dex-Net 1.0: A cloud-based network of 3D objects for robust grasp planning using a Multi-Armed Bandit model with correlated rewards". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1957–1964. DOI: 10.1109/ICRA.2016.7487342.

[4] Guoguang Du, Kai Wang, and Shiguo Lian. "Vision-based Robotic Grasping from Object Localization, Pose Estimation, Grasp Detection to Motion Planning: A Review". In: *CoRR* abs/1905.06658 (2019). arXiv: 1905.06658. URL: http://arxiv.org/abs/1905.06658.

[5] Shantanu Thakar et al. "Towards Time-Optimal Trajectory Planning for Pick-and-Transport Operation with a Mobile Manipulator". In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. 2018, pp. 981–987. DOI: 10.1109/COASE.2018.8560446.

[6] V. Sathiya and M. Chinnadurai. "Evolutionary Algorithms-Based Multi-Objective Optimal Mobile Robot Trajectory Planning". In: *Robotica* 37.8 (2019), pp. 1363–1382. DOI: 10.1017/S026357471800156X.

[7] Arjun Menon, Benjamin Cohen, and Maxim Likhachev. "Motion planning for smooth pickup of moving objects". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 453–460. DOI: 10.1109/ICRA.2014.6906895.

[8] Cong Wang et al. "Multi-Task Reinforcement Learning based Mobile Manipulation Control for Dynamic Object Tracking and Grasping". In: *CoRR* abs/2006.04271 (2020). arXiv: 2006.04271. URL: https://arxiv.org/abs/2006.04271.

[9] Balint Varga et al. "Model Predictive Control and Trajectory Optimization of Large Vehicle-Manipulators". In: *2019 IEEE International Conference on Mechatronics (ICM)*. Vol. 1. 2019, pp. 60–66. DOI: 10.1109/ICMECH.2019.8722886.

[10] Guillaume Picard et al. "Multi-Trajectory Approach for a Generic Coordination Paradigm of Wheeled Mobile Manipulators". In: *IEEE Robotics*

and Automation Letters 7.2 (2022), pp. 2329–2336. DOI: 10.1109/LRA.2022.3143894.

[11] Adam Heins, Michael Jakob, and Angela P. Schoellig. "Mobile Manipulation in Unknown Environments with Differential Inverse Kinematics Control". In: *2021 18th Conference on Robots and Vision (CRV)*. 2021, pp. 64–71. DOI: 10.1109/CRV52889.2021.00017.

[12] *ZED2 Camera and SDK overview*. Stereolabs. 2019.

[13] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY