Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Irja Gravdahl

Grasp selection in bin picking tasks for robotic manipulator arm with end-effector geometric constraints

Master's thesis in Cybernetics and Robotics
Supervisor: Kristin Y. Pettersen
July 2019

NTNU
Norwegian University of Science and Technology

Irja Gravdahl

Grasp selection in bin picking tasks for robotic manipulator arm with end-effector geometric constraints

Master's thesis in Cybernetics and Robotics
Supervisor: Kristin Y. Pettersen
July 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

Robotic bin-picking is the problem of grasping objects placed randomly in a bin and moving the objects to a desired location one by one. The problem often arises in industrial settings where items come out in bulk and need to be processed individually. Bin-picking exists as of today at the intersection between industrial application and academic research; task-specific systems exists in the industry, and the research community is executing extensive research to find a more general solution to this problem. This thesis aims to support this research trend by investigating grasp selection in a robotic bin-picking system where end-effector geometric constraints are present.

The problem at hand can be summarized as: "given a valid grasp on an object, how easy or difficult is it for the robot to reach said grasp while considering all its constraints?" This work investigates a step-wise answer to this question. To motivate the solution, a background in robotics, grasping and motion planning is given. Furthermore, current research within combined grasp- and motion planning, with a focus on workspace representation of accessible regions is presented.

Working with the robot set-up in simulation, the first step towards answering this question, was to map an appropriate part of the workspace of the robot. This workspace mapping was undertaken in order to place the bin optimally in terms of how much of the bin area can be reached by the robot. The workspace map is created by organizing the coverage of inverse kinematic solutions across it, along with investigating if a path can be found to the different regions of the space. When placing the bin optimally in terms of inverse kinematic solution- and path existence, the threshold on how easy a grasp could be reached in the bin was lowered.

In this new bin area (optimized with respect to reachability and path existence), new maps were constructed. Since the objective of the work was to identify "easy to reach"-grasps, it was of utmost importance to also map this new bin space in terms of the two metrics, 1) reachability and 2) path existence. By testing different motion planners with several optimization objectives in this area, even more information about the bin space becomes apparent. The metrics 3) path length, 4) planning time needed, and 5) the execution time utilized for different paths to cover the bin space provide more information. Using these aforementioned metrics, the most "easy-to-reach" regions in the bin could be identified. The *path reachability map* of the bin includes these metrics, implicitly including inverse kinematics, path planning and collision-checking in the map.

The work culminates in the use of the path reachability map in an algorithm, where robot abilities are taken into account when choosing a grasp to attempt. A cost function evaluates if the grasp is valid in terms of inverse kinematics and path existence, whilst also taking into account the efficiency of the grasp in terms of a previously identified path. Paths that are short, take little time to plan and execute are preferred over less efficient grasps, enabling faster, more "robot friendly" picking. Based on these results, faster and more efficient picking can be achieved by including robot capabilities in the grasp selection process.

The work undertaken during this period and the results mentioned also culminated in a paper entitled "Robotic Bin-picking under Geometric End-Effector Constraints: Bin Placement and Grasp Selection" submitted to the International Conference on Control, Mechatronics and Automation 2019 (Gravdahl et al., 2019), which is presented in its entirety in the appendix.

Preface

The following pages contains the master thesis "Grasp selection in bin picking tasks for robotic manipulator arm with end-effector geometric constraints", researched and written from January 2019 to June 2019. The work is based on the specialization project done within the same thematic from August 2018 to December 2018.

The work outlined in this thesis has been done in cooperation with SINTEF Digital Trondheim. From SINTEF Digital Trondheim, I have been provided the Robot Operating System (ROS) workspace structure along with all the files and packages needed to build said workspace, to be able to get the system up and running. I would like to thank Bjørn-Olav Holtung Eriksen at NTNU for his help one afternoon when the ROS error messages became too many, when trying to befriend the workspace.

This ROS workspace lays the foundations for the scripts written by the author, and the results obtained and presented here. From NTNU, I have been provided a complete work station with a computer which has been utilized in the last year.

At SINTEF Digital Trondheim, senior researcher Esten Ingar Grøtli (co-supervisor) and MSc Katrine Seel have provided help and guidance during bi-weekly meetings. The guidance sessions have consisted of presenting the work so far and setting goals for the next two weeks. Both of them have been very available per e-mail and have been invaluable to the work process. Their input and ideas have been a catalyst for this work and are greatly appreciated.

All photographs of the SINTEF set-up, along with the grasp set data from the neural network (which is explained and used in Chapter 8) were provided to me by Katrine. Esten also helped me get started with the programming at the beginning of this project in August 2018, when I did not know where to begin with the task I was given. The physical robot set-up has also been available to me during the work. I would like to thank the both of them immensely for their interest, guidance and help.

I would also like to thank my supervisor Kristin Y. Pettersen for taking on an additional candidate when I came to her with this task, along with providing me with the thesis template, reminding me of important deadlines, and being available when I needed help.

On a more personal note, I would like to thank my parents, for helping me reach my goals and as always, being supportive of the things I do. I would also like to thank my boyfriend for his patience during the last year, and being there for me as he always is.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The following thesis titled "Grasp selection in bin picking tasks for robotic manipulator arm with end-effector geometric constraints", details and explains the work completed during the spring of 2019 concluding a 2-year masters degree at the Norwegian University of Science and Technology. The work presented here builds on the results obtained from the specialization project within the same thematic completed during the fall of 2018. This introductory chapter will begin with motivating the problem of bin-picking and outlining the problem formulation with regards to this theme. Furthermore, the assumptions shaping the work will be presented, before the contributions of the work are accounted for. This chapter concludes with an outline of the remaining chapters, describing briefly the content of this thesis.

1.1 Motivation

[†]To motivate the thematic of bin-picking, and why this is an interesting problem from both a scientific and industrial point of view, a complementary example is presented.

Imagine that you are tasked with moving 10 apples from one crate to another. This seems a simple enough problem to solve. You grab one apple, with either hand, move the apple to its desired location and you put it down or drop it in the second crate. And then you repeat the process nine more times.

In the completion of this task you have used your eyes as a 3D sensor, scanned the environment, estimated the distance to the apples, registered its position and orientation with a "reference apple" in mind and decided which apple it is best to grab first based on how they all are positioned in the crate. You have used previous experience and knowledge to estimate its consistency, weight and centre of mass. This enables you to know where to grab the apple for an optimal grip, how hard you should grab it, how tight you should hold it when you move it and when you can drop it into the bin without damaging it. Using your fingers and palm as a gripper with force feedback to hold the apple enables you to adjust your grip as to not damage it as well, as you can feel the force you are exerting on the apple.

When you scanned your environment you also knew how far away the second crate was and what the optimal path you should take to unload the apple was. The process of moving the apples was not difficult, but you used your own sensors and actuators in real-time to achieve the goal.

[†]The example in this section is a reiteration of the same example, also presented in the specialization project (Gravdahl, 2018)

Perhaps you also tried to optimize the process after having moved a few of the apples? Maybe you moved because you thought you could get the job done faster if you stood closer to the second crate, but then moved back because this made it harder to grab an apple from the first crate? All these aspects and processes we take for granted to solve this problem must be transferred to a robot for it to manage the same task, but how can we make it see its surroundings and let it know or teach it how it should grab the apple without damaging it? And how can we make a robot aware of whether it actually can grab the apple or not when it has figured out how to grasp it?

Robotic bin-picking is a classic robot problem, where the objective is to achieve a pick-and-place routine in a randomized environment. The robot is presented with a bin, containing some object or parts and its task is to pick an item, one at the time, and place them safely in the next bin. In an industrial setting this could for example be on to the next conveyor belt or a new work station. This is a tedious task for humans where we perhaps do not reflect on the complexity of it, and being able to employ robots in our place would allow an increase in efficiency and predictable timing of for example an assembly process.

1.2 Problem formulation

1.2.1 Bin-picking set-up

Figure 1.1: Photo of the robot cell at SINTEF Digital Trondheim. Notice especially the camera housing with the L-shaped gripper attached. Photo courtesy of Katrine Seel, MSc, SINTEF Digital.

The title of this thesis is "Grasp selection in bin-picking tasks for robotic manipulator arm with end-effector geometric constraints", and in Figure 1.1 the robot cell can be seen. The cell is the space where the robot operates, also containing the different items in its immediate surroundings; for example the bin containing the objects that are to be picked and the table the bin is placed upon. The robot is a UR5 (Universal Robots), and at its final link, the end-effector, a big box can be seen. This camera housing protects a Zivid 3D camera that is used to scan the bin seen on the table with the parts in it, and is what constitutes the "end-effector geometric constraints". This is what is called an eye-in-hand contraption, where the sensor is placed in "the hand" of the robot. The L-shaped object on the side of this camera housing ends in a suction cup used for gripping the objects. The robot base is placed on a pedestal which can be seen as the black cylinder behind the camera housing.

In this system, the 3D sensor obtains depth images, point clouds, of the distribution of cylindrical objects in the bin, which can be seen in Figure 1.2. The sensor obtains the position and orientation of the objects from the point cloud. Which object to grasp is decided by a dual-resolution

convolutional neural network (Dyrstad et al., 2018) that takes the point cloud of the current distribution of objects in the bin as input. The motivation behind using a neural network to determine the best way to grasp an object will be accounted for at a later time, as well as how the grasps are presented to the robot. The neural network supplies a list of grasps in prioritized order, where the first grasp outputted is attempted by the robot. If the path to the grasp or the grasping itself fails, the next grasp in the list is chosen. If the grasp and path to it is successful, a new point cloud is captured by the 3D sensor and the process begins anew. The flow of this part of the system can be seen in Figure 1.3.

Figure 1.2: Cylindrical objects to be picked from plastic bin covered with felt. Photo courtesy of Katrine Seel, MSc, SINTEF Digital

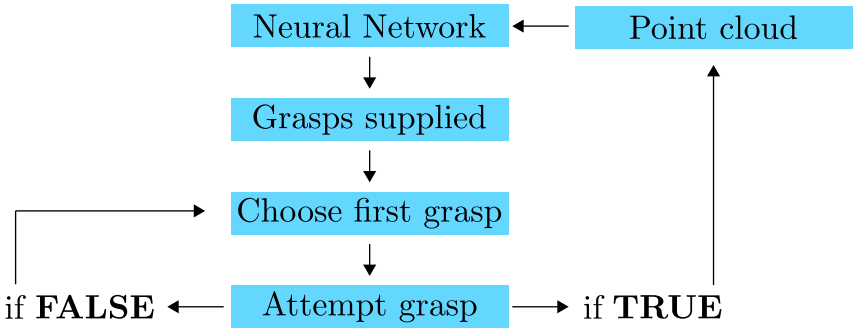


Figure 1.3: Illustration of the current work flow in the bin-picking system at SINTEF Digital Trondheim.

1.2.2 Assignment

Bin picking is the problem of grasping objects randomly placed in a bin. This is a problem that often occurs in industrial settings where objects come out of a production line packaged in bulk, without isolating individual objects, and where the objects are transported to a second production line that subsequently must isolate and process these objects individually. Information from a 3D-sensor is used to compute many possible grasps based on how the objects are placed in the bin. When the 3D-sensor is attached to the robotic arm performing the grasps this imposes additional constraints on how the robotic manipulator arm can move while avoiding self-collisions and collisions with the bin or other parts of the environment. The overall goal of this assignment is to find a way to judge which grasps are favourable for grasping with the robotic manipulator arm.

Assignment:

- Make a literature review of state-of-the-art methods relevant to achieve the described goal
- Design one or more metrics suitable to evaluate different grasps, which then can be used to judge the performance of different methods. The picking should preferably be carried

out as fast as possible, but it might be useful to consider other metrics, for instance related to safety.
- Set up a simulation environment suited for testing. The simulation should be a representation of the physical setup that SINTEF and NTNU already using for bin picking.
- Implement at least one of these methods, and test it in the simulated environment

1.3 Assumptions

To limit the scope of this thesis, the following assumptions are made, and assumed to hold true for the duration of the work period:

- The work accomplished within this thesis is all based upon the bin-picking set-up depicted in Figure 1.1. This includes the specific gripper, robot and remaining design. Other robot set-ups or bin-picking solutions will not be considered, and the results concluded upon are valid for this set-up only, as the work is only tested on this system.
- The grasps supplied to the robot for picking are supplied by a neural network. These grasps are considered optimal in terms of grasp quality, meaning that it is assumed that all the grasps provided are valid. The functionality of the neural network and how the grasps are produced is not part of this work. The aspects of the grasp selection process that is decisive for the completion of this work will be explained at the appropriate place in the text. The functionality of the network and image processing which will not be discussed further.
- The pose estimation by the Zivid 3D camera, and the calibration of this sensor is assumed to be correct and in order. The functionality of the sensor and its associated calibration routines is outside the scope of this thesis.
- The current placement of the bin as seen in Figure 1.1 is based upon the optimal range of the sensor in terms of obtaining quality images. During the process of this work, this optimal range is not considered.

1.4 Contributions

This thesis has been conducted in cooperation with SINTEF Digital Trondheim, under the supervision of senior researcher Esten Ingar Grøtli and MSc Katrine Seel. The Robot Operating System (ROS) workspace forms the basis upon which this work is built, and has been provided in its entirety by SINTEF Digital. Contributions will first be given in order of chapter in a list, and then in a more comprehensive manner.

Contributions include, in order of chapter:

- An overview of previous research conducted on the combination of grasp planning and motion planning found in Chapter 6, based on the literature review conducted on the themes separately in Chapter 4 and 5 respectively.
- A method for mapping the workspace of a UR5 with end-effector geometric constraints is presented in Chapter 7. Also in this chapter, how to use the workspace map to identify optimal bin placement in terms of the map characteristics is presented. Furthermore, for this optimal placement, a method for also mapping this space is accounted for.
- An algorithm taking into account the robot abilities and its *path reachability* is presented, implemented and tested in Chapter 8, concluding the work.

- This thesis has culminated in an article available appendix A titled "Robotic Bin-picking under Geometric End-Effector Constraints: Bin Placement and Grasp Selection" submitted to the 7th International Conference on Control, Mechatronics and Automation (Gravdahl et al., 2019).

In this work, it will be shown that mapping the workspace of the robot manipulator arm, positioning the bin based on this workspace mapping, and thoroughly re-mapping this space with potential grasps offline can increase picking success. The picking success is increased by placing the bin in a more accessible region of the workspace, as well as prioritizing grasps corresponding to short path lengths and low time consumption. Using this mapping data in conjunction with the output from the neural network to connect these subsystems, it is possible to re-arrange the preferred output from the neural network in terms of the robots ability to pick the objects, prioritizing grasps reachable by the robot. Mapping in this context refers to creating a map of the robot abilities. Due to the additional constraints on the system imposed by the pedestal, the camera housing, and potential collisions with the bin, in addition to an inverse kinematic (IK) solution existing, a collision-free motion plan must exist for a pose to be reachable.

The contributions of this work include utilizing the combined result of IK solutions and motion-plan existence in the workspace to place the bin optimally. Optimally in this context refers to the region of the workspace with the highest concentration of IK solutions and motion-plans. Furthermore, in this optimal region, different planners and optimization objectives from the Open Motion Planning Library (OMPL) (Şucan et al., 2012) was investigated in terms of several metrics; path existence, path length, planning time and execution time. Moreover, a method for introducing the aforementioned metrics of the robot into the grasp selection process, without the need for explicitly querying an IK solver, path-planner or collision-checker is described, implemented and tested upon the simulated environment.

1.5 Outline

The report is organized into chapters with the following main content:

- **Chapter 2**: A background in bin-picking is introduced, with a discussion regarding its challenges and motivating the importance of doing research within this field.
- **Chapter 3**: A background in robotics is presented, including the definitions of the robot workspace, the configuration space and robot kinematics. In this chapter, the structure of the Robot Operating System (ROS) is introduced along with the most important libraries and plug-ins utilized in this work.
- **Chapter 4**: A brief introduction to robotic grasping and associated definitions is given. The grasps supplied within the system at hand is also detailed here.
- **Chapter 5**: Motion planning is defined and introduced from an algorithmic perspective, along with the functionality of motion planning algorithms used in this thesis.
- **Chapter 6**: The definition of *path reachability* is given. A literature overview of work conducted on combining motion planning and grasping in an integrated system is presented, along with reflections on impacts these implementations might have on the set-up at hand.
- **Chapter 7**: This chapter details the method and some of the results obtained during the work. This includes:
  → A reiteration of work done on grasp selection previously on the SINTEF set-up.
  → Workspace mapping of the UR5 based on the metric path reachability.
  → Bin placement based upon this workspace mapping and metric.

  - → Creation of path reachability maps in the new region the bin is placed.
  - → A procedure for testing of different planners and optimization objectives in terms of different path metrics.
- **Chapter 8**: This chapter details the main results and contains a case study detailing the use of the path reachability algorithm and the results obtained with its use.
- **Chapter 9**: Concluding remarks are made, and suggestions for future work is accounted for.
- **Appendices**: In the appendix, the article titled "Robotic Bin-picking under Geometric End-Effector Constraints: Bin Placement and Grasp Selection" submitted to the 7th International Conference on Control, Mechatronics and Automation is presented in its entirety.

Chapter 2

Bin-picking background

Randomized bin-picking is the problem of picking objects from one location where they are distributed randomly and place them in another location, often in a more ordered fashion. The problem often occurs in industrial settings. Several workstations may exist for a production line where individual parts are clustered together and must be moved to a second stage in the process for further work. The objective is to pick and place each individual part from a bin of many. The following sections will detail the problem of randomized bin-picking, as well as discuss the motivation behind looking for a solution to the problem, and the impact a solution might have on the industry.

2.1 Bin-picking

A bin-picking system is a mixture of technologies working together towards a main goal. The process of bin-picking has been nicely summarized by Kraft et al. (2014) into these four steps:

1. Use a sensor system to detect an object in the bin and its position and orientation (pose).
2. Find and select an appropriate way to grasp the object.
3. Execute the appropriate robot motion to grasp the object.
4. If the grasp was successful, move the object to a desired location.

The task at hand; detect object, plan a grasp, plan and execute a robot motion, and move the object to a new location. As with the apple example in the previous chapter, the process itself is clear in its goal, and it is a comprehensible problem. However, when investigating each of these well defined steps in the process, a plethora of sub-steps become apparent. With the theme of this work in mind, step 3 in the list by Kraft et al. (2014) above resembles the objective the most. The goal is not only to execute the appropriate robot motion to reach a grasp, but focus on grasps that are easy to reach for the robot. This step is accessible in understanding, but several smaller steps must be considered.

A bin-picking system is not a "one system fits all" application. Which type of robot one wishes to use, and its reach as well as its geometry and design will constrain a bin-picking system to operate within the constraints enforced by the robot (Bogue, 2014). Also considering the item to be picked is of utmost importance. The same system can not necessarily be used to pick both needles and wooden logs for example, even though the software side of the system can be the same in design. If these two items are to be picked, a robot with a larger payload limit will be necessary to pick the logs, whilst for picking the needles a lightweight robot with perhaps

a magnet gripper of sorts will suffice. The type of gripper will also need to be customized to the system at hand. The objects to be picked also dictate the kind of vision system that is to be used, so a choice of sensor is also something to consider. This goes for both the type of objects as well as how randomly they are placed in the bin, which can range from organized to completely random (Bogue, 2014). Palletizing for example, can be thought of as an "ordered" bin-picking process, and this is widely implemented in the industry.

A bin-picking solution is often tailored specifically to one type of object, vision system and robotic set-up. Marvel et al. (2012) have undertaken a technology readiness study of bin-picking and mention metrics that a bin-picking system may be evaluated on the basis of. When a solution has been obtained, three principal performance metrics can be used to evaluate the solution; speed, efficiency and accuracy (Marvel et al., 2012).

The speed of the solution refers to both the time needed to pick an object, called picking time, as well as the rate of the picking, the bandwidth. A slow solution to the bin-picking problem, even though it is a solution, might not fulfill the demands of the industrial parties interested in using such a system. When automating a process, one is often interested in increasing the bandwidth, to get more work done over a lesser time period. The next performance metric mentioned is the efficiency of the solution. This refers to the amount of time used to acquire an object from the bin, compared to the amount of time spent searching and processing the information needed to pick it, the grasping quality and how efficiently the robot moves in and out of the bin. If a system is very fast in the robotic picking, it has high speed, but spends twice as long planning a full picking operation, and the robot often collides with the bin, the system is not very effective. In terms of using a system in the industry, reliability is key, and if the system is not very efficient, problems in predicting how much can be manufactured and the output volume of products will prove difficult to estimate. The final metric is the accuracy of the system, and refers to how small the measurement error is in both object recognition and pose estimation. Using a reliable, efficient system will be decisive if it is to be used in the industry (Marvel et al., 2012).

When reflecting over these performance metrics, the efficiency of the system entails the most details where the robot is at the centre. An educated guess as to what is the limiting factor within this metric leads to the robot motion generation step. A robot motion and its execution time will be much longer than the time needed to plan for a grasp relatively speaking. From this it is possible to deduce that more efficient picking can be obtained with more optimal robot motions, and prioritizing grasps that are accessible to the robot, also in terms of collisions. It is at least, an area worth keeping in mind.

Along with the three performance metrics, three primary challenge domains are also presented by Marvel et al. (2012), that highlight well the complicated structure of bin-picking systems from an industry perspective. This perspective is particularly useful, since it is in this domain the applications are most likely to be used. The challenge domains are sensing, hardware issues and solution integration (Marvel et al., 2012).

The sensing domain includes algorithm development with respect to the sensors involved and the optimization of the work cell; the space where the robot operates including any obstructions. A common input to the 3D sensor, which is to identify the parts in the bin, is a CAD-model of the objects. If the sensor for example is ill-calibrated, one runs the risk of not being able to detect any objects. And if no object is detected, no picking can be carried out. Another aspect of the sensing domain is lighting conditions in the work cell. Light sources may provide challenges with reflectivity, as is a challenge in the system at hand, and shadows "confusing" the sensor which may lead to a misinterpretation of the image data. Product variations due to

previous steps in the manufacturing process or damages or inconsistencies with the bin may also lead to misinterpretation and false detection due to a mismatch between the CAD model and the object detected (Marvel et al., 2012). The challenges within this domain are many, but the field of robot vision has experienced a rapid development in later years (Bogue, 2014).

The second challenge domain is hardware. This includes the reach of the robot and how it handles the objects it is to acquire. This also includes the robot gripper. The positioning of the robot in the cell, perhaps also constrained by other machines or robots, may limit its ability to reach certain grasps or points in its workspace (Marvel et al., 2012). This has been an issue with the set-up at hand, where for example the robot has problems when interacting close to the pedestal it is stood upon.

The third problem domain, which Marvel et al. (2012) claim to be the most difficult one, is solution integration. This is a more general problem domain as it is not specifically related to the robot set-up or specific other hardware or software solutions. The cost of a bin-picking system includes of course the investment in the system, but also the time it takes to integrate the system in existing production lines, how much time it takes to get the system online, and how versatile it is in its use. If the system is too task-specific, it will only function optimally on a few tasks, but if it is too general it might not satisfy the needs of the buyer in terms of efficiency and speed. This domain also includes bridging the gap between those who are to integrate a system and the developer of it when it comes to knowledge about the system as a whole, and that both parties must have realistic expectations considering what one can expect from a robotic bin-picking system (Marvel et al., 2012).

2.1.1 Motivation and importance

Robotic bin-picking serves its purpose by fulfilling the three D's of robotics; dirty, dull and dangerous (Vasilash, 2017). Moving objects from place to place is a dull and repetitive task, which by design is an optimal task for a robot. Palletizing for example, can be seen as a subset of bin-picking where the location of the pallets are known and organized. Their goal position is also orderly, and the use of robots when it comes to palletizing is widespread in the industry today, particularly due to the repetitive nature of the task. Bin-picking can also handle the dirty and dangerous. If a solution to the bin-picking problem can be found generally, it can be applied to near everything that needs picking. This will then include industrial processes that are contaminated. If robots can be used in place of humans when it comes to high-polluting materials like radioactive waste, this would be an additional advantage. Being able to bin-pick other objects like razor blades or syringes would also limit human exposure to potentially dangerous objects. The motivations behind solving randomized bin-picking are many, and a few are accounted for here.

With a more general bin-picking solution, the availability of reasonable solutions would increase and become available to all industry segments, independent of size. If a solution is found, the solution integration problem domain as accounted for by Marvel et al. (2012) might also shrink in size due to the demand for more "easy to integrate" solutions. The collective knowledge of bin-picking systems would also increase, and not be limited to those who produce a system, has a system or those who perform research within the field. This again would contribute to overcoming the challenges regarding both solution integration and hardware issues, since an increased availability often sparks interest and knowledge.

If a solution should be found, this would decrease the need for large expensive specific machinery like tumblers that "shake" individual components over a distance such that that align for further

processing. A solution to the bin-picking problem would lead to them outperforming tumblers and other expensive machinery in terms of cost (Bogue, 2014). A solution would also entail an increased flexibility in system set-ups when specific machinery is not applicable anymore, and widespread use of the bin-picking system will again breed new ideas and solutions.

If more industry segments have available to them bin-picking systems, this would provide a decreasing cost in terms of work hours should the picking be done by hand. In addition, increased efficiency, predictability and reliability in production volume would be experienced, which often leads to increased speed and larger production volumes (Bogue, 2014).

Chapter 3

Robotics

[†]The definition of a robot is *a programmable, multifunctional manipulator designed to move material, parts, tools or specialized devises through variable programmed motions for the performance of a variety of tasks.* From this definition we can extract the concept of reprogrammability, meaning that we may again and again utilize a robot in different tasks, and this gives a robot its utility and adaptability. Some of the most commonly stated advantages of the introduction of robots is decreased labour costs, an increased flexibility with regards to its reprogrammability and the creation of more humane working environments where robots are able to perform dangerous, repetitive and dull operations (Spong et al., 2006).

3.1 Rigid bodies and motion

To be able to comment on how easy or difficult it is for a robot to reach a certain grasp, a background in robotics is necessary. Though this thesis is regarding grasp selection, subjects such as rigid motion and rotation play a central role. In robot manipulation the geometry of the three-dimensional space is of utmost importance, and these types of topics will be presented in this chapter, along with other robot related subjects.

3.1.1 Rotations

A rotation matrix in three dimensions belongs to the group $SO(3)$, the special orthogonal group of order 3 (Spong et al., 2006). The set of all matrices that are orthogonal and have a determinant equal to 1 exist in this group, defined by:

$$SO(3) = \{\mathbf{R} \mid \mathbf{R} \in R^{3\times 3}, \quad \mathbf{R}^T\mathbf{R} = \mathbf{I} \quad \text{and} \quad \det \mathbf{R} = 1\}. \tag{3.1}$$

The rotation matrix $\mathbf{R}_b^a$ from frame a to frame b can be interpreted in two ways; either as the coordinate transformation from b to a or as a rotation matrix where a vector $\mathbf{p}^a$ in a is rotated to the vector $\mathbf{q}^b$ by $\mathbf{q}^a = \mathbf{R}_b^a\mathbf{p}^a$. The rotation of a composite rotation is the product of the rotation matrices, such that $\mathbf{R}_c^a = \mathbf{R}_b^a\mathbf{R}_b^c$.

[†]This chapter bears similarities to chapter 3 in the specialization project (Gravdahl, 2018). However, multiple sections are rewritten, and several new sections are added

For rotations ϕ, θ and ψ around the principal axes, x, y and z respectively, the following holds and can be proven, where $s\alpha = \sin\alpha$ and $c\alpha = \cos\alpha$ (Egeland and Gravdahl, 2002):

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix}, \mathbf{R}_y(\theta) = \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \text{ and } \mathbf{R}_z(\psi) = \begin{bmatrix} c\phi & -s\phi & 0 \\ s\phi & c\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.2}$$

3.1.2 Transformations

The concept of a rotation matrix can be expanded to include both orientation, as in Equation (3.2), and position, $\mathbf{d}$, of one coordinate frame relative to a another frame. This is called a homogeneous transformation matrix $\mathbf{T}$, defined as:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{d} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in SE(3), \tag{3.3}$$

where $SE(3)$ is the special Euclidean group of order 3, defined as:

$$SE(3) = \left\{ \mathbf{T} \mid \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{d} \\ \mathbf{0}^T & 1 \end{bmatrix}, \quad \mathbf{R} \in SO(3) \quad \text{and} \quad \mathbf{d} \in \mathbb{R}^3 \right\}. \tag{3.4}$$

When utilizing a homogeneous transformation matrix, we may describe the pose of a coordinate frame in terms of either a pure rotation, a pure translation or a combination of the two, relative to another coordinate frame. The inverse of the transformation matrix is defined as $\left(\mathbf{T}_b^a\right)^{-1} = \mathbf{T}_a^b$ and the product of composite transformations is the same as for rotation matrices $\mathbf{T}_c^a = \mathbf{T}_b^a \mathbf{T}_b^c$ (Egeland and Gravdahl, 2002). Transformation matrices form the basis of the Denavit-Hartenberg convention which is a popular way to describe the forward kinematics of a robot configuration, as will be further explained in Section 3.5.

3.2 Workspace

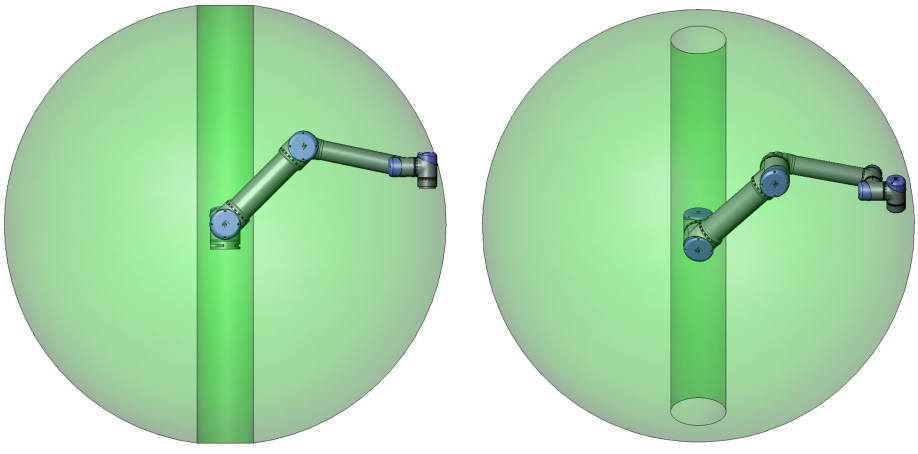


Figure 3.1: Workspace of the UR5 (Universal Robots A/S, 2016). © 2009-2019 Universal Robots A/S

A robot is composed of links with joints in-between to form a kinematic chain. A joint can either be revolute, have a rotation, or prismatic, have an extension. The workspace of a robot is the volume swept out by the manipulator end-effector, or more easily put, its reach. This space is limited by the manipulator geometry and a manipulator is often chosen based upon the workspace, with regards to the task the robot is to perform in mind. Constraints on motors and actuators will need to be taken into account as well as the robot work cell and possible obstacles. The workspace is commonly split into the dexterous workspace and the reachable workspace. The reachable workspace consists of all the points the robot can reach with its end-effector. Moreover, the dexterous workspace is the space where the robot for example can grasp an object and still move all its joints, and not be at a singularity. The dexterous workspace is then a subset of the reachable workspace. (Spong et al., 2006).

If a robot is to avoid an obstacle or locate an object, using workspace coordinates is advantageous since the coordinates of the robot position and the position of the object would be given in the same coordinate frame. However, not all coordinates by this representation are within reach, since the robot is non-linearly constrained by its own geometry, for example by a link between two joints. Using the configuration space assists in solving this problem. (Russell and Norvig, 2016)

3.3 Configuration space

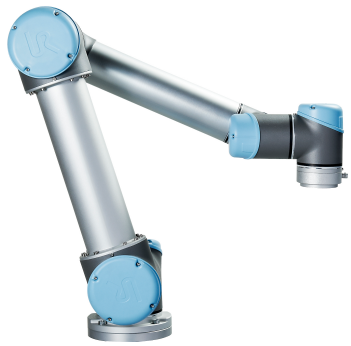


Figure 3.2: Photo of the UR5 6DOF robot (Universal Robots A/S, 2016). © 2009-2019 Universal Robots A/S

An alternative to representing the manipulator in Cartesian coordinates as in Section 3.2, is to represent it as a function of its joint variables. If the manipulator is a chain of rigid links, its base is fixed and the values of the joint variables are known, it is straightforward to find the coordinates of any point on the manipulator.

Let $\mathbf{q} = [q_i, q_{i+1}, ..., q_n]^\top$ be a column vector of dimension n, where the elements q are the joint angles, i is the number of the joint and n is the number of joints. We may then state that the robot is in configuration $\mathbf{q}$, where $q_i = \theta_i$ for a revolute joint and $q_i = d_i$ for a prismatic joint (Spong et al., 2006).

The configuration of a manipulator is then the complete specification of the location of every point on the manipulator (Russell and Norvig, 2016). The set of all possible configurations is called the configuration space $\mathcal{C}$, or the C-space. This space represents the set of all transformations that can be applied to a robot given its kinematics (Kavraki and LaValle, 2008). The advantage of the C-space is that a robot with a complex geometry can be mapped to a single point in this space, where the number of joints, n is the dimension of the space. This space is particularly useful when it comes to motion planning as will be further demonstrated throughout this text.

The number of joints n of a robot is commonly referred to as the degree of freedom, DOF, of the robot. For example, a robot with six revolute joints will have six degrees of freedom since we have three parameters to denote its position and three to denote its orientation in $\mathbb{R}^3$. From this is follows that the C-space of the UR5 is of dimension 6.

3.4 Forward Kinematics

The forward kinematics problem is concerned with finding the position and orientation of the last link in the kinematic chain, often the position of the end-effector with an attached tool, for example a gripper. This is done by attaching a coordinate system to each link in the chain making up the robot. Usually one does this design by attaching coordinate system $o_i x_i y_i z_i$ to link i, such that the coordinates of the i^{th} link are constant in coordinate system i.

With each joint, a joint variable q_i, is associated; often $q_i = \theta_i$ for revolute joints and $q_i = d_i$ for prismatic joints. The inertial frame is usually attached at the robot base and denoted as the 0-frame, $(o_0 x_0 y_0 z_0)$. This can also be named the world frame. When each link has been assigned a coordinate system and with the homogeneous transformation matrices in mind, suppose that $\mathbf{A}_i$ is the transformation matrix giving the position and orientation of coordinate frame $(o_i x_i y_i z_i)$ with respect to $(o_{i-1} x_{i-1} y_{i-1} z_{i-1})$. Since we have introduced the general coordinate q_i, $\mathbf{A}_i$ is a function of only one joint variable dependant on if the joint is revolute or prismatic, $\mathbf{A}_i = \mathbf{A}_i(q_i)$.

By this we may now express every point on the robot in terms of its joint variables q_i and by multiplying $\mathbf{A}_1(q_1)$ up until $\mathbf{A}_n(q_n)$ we can find the position and orientation of the last attached coordinate system given in the inertial frame:

$$\mathbf{T}_n^0 = \mathbf{A}_1(q_1) \cdot \ldots \cdot \mathbf{A}_n(q_n) \tag{3.5}$$

The forward kinematics problem is thus to find the pose of the final attached coordinate system, usually given in the inertial frame. The problem consists of knowing the joint variables and multiplying outwards in the kinematic chain. By using different conventions, such as the DH-convention, this is a solvable problem with one solution (Spong et al., 2006).

3.5 Denavit-Hartenberg convention

The Denavit-Hartenberg convention, or DH-convention, is used to find the forward kinematics of a manipulator. By following the convention, which entails choosing frames for each link in a set fashion, the problem is simplified and each transformation $\mathbf{A}_i$ is a product of four basic transformations. The final product for each link will be:

$$\mathbf{A}_i = \mathbf{Rot}_z(\theta_i) \cdot \mathbf{Trans}_z(d_i) \cdot \mathbf{Trans}_x(a_i) \cdot \mathbf{Rot}_x(\alpha_i), \tag{3.6}$$

where θ_i, d_i, a_i and α_i are parameters associated with link and joint i,

$$\mathbf{Rot} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{bmatrix} \text{ and } \mathbf{Trans} = \begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{p} \\ \mathbf{0}^\top & 1 \end{bmatrix}, \tag{3.7}$$

where the appropriate rotations $\mathbf{R}$ and translation $\mathbf{p}$ is used such that

$$\mathbf{A}_i = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.8}$$

By a systematic and proven way of choosing the coordinate frames for the manipulator one can decrease the number of variables needed from six to four. For a complete overview of how to choose the frames according to this convention, see for example the book "Robot modeling and control" by Spong et al. (2006).

3.6 Inverse kinematics

The general inverse kinematics, IK, problem is the opposite of the forward kinematics problem. Given the pose of the end-effector or the desired origin of the last attached coordinate frames, how must the joint variables be assigned to achieve this configuration? So, given the desired pose of the end-effector, $\mathbf{H} \in SE(3)$, find a solution which satisfies the following relation by identifying $q_1 \dots q_n$ (Spong et al., 2006).

$$\mathbf{T}_n^0(q_1 \dots q_n) = \mathbf{H}, \tag{3.9}$$

$$\mathbf{T}_n^0(q_1 \dots q_n) = \mathbf{A}_1(q_1) \cdots \mathbf{A}_n(q_n) \tag{3.10}$$

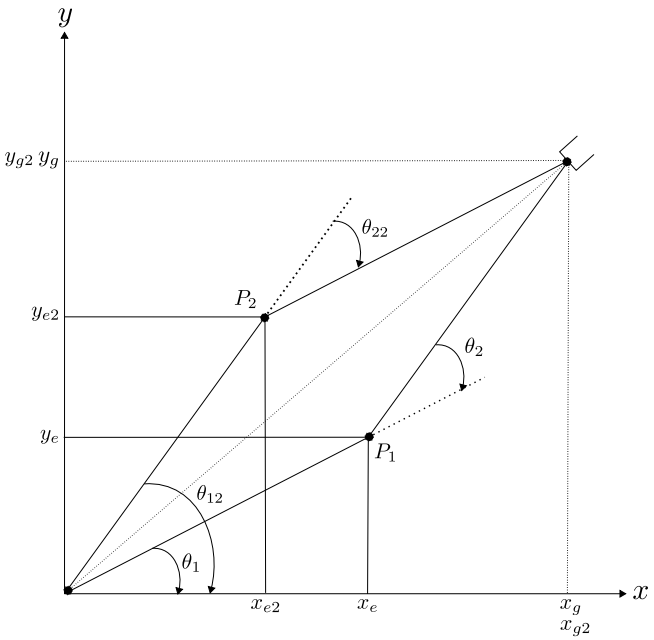


Figure 3.3: Example of the "elbow up, elbow down" phenomenon yielding multiple IK solutions, illustrated with a planar 2DOF robot.

A problem with inverse kinematic solutions is that they are not unique. This is generally not a problem on paper, but when working with a robot, this leads to choices. The simplest example is a 2DOF planar robot, which will have two solution, elbow joint up or elbow joint down. Which configuration to choose is a choice that needs to be made by the engineer, often with a specific task in mind, or considering the robot work cell.

Consider the simple planar robot with two joints shown in Figure 3.3. Moving the joints changes the (x, y)-coordinates of the gripper and the elbow. The robot can thus be described by the coordinates of the elbow and gripper, (x_e, y_e) and (x_g, y_g) respectively, relative to the environment. These four coordinates describe the full workspace of the robot (workspace coordinates) and the manipulation of these coordinates allow full reach in the workspace and can describe the location of the manipulator in the environment. We may also represent the robot configuration by its joint angles instead of the joint positions in the configuration space (θ_g, θ_e) (Russell and Norvig, 2016).

Viewing the figure, it is clear that the combination of (x_e, y_e) and (x_g, y_g) as well as (x_{e2}, y_{e2}) and (x_{g2}, y_{g2}) in workspace coordinates, and (θ_1, θ_2) as well as $(\theta_{12}, \theta_{22})$ in configuration space coordinates will all yield the same pose for the end-effector. With this, we know that the IK problem for this 2DOF planar manipulator has two solutions.

3.7 Motion planning

When dealing with robots, the objective is usually to allow the robot to execute some task, re-position and complete the same, or a different, task. Since all robotic manipulation includes some degree of motion, motion planning is central for a robotic system. A difference between between path planning and trajectory planning is common to consider.

3.7.1 Path planning

As previously stated, a robot has a unique configuration space, C-space, which is the set of all possible robot configurations. When planning a collision-free path, ensuring that the robot does not make contact with an obstacle is central. Defining the space the robot can move in and the space to avoid can be done by using the workspace $\mathcal{W}$ and the configuration space $\mathcal{C}$, by dividing the C-space into the *configuration space obstacle* and the *free configuration space* (Spong et al., 2006).

From Kavraki and LaValle (2008); let the closed set $\mathcal{O} \subset \mathcal{W}$ represent the workspace obstacle region, and $\mathcal{A}(\mathbf{q}) \subset \mathcal{W}$ represent the points occupied by the robot at configuration $\mathbf{q} \in \mathcal{C}$. Then the C-space obstacle region can be defined as:

$$\mathcal{C}_{obs} = \{\mathcal{A}(\mathbf{q}) \cap \mathcal{O} \neq \emptyset\}. \tag{3.11}$$

Since $\mathcal{A}(\mathbf{q})$ and $\mathcal{O}$ are closed sets in $\mathcal{W}$, the obstacle region $\mathcal{C}_{obs}$ is a closed set in $\mathcal{C}$. By this, the set of configurations that avoid collisions, $\mathcal{C}_{free}$ is defined as:

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}. \tag{3.12}$$

To summarize, the configuration space obstacle is the set of all robot configurations for which the robot collides with an obstacle and the free configuration space is the set difference between the whole configuration space and the obstacle. (Spong et al., 2006)

The path planning problem can be summarized as finding a path from $\mathbf{q}_s$, an initial joint configuration, a starting point, to $\mathbf{q}_f$, a final joint configuration where obstacles are avoided when traversing the path. Formally defined by Spong et al. (2006) a collision-free path from $\mathbf{q}_s$ to $\mathbf{q}_f$ is a continuous map;

$$\gamma : [0, 1] \to \mathcal{C}_{free}, \text{ where } \gamma(0) = \mathbf{q}_s \text{ and } \gamma(1) = \mathbf{q}_f. \tag{3.13}$$

Methods for path planning, to find these collision-free paths, are many, and the use of sampling-based motion-planners will be explained in Chapter 5. Path planning does not depend on the variable time, whilst the next section on trajectory planning does just that (Spong et al., 2006).

3.7.2 Trajectory planning

A trajectory, unlike a path, is a function of time. By employing the same notation as for path planning, $\mathbf{q}_s = \mathbf{q}(t_s)$ and $\mathbf{q}_f = \mathbf{q}(t_f)$, where the difference $t_f - t_s$ is the time needed to execute a given trajectory. From this we can present path planning as a subset of trajectory planning where a path is a trajectory completed during one time unit. In some cases, paths are specified by a series of end-effector poses, where the IK solution gives a sequence of joint configurations. Since the trajectory is time varying, velocity and acceleration information can be found by differentiation, and as such these variables can be controlled and used in the planning. Consider for instance the velocity close to an obstacle in comparison to the velocity which can be employed in the free space (Spong et al., 2006).

3.8 Robot Operating System

Having defined bin-picking and accounted for some of the primary aspects of robotics, other necessities in the robotic system are introduced here. Furthermore, the structure of the robot operating system, ROS, will be briefly explained, and the main ROS building blocks used in this thesis will be accounted for. The following sections consider the specific inner workings of the system at hand needed to solve the task; finding a way to identify with what ease a robot can reach a specific grasp pose. The whole system naturally consists of much more than what is supplied here, but this is beyond the scope of this work.

"ROS, Robot Operating System, is an open-source, meta-operating system for your robot" (ROS, 2019). ROS runs on UNIX-based platform, and aims to be language independent, for exmaple the interface for Python is utilized in this work. On of the goals of ROS is to increase collaboration, and to facilitate code reuse independently of the hardware one works with. As long as manufacturers of robots supply the drivers, ROS can be used on practically any robot independent of the proprietary languages of manufacturers. This in turn enables increased sharing of competence and code (ROS, 2019).

ROS has three levels, the filesystem level, the computational graph level and the community level. The community level consists of resources such as ROS Wiki and the distributions (ROS, 2019). The filesystem level consists of the packages installed on the computer when ROS is installed, and the computational graph level is how these packages interact with each other, and the robot at hand.

The computational graph level dictates the interaction between data structures in ROS. The Master in the computational graph keeps track of all the nodes in network. "A node is a process that performs computation" ROS (2019). One node can perform path planning, and another node could visualize the scene (such as the ROS Visualizer, Section 3.8.1). The nodes connect to each other through the use of messages, a data structure. To identify the content of a message, topics are used. A node can publish to a topic or subscribe to one. Nodes often do not know if another node is listening to the topic they are publishing, and vice versa. For more direct request and replies, services are used. "A node can offer a service, and a client node can use the service by sending the request message and awaiting the reply" (ROS, 2019).

3.8.1 ROS Visualizer

ROS Visualizer, RViz, is a 3D visualization tool for ROS, a graphical interface. It supports using plug-ins for many kinds of available topics, and allows the user to visualize the robot workspace, the robot itself and for examples paths that are executed. Rviz is utilized in this work due to the nature of the problem at hand, path planning and grasp analysis (ROS, 2019), it is an advantage to see paths being executed. Many of the images presented further in this thesis are screengrabs from RViz.

3.9 MoveIt

MoveIt functions on top of ROS. It builds on the ROS messaging system and utilizes RViz for visualization. MoveIt provides functionality for motion- and path planning, kinematics and collision-checking etc., and is the primary source for a lot of robot manipulation in ROS (MoveIt, 2019). An advantage of using MoveIt is that it supports the use of the Universal Robotic

Description Format (URDF). The format is a common way to describe the design and geometry of a robot manipulator. This allows for introducing a complete visual of the robot in RViz (ROS, 2019). MoveIt uses the URDF for collision-checking, so that a complete description of the robot, and for example its tool if this is large, is advantageous (MoveIt, 2019). The URDF-file for the SINTEF set-up includes the camera housing at its end-effector, allowing this to be considered when checking for a collision when planning to a pose. Path planning is central in this work and the MoveIt plug-in was quickly utilized in conjunction with the Open Motion Planning Library.

3.9.1 Open Motion Planning Library

These sections will briefly introduce OMPL, whilst Chapter 5 will go into detail on what sampling-based motion planner are, the functionality of different sample-based planners, and their fundamental building blocks. The Open Motion Planning Library, OMPL for short, is a library for sampling-based motion planners, and contains implementations of a variety of state-of-the-art algorithms designed to solve motion planning problems. OMPL is the most used, and maintained, planning library for the MoveIt plug-in for ROS (Şucan et al., 2012).

The motivation behind OMPL is traced back to that finding paths and motion plans for a wide variety of robotic applications is critical for optimal utilization of the system. To illustrate, Şucan et al. (2012) use the example of search-and-rescue robots. These robots are dependant on being able to continuously plan safe and efficient paths through rubble and obstructed environments when completing their task, and without a motion planner this would be quite difficult. OMPL was designed for motion planning research, education within robotics and end-users in robotics industry purposes and intended to be useful in practical applications, making it wide in its use.

3.9.2 `tf`

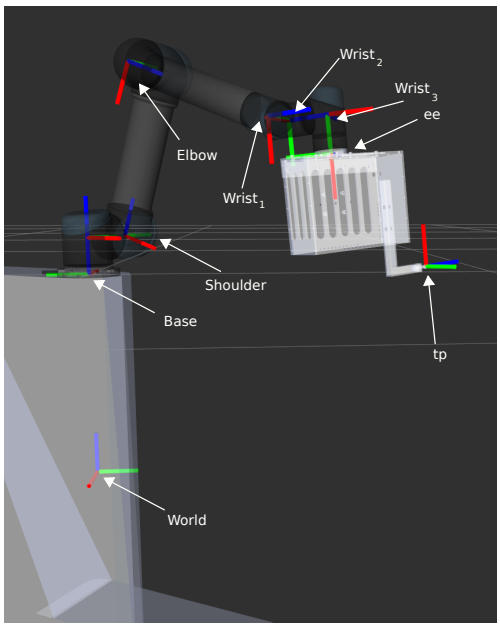


Figure 3.4: The robot based on the URDF-file, visualized in RViz using `tf` to keep track of coordinate frames.

`tf` is a library enabling keeping track of multiple coordinate frames over time. For any manipulation task, information on where an object is in relation to the robot is decisive. To be able to compare the two, the pose of the object and that of the robot end-effector, it is common to attach a coordinate system to each object and present the transformation of each of them in the same coordinate system. `tf` was designed to solve this issue and provide a standard way to keep track of frames and transform data in robotic systems (Foote, 2013).

The data structure of `tf` is a graph, where coordinate frames are nodes and transforms are the edges between them (Foote, 2013) (a more comprehensive explanation on graphs and trees is available in Section 5.1.1). For example, viewing Figure 3.4, the frame at the base, and the frame at the shoulder link of the robot can be seen as nodes, and the transform required to bring the shoulder to the base is the edge between them. Bringing the base to the shoulder

would require the inverse transform. It is common to represent all the transforms representing a robot with respect to the same fixed frame, most often the base frame. Viewing the figure, the transform of the final coordinate frame at the tool point, tp for short, given in the base frame is given by the following relation and is maintained by the `tf` library:

$$T_{\text{tp}}^{\text{base}} = T_{\text{shoulder}}^{\text{base}}\, T_{\text{elbow}}^{\text{shoulder}}\, T_{\text{wrist}_1}^{\text{elbow}}\, T_{\text{wrist}_2}^{\text{wrist}_1}\, T_{\text{wrist}_3}^{\text{wrist}_2}\, T_{\text{ee}}^{\text{wrist}_3}\, T_{\text{tp}}^{\text{ee}}. \tag{3.14}$$

The coordinate frames supplied by the library follow the color code $(x, y, z) \rightarrow$ RGB. The x-axis is always red, y is always green and the z-axis is always blue. It should also be noted that the frame seen inside the pedestal below the base frame is the world frame.

3.9.3 move_group

The `move_group` node provides functionality for, among other things, setting path goals, creating motion plans and executing them. The node communicates with the robot through topics and actions. For example it communicates to obtain current state information from the `joint_states` topic. Moreover, the node monitors transform information from the ROS `tf` library so it at all times is aware of the robot's pose in the workspace. In this work, the python interface for the `move_group` node using the `MoveGroupCommander` class was used. The class contains multiple functions, for example to set motion planning goals, specifying the planner from OMPL to be used and setting an allowed planning time (MoveIt, 2019).

One advantage of ROS is the ability to visualize the computational graph, a connection of running nodes with connecting topics. This aids in seeing which nodes are communicating and supplies the user with valuable information regarding data flow. A simple example of such a graph is shown in Figure 3.5. The Figure depicts several nodes in circles, connected by topics, the arrows between the nodes. Observe for example the node which is the `move_group`. This node receives information on coordinate systems circled in blue over the topic `/tf` which keeps track of the different transforms present in the scene. The nodes circled in blue are four coordinate systems defining the corners of the bin in the set-up. The `move_group` node also receives information from the `ur_driver` on the status of the `joint_trajectory` for example. The node `ur_driver` will be supplied by the manufacturer of the robot, and enables the user to program the robot through ROS since relevant robot information is stored in this node.

Another example of data flow in Figure 3.5 is the `ur_driver` providing the `joint_states` to the `robot_state_publisher` which again supplies this as a `tf_static` (static transform) to the `move_group`. Also take note of the node `/tf2_buffer_server` storing the transforms present in the scene.

What can be deduced from this graph is that the `move_group`, the node that is used for path planning has information on the robot state at any given time, along with any other coordinate systems present in the scene. This makes it possible for example to instruct the `move_group` to align two coordinate systems since it has the information on all of them. For grasping, as is an important topic in this work, the `move_group` can be instructed to align the coordinate system of the tool point of the robot (found in the `/robot_state_publisher` through the topic `joint_states` supplied by the `ur_driver`) with for example a bin corner (any node circled in blue), or any other arbitrary coordinate system published to the robot scene. If two coordinate systems can be aligned, a path from one to the other exists, the coordinate system is reachable by the robot.

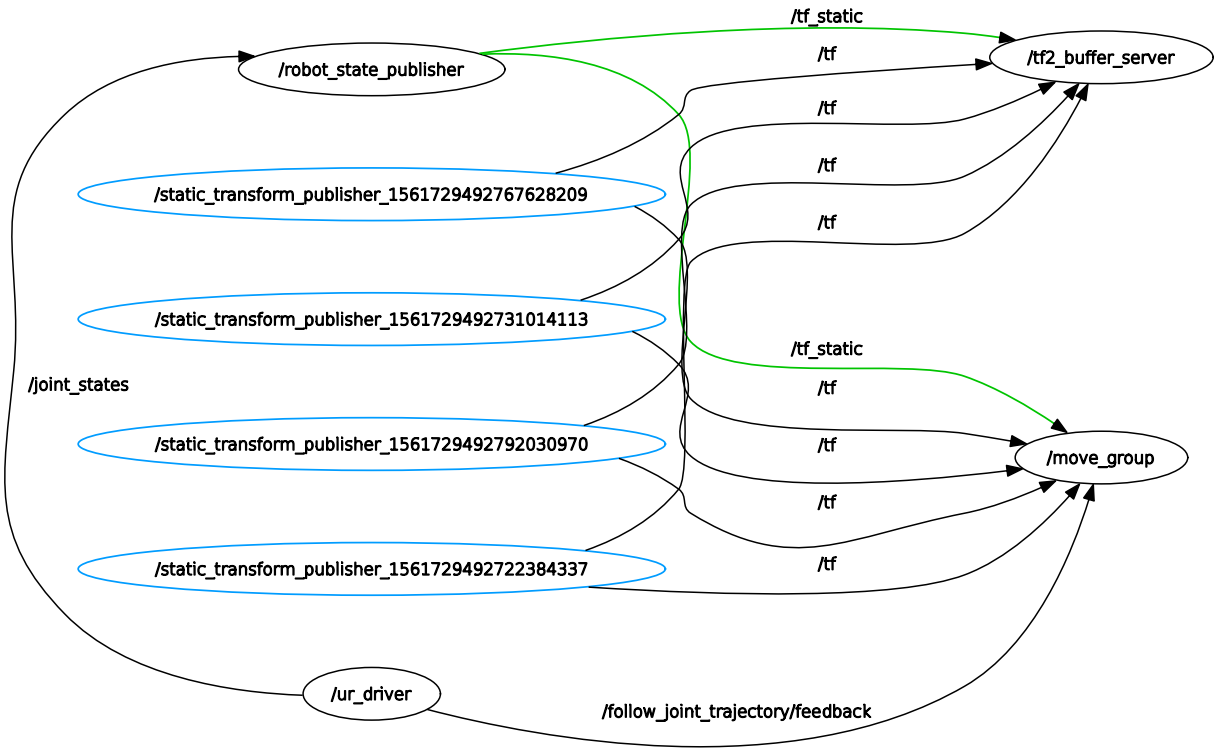


Figure 3.5: Example of a ROS computational graph depicting dataflow between nodes and what topics are communicated over. Nodes circled in blue are coordinate systems defining the bin corners in the set-up, and the topics marked in green is the robot configuration given as a series of transforms at a given time instance.

Chapter 4

Grasping

In a bin-picking system, identifying and being supplied grasps is central. In this chapter, the definitions of a good grasp will be explored, grasping with different kinds of tools will be briefly considered, and the functionality of a vacuum gripper will be explained. Furthermore, how grasps are obtained will be looked into, along with a detailed explanation on how these "good grasps" are found in this set-up using a dual-resolution convolutional neural network (Dyrstad et al., 2018).

4.1 What is a "good grasp"?

4.1.1 Defining grasping

[†]To motivate and define grasping, forces and torques are commonplace. These forces and torques can be represented in a 6D space, called the wrench space. Let $\mathcal{W}_w$ denote this space. A wrench is a vector $\boldsymbol{w} \in \mathbb{R}^p$, and is defined according to Ferrari and Canny (1992) as:

$$\boldsymbol{w} = \begin{pmatrix} \mathbf{F} \\ \boldsymbol{\tau} \end{pmatrix}, \tag{4.1}$$

where $\mathbf{F}$ is the force components of the wrench vector, and $\boldsymbol{\tau}$ is the torque vector. In considering 3D objects the dimension of this space is $p = 6$. Advantages of using the wrench space includes that any force and torque can be represented by a point in this space (Ferrari and Canny, 1992). The magnitude of a wrench is also defined by Ferrari and Canny (1992) as:

$$\|\boldsymbol{w}\| = \sqrt{\|\mathbf{F}\|^2 + \lambda \|\boldsymbol{\tau}\|^2}, \tag{4.2}$$

where choosing $\lambda = 1$ is equivalent to the L_2 norm of the wrench. The objective behind introducing wrenches is that some grasps are better than others, and grasps that can withstand and balance any external force and torque at the contact points can be considered good grasps. This leads to closure properties of grasps.

[†]This section is similar to material also found in the specialization project. It is supplied here to complement the documentation (Gravdahl, 2018)

Consider a gripper with N contact points. This can either be a parallel gripper, a three-fingered hand, or another multi-fingered tool used for grasping. These contact points can be modelled differently, often as one of three different models; a frictionless contact point, a frictional point contact or as a soft contact. A frictionless contact point is considered a contact where only the normal force in the contact point exists. In a frictional point, both a normal force and a tangential force can be exerted. A soft contact "also allows the finger to exert a pure torsional moment about the common normal at the point of contact" (Bicchi and Kumar, 2000). At each of these contacts, the object is subjected to a normal force, a tangential force and a a torsional moment about the normal (Bicchi and Kumar, 2000).

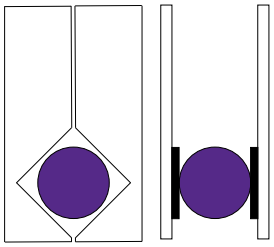


Figure 4.1: Illustration of form- and force closure. The drawing is inspired by a similar figure found in Bajd et al. (2010).

Let $\mathbf{W}$ denote the combined wrench matrix, containing the N wrenches of the N contact points on an object. A grasp has the property of force closure if for any arbitrary wrench vector $\hat{\boldsymbol{w}}$, there exists an intensity vector $\boldsymbol{\lambda}$ such that (Bicchi and Kumar, 2000)

$$\mathbf{W}\boldsymbol{\lambda} = \hat{\boldsymbol{w}}. \tag{4.3}$$

Similarly, another closure property is form closure. A grasp with N wrenches is said to be form closed if there exists a $\boldsymbol{\lambda} > 0$ and $\mathbf{W}$ is full rank such that (Bicchi and Kumar, 2000)

$$\mathbf{W}\boldsymbol{\lambda} = 0. \tag{4.4}$$

Figure 4.1 illustrates examples of both form closure (left) and force closure (right) for a two-fingered gripper, where the objective is to grasp the purple circle. The property names reflect how the object is constrained by the gripper.

4.1.2 Vacuum gripper

The use of a suction cup differs from the use of robotic fingers in terms of there only being one contact point. The force exerted on the object is the force needed to compress the suction cup, since a valid grasp is often one where complete vacuum can be achieved such that the object can be lifted. Suction is useful in a wide variety of applications due to an increased freedom in what objects can be grasps (Kessens and Desai, 2011). The size of a two-finger gripper for example must be adjusted to accommodate the size of the object such that it is able to fulfill for example force-closure as seen in Figure 4.1. A suction cup however, as long as the surface area of the cup is within the appropriate size range to be able to lift, is not that dependent on the design of the object. Consider also the weight of an object, independent of its geometry. A two-finger gripper is more focused on the geometry of the object in terms of finding stable contact points to exert a force. A suction cup can be more versatile in terms of manipulating objects of similar weights as long as one contact surface can be found to obtain vacuum upon contact. The use of a suction cup can increase the robots capability of manipulating the object (Kessens and Desai, 2011).

Tools or objects attached at the final link impose the greatest torque on the robot. As with any tool attached to the robot end-effector, its weight and design dictate payload capabilities. The size of a suction cup must be optimized to maximize the range of manipulable object shapes and sizes (Kessens and Desai, 2011). A larger suction cup will be able to lifter larger and heavier things, whilst a smaller suction cup will be more able to adjust to the surfaces of objects to be picked. If the surface of an object is rough, a smaller suction cup would aid in obtaining a

complete seal, but if the rough object in addition is large, a larger suction cup must perhaps be utilized to be able to lift it. A solution to this weight-off is to utilize several suction cups of the appropriate size to fulfill the demand of a complete seal, and use enough cups to be able to lift (Kessens and Desai, 2011).

A good grasp with a suction cup can be summarized as a grasp where the suction cup is of the appropriate size to obtain a seal on the object and be able to lift it, whilst exerting a force that compresses the cup to obtain this seal, whilst not moving the objects to be picked with this force.

4.2 Obtaining grasps

Now that grasping has been introduced, the next question is how these grasps are obtained in a system. The following two sections will explain grasp planners briefly, and give a thorough explanation on how the grasps in this system are found. A grasp planner often relies on force-closure and the notion of being able to resist a wrench applied at a contact points, whilst these demands are not prominent with the use of a suction cup.

4.2.1 Grasp planners

Grasp planners are a popular way of obtaining grasps for different applications. In essence, a grasp planner takes the end-effector/tool and an object as input, and supplies as output valid grasps on the object by evaluating some metric, for example based on the force-closure property. What has been an obstacle for grasp planners in robotic applications, is that they often assume that the tool and the object are alone in the environment, and often that they are free-floating in space. For example, consider the objects in the bin in this set-up. If a grasp planner was used to plan grasps on the objects, it would just as often spend time on evaluating grasps coming up through the table than it would spend time on evaluating grasps from above, assuming one and one cylinder alone in the environment (Akinola et al., 2018). Of course, the only valid grasps would be the ones from above since we cannot grasp through the table. OpenRAVE (Diankov and Kuffner, 2008), OpenGRASP (León et al., 2010) and GraspIt! (Miller and Allen, 2004) are examples of grasp planners. It should also be mentioned that several grasp planners are including robot kinematics in the grasp planning schemes due to demand.

4.2.2 System at hand

The grasps in this system are supplied by a dual-resolution convolutional neural network trained on simulated data (Dyrstad et al., 2018). The motivation behind this application will be described here. Furthermore, the specific format of the grasps supplied to the robot will be accounted for.

The motivation behind utilizing a neural network as opposed to a grasp planner, is the desire to develop a system independent of the objects to be picked aiming for a more general solution to the bin-picking problem. As previously mentioned, a grasp planner is dependent on the model of both the object and the tool to do the picking, but with this approach it will be more available to reuse the network on other applications, both robot-, gripper- and object-wise. Even though also the network requires a model of the object, it can be retrained. (Dyrstad et al., 2018)

The objects to grasp in this set-up are reflective steel parts, which results in the images lacking data in varying degree due to reflectivity. The workaround, is to train the network on a large data set of synthetic images meant to simulate the missing data in the real images. By training on synthetic images one avoids the process of labeling a large data set, by supplying pointers for the network instead. A number of valid grasps were supplied to the network for training, as well as the model of the object. (Dyrstad et al., 2018)

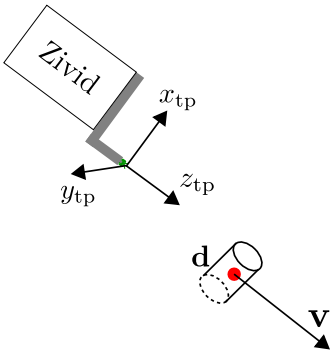


Figure 4.2: Illustration of the end-effector tool and grasps supplied by the neural network. The goal is to align z_{tp} with $\mathbf{v}$, and the the tool center point with $\mathbf{d}$.

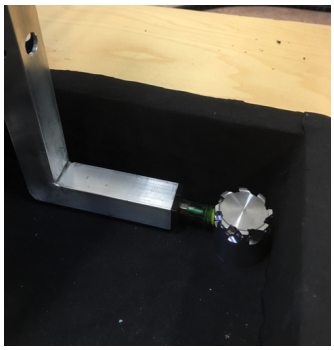


Figure 4.3: Successful grip with gripper. $\mathbf{v}$ is aligned with z_{tp}, and the contact point is at $\mathbf{d}$. Photo courtesy of Katrine Seel, MSc, SINTEF Digital

The use of a dual-resolution network allows for twofold consideration of the scene. The network enables high accuracy in a focus region when placing a grasp and estimating the pose of an object, as well as enabling enough of an overview of the scene to avoid collisions with other objects to picked in the bin. (Dyrstad et al., 2018)

A grasp is considered valid if there are no collisions present with either other objects in the bin or the bin itself. The use of a local collision-checker ensures these collisions are avoided, but a collision-check for the robot manipulator arm is not considered when planning for the grasps. Multiple grasps are generated for each part, but only one is chosen. This choice is made by favouring grasps which are close to the world z-axis, and in the direction of the camera frame. (Dyrstad et al., 2018)

The input to the neural network is a depth image of the current distribution of objects in the bin courtesy of the 3D camera, and the output is multiple grasp pairs, $\{\mathbf{d}_i, \mathbf{v}_i\}$, where $i \in \{1, \ldots, N\}$, $N \in \mathbb{N}$, $\mathbf{d}_i \in \mathbb{R}^3$ is a point and $\mathbf{v}_i \in \mathbb{R}^3$ is an approach vector.

When grasping an object in this set-up there is a coordinate frame attached to the suction cup, the tool point, and the objective is to align the z_{tp}-axis with the approach vector $\mathbf{v}$ supplied and $\mathbf{d}$ with the tool center point, see Figure 4.2. Since the approach vector is not a complete coordinate system, a rotation about this axis can present multiple solutions. This indicates that a rearrangement of the joint angles of the robot might aid in obtaining a valid grasp.

Going forward, it is assumed that grasps supplied by the neural network are optimal in the sense of grasp quality. When the network supplies its list of grasps, we assume these are indeed valid and preferred grasps. With this assumption, considering the robot abilities in terms of reaching these grasps will be considered.

Chapter 5

Motion planning in-depth

The objective of motion planning is to find a collision-free path from an initial start configuration $\mathbf{q}_s$ to a final configuration $\mathbf{q}_f$ whilst avoiding obstacles in the workspace and self-collisions. It is clear that the collision-free path from $\mathbf{q}_s$ to $\mathbf{q}_f$ is a continuous map $\gamma : [0, 1] \rightarrow \mathcal{C}_{free}$, where $\gamma(0) = \mathbf{q}_s$, $\gamma(1) = \mathbf{q}_f$ and $\mathcal{C}_{free}$ is the unobstructed part of the configuration space, as defined in Section 3.7. Since motion-planning is a decisive part of this work this chapter will deal with the subject from an algorithmic perspective to gain insight in the planners and their functionality the way they are implemented, and used, in OMPL, and how it connects with the work done.

5.1 The geometric path planning problem

The fundamental motion planning problem, or the geometric motion planning problem, must be defined to be able to pick it apart and study its components. The following definition is adopted from the Springer Handbook of Robotics, Part 5: Motion Planing (Kavraki and LaValle, 2008) and is as follows:

Given

1. A workspace $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An obstacle region $\mathcal{O} \subset \mathcal{W}$.
3. A robot defined in $\mathcal{W}$. Either a rigid body $\mathbf{A}$ or a collection of n links: $\mathbf{A}_1$, $\mathbf{A}_2$, ..., $\mathbf{A}_n$.
4. The configuration space $\mathcal{C}$ (both $\mathcal{C}_{free}$ and $\mathcal{C}_{obs}$ are then defined).
5. An initial configuration $\mathbf{q}_s \in \mathcal{C}_{free}$.
6. A goal configuration $\mathbf{q}_f \in \mathcal{C}_{free}$. The pair $(\mathbf{q}_s, \mathbf{q}_f)$ is often called a query.

Compute a continuous path, $\gamma : [0, 1] \rightarrow \mathcal{C}_{free}$, such that $\gamma(0) = \mathbf{q}_s$ and $\gamma(1) = \mathbf{q}_f$.

This problem is also known as the piano mover's problem. When viewing this list of requirements to be able to solve the motion planning problem, the effort of finding all the components vary. For example, the workspace is often dependent on the robot geometry, and should be possible to acquire. The same goes for the obstacle region, provided the obstacles are stationary in the environment. $\mathcal{C}_{free}$ and $\mathcal{C}_{obs}$ however, are more challenging to obtain due to the complexity of directly computing them. In addition, the dimensionality of the C-space is often high (Kavraki and LaValle, 2008).

In terms of the computational complexity of the problem, it was found and can be shown, that the problem is PSPACE-hard (Kavraki and LaValle, 2008). PSPACE-hard problems are

problems that can be solved by a deterministic Turing machine with a polynomial amount of space; these are difficult problems. PSPACE-hard problems are believed to be more difficult than NP-hard problems. A problem is NP-hard if if there is an algorithm that can guess a solution to the problem and verify this solution in polynomial time (Russell and Norvig, 2016).

It is due to the complexity of the path planning problem that alternative approaches to finding good paths emerged in the form of sampling-based planning. Sampling-based planners do not model the exact geometry of the C-space and avoids this computationally expensive operation, but at the cost of not being able to provide the guarantees of a complete algorithm. A complete and exact algorithm is able to provide the definitive answer; "there does not exist a path to this configuration". Instead, sampling-based planners will, provide a weaker form of completeness; "if a solution path exists, the planner will eventually find it", this is known as an algorithm being probabilistically complete (Şucan and Kavraki, 2010). Sampling-based planners are for this reason, and the fact that they have proved themselves efficient for a large number of different problems, the method of choice for a very general class of problems (Kavraki and LaValle, 2008).

Sampling-based planners investigate $\mathcal{C}_{obs}$ implicitly, by exploiting advances in collision detection algorithms that compute whether a configuration is collision-free. A planner samples different configurations to construct a data-structure that stores one-dimensional C-space curves which represent collision-free paths. Since $\mathcal{C}_{obs}$ is not entered directly, these planners are applicable to a large range of robots considering that they make use of collision-detection software which is a vital part of any robotic system which requires paths to be found. The way the valid paths are chosen depends on what objective a planner uses. A sampling-based planner which seeks to minimize the path length will thus choose the shortest paths from configuration to configuration until the goal is reached. Planners differ in how they sample configurations, and what type of data structure is constructed to hold the collision-free paths, and a typical distinction is between single- and multi-query approaches. (Kavraki and LaValle, 2008)

5.1.1 Graphs and trees

A brief introduction to graphs and trees is needed before discussing singe- and multi-query planners, since these data structures are heavily used in these algorithms. The majority of this background theory is gathered from "Introduction to Algorithms" by Cormen et al. (2009). First, the concept of a graph is introduced along with a quick explanation. Then, trees are introduced and briefly explained. Graph theory is a vast topic and only a small selection of terms, examples and explanations are given in the following sections.

5.1.1.1 Graphs

A directed graph G is a pair (V, E), where V is a finite set and E is a binary relation on V. An example of a directed graph is shown in Figure 5.1. The set V is the vertex set on G and its elements are the vertices of the graph. The set E is the edge set of G, and its elements are the edges of G. Vertices are often represented by circles, and edges by arrows between them (Cormen et al., 2009).

In an undirected graph, $G = (V, E)$, the edge set E consists of unordered pairs of vertices rather than ordered pairs which is the case for an ordered graph. An example of an unordered graph can be seen in Figure 5.2. That is, an edge is a set $\{u, v\}$, where $u, v \in V$ and $u \neq v$. Using the

convention in Cormen et al. (2009) and using the notation (u, v) for an edge, in an undirected graph (u, v) and (v, u) is the same edge. (Cormen et al., 2009)

A path of length k from a vertex u to a vertex u' in a graph $G = (V, E)$ is a sequence $\langle \nu_0, \nu_1, ..., \nu_k \rangle$ of vertices such that $u = \nu_0$ and $u' = \nu_k$, and $(\nu_{i-1}, \nu_i) \in E$ for $i = 1, 2, .., k$. The length of a path is the number of edges in the path. If there is a path p from u to u', one says that u' is reachable from u via p. In a directed graph, a path $p = \langle \nu_0, \nu_1, ..., \nu_k \rangle$ forms a cycle if the starting position is the same as the final position; $\nu_0 = \nu_k$. In an undirected graph, a path $p = \langle \nu_0, \nu_1, ..., \nu_k \rangle$ forms a cycle if $k > 0$, $\nu_0 = \nu_k$, and all edges on the path are distinct. A graph with no simple cycles is acyclic. The term simple here refers to all vertices in the path being distinct (Cormen et al., 2009).

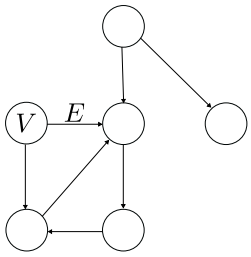


Figure 5.1: An example of a directed graph

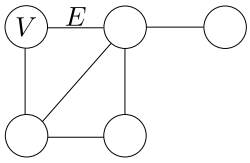


Figure 5.2: Example of an undirected graph

5.1.1.2 Trees

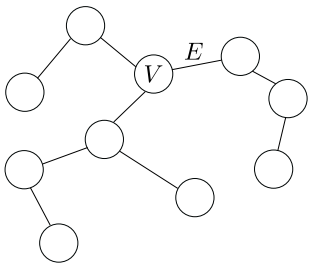


Figure 5.3: An example of a free tree.

Trees can be said to be a subset of graphs, since all trees are graphs, but not all graphs are trees. A free tree is an undirected graph which is acyclic, where the term "free" often is omitted. An unconnected tree is often referred to as a forest, since it consists of two or more stand-alone trees. In a tree all vertices are reachable from other vertices through simple paths, and any vertex can be the starting node. As mentioned a tree is acyclic, but with a cycle it is neither a tree nor a graph. A tree can also be rooted, where one of the vertices are distinguished from the others and called the root of the tree. A vertex of a rooted tree is often referred to as a node. (Cormen et al., 2009)

5.2 Multi-query planners

A distinction is made between multi- and single-query planners, recall that the pair $(\mathbf{q}_s, \mathbf{q}_f)$ is often called a query. In multi-query planners, an undirected graph G, where the edges are collision-free paths, and the vertices are collision-free configurations, is computed once for a static problem. The undirected graph is created to map the connectivity properties of $\mathcal{C}_{free}$ in a precomputation step. When the graph is completed, multiple queries can be made in the same environment using the same graph, often called a roadmap. (Kavraki and LaValle, 2008)

An example of a multi-query planner is the Probabilistic Roadmap Method, PRM, created by Kavraki et al. (1996). PRM consists of a learning phase, and a query phase. In the learning phase, robot configurations in $\mathcal{C}_{free}$ are randomly sampled and checked for collisions before they are declared vertices V to create a probabilistic roadmap. The simple edges, E, between them are connected using a fast internal motion planner. This internal motion planner is called the local planner, and is often adapted depending on the application. The local planner attempts to connect two vertices in the graph with a line segment. Let $\mathbf{q}_1, \ldots, \mathbf{q}_m$ denote the discretized

line segment of m configurations, such that for any consecutive configurations $(\mathbf{q}_i, \mathbf{q}_{i+1})$, $\mathbf{q}_i$ is no longer than some ε away from the robot configuration at $\mathbf{q}_{i+1}$, where ε is a small positive constant. For each configuration, $\mathbf{q}_{i+1}$, ε away away from $\mathbf{q}_i$, the configuration is checked for collisions. If all m configurations are found to be collision-free, the path is deemed valid and can be used as an edge E (Kavraki et al., 1996). The value of ε is often determined experimentally dependant on the layout of the robot workspace and the obstacles present, since checking every point along the line would require an infinite amount of calls to collision-checking (LaValle, 2006).

The roadmap created is attempting to present the connectivity of $\mathcal{C}_{free}$, and is saved as an undirected graph $G(V, E)$. The roadmap does not uniformly cover $\mathcal{C}_{free}$, but is denser in more difficult to reach regions of the workspace. By increasing the connectivity in these regions, the planner is able to efficiently solve problems requiring it to maneuver in narrow passages (Kavraki et al., 1996).

Following the learning phase, several queries, $(\mathbf{q}_s, \mathbf{q}_f)$ can be answered using the same graph, since the graph is a representation of $\mathcal{C}_{free}$. This is done by specifying the start and final configurations, and attempting to map these two configurations to nodes in the graph. When this is achieved, a graph search is performed with the objective of identifying edges connecting the two. An edge (u, v) is a feasible path connecting the two configurations, since the local planner has already found the edge valid. When the appropriate edges are identified, the concatenation of these path segments results in a valid path for the robot (Kavraki et al., 1996).

The use of a collision-checker is decisive in path-planning. The goal of the collision-checker is to return information on possible contacts between objects in the workspace. For sampling-based planner to be effective, fast collision-checking is crucial. There exists many different collision-checkers, and some can return information on how close to a collision a configuration is. This can aid in predicting more generally about the validity on larger regions of $\mathcal{C}$. When traversing the graph on search for a collision-free path, paths are often validated using a collision-checker incrementally with a small step along the valid edges of the graphs (Kavraki and LaValle, 2008), such as the method of the local planner described above.

The type of local planner is important when considering time consumption. If a powerful planner is used, in terms of the probability of it finding a path if one exists, this results in fewer calls to the planner, but more time spent on each call. Consequently fewer configurations are needed to answer a query. If a less powerful planner is used, it will need more configurations to map the connective of $\mathcal{C}_{free}$, but each call to the planner will be computationally cheaper. Considering that the queries need to be answered "quasi-instantaneously" (Kavraki et al., 1996), a fast planner is preferred, where a more dense roadmap is constructed. It is also worth noting that the best results obtained by Kavraki et al. (1996), were found using a deterministic local planner.

To summarize, multi-query planners can be represented by the following steps, quoted from the Handbook of Robotics, chapter 5 (Kavraki and LaValle, 2008). Initialization: Let $G(V, E)$ be an undirected initially empty graph. Vertices will correspond to collision-free robot configurations, and edges to collision-free paths. Configuration sampling: A configuration is sampled from $\mathcal{C}_{free}$ called $\alpha(i)$, where $\alpha(\cdot)$ is a sample sequence and i is the i^{th} entry in this sequence. Neighborhood computation: A metric $\rho : \mathcal{C} \times \mathcal{C} \to \mathbb{R}$ is defined as a distance measure used to locate configurations, $\mathbf{q}$, in the neighborhood of $\alpha(i)$. Edge consideration: For vertices $\mathbf{q}$ not belonging to the graph yet, an attempt to connect them to the existing graph by an edge is made. Local planning method: Given $\alpha(i)$ and $\mathbf{q} \in \mathcal{C}_{free}$, a local planner is used to construct a collision-free path between the two. Edge insertion: The edge from $\alpha(i)$ to $\mathbf{q}$ is inserted into the

set E. Termination criteria: The algorithm usually terminates when a predetermined number of vertices are identified and in the map. (Kavraki and LaValle, 2008)

5.3 Single-query planners

Single-query planner operate slightly different from multi-query planners. These types of planners build tree structures when simultaneously attempting solve a query $(\mathbf{q}_s, \mathbf{q}_f)$. The objective of these planners is to map the connectivity of $\mathcal{C}_{free}$ fast, for a single query.

With single-query planners, the graph G can be organized in several ways. One method is to allow $\mathbf{q}_s$ to be the root of a rooted tree and then span the tree looking for a solution from this node. Another option is to grow two trees, one from $\mathbf{q}_s$ and one from $\mathbf{q}_f$, and connect the trees when they are within range of each other. This bi-directional approach may aid in finding paths in narrow passages in the workspace. Sampling-based methods combine sampling and searching to overcome the problems of local minima such as narrow passages (LaValle, 2006).

An approach that yields good results, uses an incremental sampling and searching approach to construct a search tree which covers $\mathcal{C}_{free}$ densely. A dense sequence of samples is used in the construction of the tree. If this sequence of samples is random, the resultant tree is called a Rapidly Exploring Random Tree, an RRT, and if it is deterministic, the resultant tree is referred to as a Rapidly Exploring Dense Tree, an RDT. These methods were originally designed to handle motion planning under differential constraints (LaValle, 2006), like limitations on the robot velocity v and acceleration a (Şucan et al., 2012).

In the tree, several main branches are constructed first to reach the outer portions of $\mathcal{C}_{free}$, and more and more branches are added gradually as more vertices and edges are found, densely covering the space. As the number of iterations increases, the resolution of the representation of $\mathcal{C}_{free}$ increases also. Recall as well that $\mathcal{C}_{obs}$ is not explicitly represented and that collision-checking when constructing the tree is important. The higher the resolution the more certain one can be that obstructions will be avoided (LaValle, 2006).

When the tree is constructed and $\mathcal{C}_{free}$ is explored, a query can be answered by performing a search in the tree. There are several ways to search for the valid path from $\mathbf{q}_s$ to $\mathbf{q}_f$. Consider a single-tree approach where the tree is grown from $\mathbf{q}_s$. An approach is to densely sample with a sequence $\alpha(\cdot)$, and insert $\mathbf{q}_f$ at regular intervals to investigate if this vertex is reachable from the closest vertex to it in the tree grown from $\mathbf{q}_s$. Another approach is to perform the bi-directional search, where two trees are grown, one from $\mathbf{q}_s$ and one from $\mathbf{q}_f$. In a balanced bi-directional search, the trees are kept equal in size (LaValle, 2006).

The summary of steps undertaken for single-query planners is also given in the Handbook of robotics, chapter 5 by Kavraki and LaValle (2008), and is repeated here. Initialization: Let $G(V, E)$ represent an undirected search graph, where V contains a vertex for one or more configurations, where $\mathbf{q}_s$ is commonly present, and the edge set E is empty. It is also not uncommon that $\mathbf{q}_f$ is in V (LaValle, 2006). As with multi-query planners, the vertices are collision-free robot configurations and the edges are collision-free paths. Vertex selection method: Choose a vertex $\mathbf{q}_{current} \in V$ for expansion. Local planning method: For some $\mathbf{q}_{new} \in \mathcal{C}_{free}$ which is either a sam-

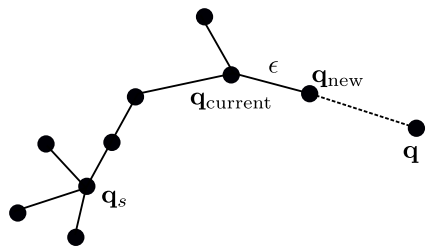


Figure 5.4: Figure inspired by Kuffner and LaValle (2000). Illustration of an RRT extending its cover of a space by growing the tree one ϵ at a time.

pled configuration or part of another tree, attempt to plan a path such that $\mathbf{q}_{current}$ is the start state and $\mathbf{q}_{new}$ becomes to goal state. The resultant edge must be checked for collision. If this step fails, $\mathbf{q}_{current}$ is chosen anew. Inserting the edge in the graph: Insert the new found edge in E, as an edge from $\mathbf{q}_{current}$ to $\mathbf{q}_{new}$, and if $\mathbf{q}_{new}$ is not already in V, it is added as a vertex. Check for a solution: Determine if $G(V, E)$ constitutes a solution to the query $(\mathbf{q}_s, \mathbf{q}_f)$. If a solution has not been found, the algorithm again returns to deciding upon a new $\mathbf{q}_{current}$.

5.4 Path planners in use

Meijer et al. (2017) have undertaken a study of the performance of the different planners included in OMPL, and tested each one on three different manipulators, and three different objectives, to benchmark results depending on the task to be solved. Based upon their work, three planners, which later will be used for testing with the set-up at hand, were selected for this work. OMPL supplies 23 different sampling-based motion planners, and as such to make a choice based upon already executed benchmarking was advantageous. The three planners of interest and their functionality will be detailed in the following sections, whilst motivation behind the choice, their use and specific performance for the set-up at hand will be accounted for in Chapter 7.

5.4.1 LBKPIECE

In the case of real-world robotic application, dynamic constraints need to be taken into account. This is what is known as kinodynamic motion planning. The algorithm LBKPIECE, lazy bi-directional kinodynamic planning by interior-exterior cell exploration, is one such planner. The LBKPIECE is the lazy, bi-directional version of KPIECE which was designed specifically for robotic applications with complex dynamics. In addition to fulfilling the goal of creating motion plans for complex systems, there is no requirement for state sampling or use of distance metrics between states (Şucan and Kavraki, 2010).

The algorithm KPIECE iteratively constructs a tree of motions in the configuration space of the robot. Each motion μ is a function of a state (configuration), a control and a time duration; $\mu(s, u, t)$. It is also possible to split the time duration into t_1 and t_2 and superposition the motions. The control u is applied for the duration of t from s, which is what produces a motion. During the exploration and growing of this tree, it is a requirement to try to cover as much of the configuration space as possible. The KPIECE uses an approximation of the state space by employing grids of different resolutions which reflect the degree of previous exploration by the tree. For example, a course grid can be used to identify regions where little exploration has been done, and a finer grid to see where in this region more exploration is warranted. The discretization of the state space for KPIECE consists of k levels, $\mathscr{L}_1 \dots \mathscr{L}_k$, where each grid is a multiple of a previous one in resolution. For the implementation in OMPL of LBKPIECE there is only one level of discretization. The first level is the one with the highest resolution (Şucan and Kavraki, 2010).

The prefix LB in LBKPIECE means that planner is lazy and bi-directional. This means two trees are grown and maintained, one from $\mathbf{q}_s$ and one from $\mathbf{q}_f$, in the hope of them connecting somewhere in the middle of the path. A lazy planner in this context means that collision-checking of nodes and edges is delayed until the query phase. When a solution candidate is identified in the tree, a path from $\mathbf{q}_s$ to $\mathbf{q}_f$, only then are the configurations and path segments checked to see if they are in $\mathcal{C}_{\text{free}}$ (Şucan et al., 2012). If a path contains a collision, the edges

and nodes that constituted a collision are often removed from the tree before a new query is made, and this process is continued until a collision-free path is found. If a node is in collision, all edges associated with this configuration are removed, so a check of the node is often made first (Bohlin and Kavraki, 2000).

5.4.2 RRTConnect

RRTConnect was originally designed to plan motions for a human arm modelled as a 7DOF kinematic chain for animated grasping and manipulation. However, it has proved itself useful in planning collision-free motions for rigid bodies both in 2D and 3D, as well as for a 6DOF PUMA robotic manipulator arm. The objective behind the design of the algorithm was the desire to maintain the good properties of a probabilistic roadmap, a multi-query scheme, in terms of reliability but designed for single-query path planning. (Kuffner and LaValle, 2000)

Recall, from Section 5.3 that a Rapidly-exploring Random Tree, an RRT, is a data structure and sampling-scheme which searches high-dimensional spaces like the C-space, and attempts to cover the space by expanding its tree structure as illustrated in Figure 5.4. The key idea behind an RRT is to explore unexplored regions of this space. Furthermore RRTs arrive at a uniform coverage of the space, which is another advantage of probabilistic roadmaps there was interest in projecting onto a single-query planner. (Kuffner and LaValle, 2000)

The RRTConnect algorithm is designed specifically for path planning problems without differential constraints. The method is based on two ideas; growing two trees, one from $\mathbf{q}_s$ and one from $\mathbf{q}_f$, and using a heuristic "that attempts to move over longer distances" (Kuffner and LaValle, 2000). Viewing Figure 5.4, where the growth of an RRT is shown, the Connect heuristic is a greedy function which instead of extending the tree one ϵ, iterates this step until either $\mathbf{q}$ or an obstacle is reached. The configuration $\mathbf{q}$ in this instance is a collision-free configuration in the space (Kuffner and LaValle, 2000).

The algorithm RRTConnect builds two trees, $\mathcal{T}_s$ and $\mathcal{T}_f$, one from $\mathbf{q}_s$ as can be seen in Figure 5.4, and an equal one from $\mathbf{q}_f$ where both are maintained simultaneously. In each iteration of the algorithm, a new vertex is added to either of the trees and an attempt to connect them is made. When trying to connect the trees, the nearest vertex in the other tree and the vertex just added are tried. The roles of the trees are then swapped and a new vertex is attempted added to the other tree. The algorithm is based on RRTs explained in Section 5.3, and when advancing towards a new configuration in the space, collision-checking is done before the edge is inserted in the trees. When the tree is constructed, queries can be answered quickly (Kuffner and LaValle, 2000).

5.4.3 SBL

The algorithm SBL, Single-Query, Bi-Directional, Lazy in Collision Checking, is as the name suggest both single-query and bi-directional. It is to solve a single motion planning problem, and it builds two trees, one from $\mathbf{q}_s$ and one from $\mathbf{q}_f$ attempting to connect these two. It is a PRM planner, it builds a probabilistic roadmap to answer the query, and it has lazy collision-checking, meaning it delays checking for collisions until it is absolutely necessary. As with other sampling-based planners, $\mathcal{C}_{free}$ is not mapped explicitly, but a collision-checker returns whether $\mathbf{q} \in \mathcal{C}_{free}$, given any $\mathbf{q} \in \mathcal{C}$ (Sánchez and Latombe, 2003).

A PRM planner, as accounted for in Section 5.2 samples the configuration space at random and saves the collision-free configurations as vertices V. The edges between the vertices are

simple paths, straights segments in the space, and denoted E. The sets V and E make up the probabilistic roadmap. A PRM can either be multi-query, where a roadmap is built once and queried multiple times for the same environment, or single-query, building a new roadmap for each query. In a changing environment, a single-query planner will be preferred.

According to Sánchez and Latombe (2003), PRM planners spend most of their time collision-checking. Several ways to reduce this time usage is listed by the authors and include designing faster collision-checkers, building smaller roadmaps to reduce the time spent checking for a collision and delaying collision-checking until it is needed. The authors opted for the latter option (Sánchez and Latombe, 2003). In comparison to RRTConnect for example, path segments are only added to the tree if it is found collision-free during the growth phase of the tree. Here, the path segments are added without this consideration, and this is dealt with afterwards.

Sánchez and Latombe (2003) define a metric d over $\mathcal{C}$: "for any $\mathbf{q} \in \mathcal{C}$, the neighbourhood of $\mathbf{q}$ of radius r is the subset $B(\mathbf{q}, r) = \{\mathbf{q}' \in \mathcal{C} \mid d(\mathbf{q}, \mathbf{q}') < r\}$. With $d = L_\infty$, the metric used by SBL, is an $n - D$ cube" (Sánchez and Latombe, 2003).

The algorithm is given two parameters; s, which is the number defining how many vertices V are to be generated in the roadmap and ρ, a distance threshold. Two configurations, for example $\mathbf{q}_1$ and $\mathbf{q}_2$ are considered close, if the L_∞ distance between them is less than ρ (Sánchez and Latombe, 2003);

$$\text{iff } \sup\big(|(\mathbf{q}_1 - \mathbf{q}_2)|\big) < \rho \rightarrow \mathbf{q}_1 \text{ close to } \mathbf{q}_2. \tag{5.1}$$

The planner builds two trees $\mathcal{T}_s$ and $\mathcal{T}_f$, rooted at $\mathbf{q}_s$ and $\mathbf{q}_f$. Which tree is grown varies to keep the size of the trees consistent. If one tree grows disproportionately large, the advantages of bi-directional search disappears (Sánchez and Latombe, 2003).

For each expansion of the roadmap, a collision-free vertex is added to the roadmap. The "lazy" part of the algorithm with regards to delaying collision-checking is reserved for checking the path segments. In each expansion of one of the trees, the two are attempted connected by the previously defined metric. Let m denote the newly added vertex, and m' the closest vertex to it in the other tree (the one not being grown at this iteration). If the distance between them, $d(m, m') < \rho$, m and m' are connected by an edge called the bridge. The alternating growth of the trees is done s times, and if a path γ from $\mathbf{q}_s$ to $\mathbf{q}_f$ has not been found there either does not exist a path, or it has not been found. Recall, that sampling-based planners are non-deterministic. If a bridge is found, this is the final path segment needed to connect $\mathcal{T}_s$ and $\mathcal{T}_f$ via the path γ (Sánchez and Latombe, 2003).

It is at this point the path segments in the trees, including the bridge, are tested for collision. It is known at this point, that the vertices are collision-free since this was a demand for them to be added to the roadmap, but the segments have not been checked. For each path segment, w, there is an associated index $\kappa(w)$, indicating how much of the path segment has been checked for collision. If $\kappa(w) = 0$, only the vertices connected by w are tested collision-free, whilst if $\kappa(w) = 1$, both vertices and the midpoint of w are collision-free. The necessary collision-checking is then performed to investigate if this path answers the query (Sánchez and Latombe, 2003).

If the path is not collision-free, the segment containing the collision is removed from the graph. This results in the roadmap becoming two trees again. If the colliding segment is the bridge, the two trees are the same as before the collision-checking. If the colliding segment is another segment than the bridge, a transfer of vertices from one tree to another is made of the collision-free vertices, and a new attempt is made to connect the two trees (Sánchez and Latombe, 2003).

Chapter 6

Motion planning + grasping

Bin-picking problems and set-ups are a concoction of different technologies, utilizing different forms of technology, from computer vision to pose estimation to robot actuator control. Multiple fields are combined to find solutions to the bin-picking problem. During the theory work for this thesis, and for the specialization project, one finds readily available "separate solutions" to the different sub problems that go into the bin-picking problem, such as looking at optimal grasp planning assuming the tool point or end-effector already is at the pose of the grasps, and optimal motion planning to a pose, not considering if this pose is an optimal grasp. The hypothesis and basis for this work regarding this conundrum is that a combination of optimal grasp and optimal path planning to said grasp must be combined to obtain good results.

6.1 Reachability vs. Path Reachability

"The reachability of a robot manipulator to a target is defined as its ability to move its joints and links in free space in order for its hand to reach the given target" (Ying and Iyengar, 1995). Furthermore, Ying and Iyengar (1995) define the "Reachability Test" as testing "whether the robot has the ability to move its joints for its hand to reach the given target". Recall, that all the points the robot can reach in the workspace is commonly called the reachable workspace. With the definition of the reachable workspace as stated, the reachability of a given point is a problem of testing whether the point is within the reachable workspace of the robot. If at least one point on an object can be reached, the object is said to be reachable by the robot. (Ying and Iyengar, 1995)

When transferring this to a bin-picking system, the given target is the given grasp we would like to reach. How well a robot can reach this given grasp is closely related to its reachability. If the reachability of the robot is "good" for a given grasp we can be more certain that the robot will be able to reach the grasp.

To the extent of the author's knowledge, there exists some ambiguity on the use of the term reachability in robotics. In (Ying and Iyengar, 1995), they define the reachability of a robot as something to do with the robot's ability to move. However, for example in Zacharias et al. (2007), Zacharias et al. (2009) and Berenson et al. (2007) it refers to the existence of an inverse kinematic solution only. The existence of an IK solution is not equivalent to a valid motion plan existing, especially not in a robot scene with constraints and obstacles. In the rest of this work, reachability will refer to the existence of an IK solution, and the term *path reachability* is introduced, and will refer to instances where an IK solution and a collision-free path exists.

Definition 6.1. Consider a point p in the reachable workspace of a robot manipulator. If there exists an inverse kinematic solution at p, a motion plan can be found from the current configuration $\mathbf{q}_s$ to the final configuration $\mathbf{q}_f$ at p, and this path from $\mathbf{q}_s$ to $\mathbf{q}_f$ is in $\mathcal{C}_{free}$, meaning it is collision-free, then the point p is said to be path reachable.

6.2 Separated solutions

[†]Bin-picking is, as previously mentioned, a concoction of different technologies, and branches within those technologies. Attempting a solution of the bin-picking problem by solving it part by part seems a good strategy due to the complexity of the system as a whole. Combining solutions to subsystems is reasonably assumed to lead to the solution of the system as a whole. As such, following this reasoning, much research has been done on one of two things; finding good trajectories to reach a desired pose and finding high quality grasp candidates. This leads to a new problem: an optimal pose for grasping may not be feasible to reach for the robot due to constraints in the workspace (Akinola et al., 2018).

There exists extensive previous work on the notion of grasping an object given that the end-effector is already at the appropriate contact point, to initiate the actual grasping. If one is given a good grasp it does not matter that it is perfect if the robot cannot reach it (Akinola et al., 2018).

Taking into account that the robot must be able to reach the grasp pose is decisive for a successful bin-picking system. Given an optimal grasp from a grasp planner, or a vision-based system as is the case with the SINTEF set-up, at least an inverse kinematic solution must exist for the given grasp pose. If there are no constraints on the reachability of the robot, the optimal grasp is the best ranked grasp from the planner. However, considering that the arm kinematics and the reachability of the robot is a constraint on the system, additional consideration is necessary.

In its simplest form, checking for an inverse kinematics solution at the grasp candidate poses takes the arm kinematics and the robot into the equation. Once one has a list of grasp candidates, the robot constraints are introduced and the reachability comes into play. If there are no additional constraints, this is enough to find out if a grasp is feasible or not (Saut and Sidobre, 2012). If the workspace is clear of all obstruction and constraints other than the demand for reachability, an inverse kinematics solution is a sufficient condition. The process then goes from

→ Obtain grasp candidates
→ Choose the optimal candidate based on grasp quality measures

to

→ Obtain grasp candidates
→ Check the IK solution for candidates
→ Choose the optimal candidate based on grasp quality measures and that an IK solution exists

Saut and Sidobre (2012) state that if there are no additional constraints other than the robot constraints an IK solution is sufficient for introducing the robot into the grasp planning process.

[†]The majority of the section labeled "Separated solutions" is copied from the specialization project (Gravdahl, 2018)

Seeing as this is rarely the case in a real-life application such as with the set-up at hand, an IK solution should perhaps be viewed as only a necessary condition to be certain of the reachability of a pose in the workspace. Since there are constraints and obstructions with the camera housing attached to the robot arm, and the bin itself constitutes a possible collision, together with the objects to be picked, an IK solution does not seem to suffice. A path must also exist and this path must be collision-free, to be able to say with absolute certainty that the grasp is feasible/reachable/possible.

6.3 Combining the problem

Regardless of the method used for grasp selection, planning for a grasp is generally done, in much of the research according to Berenson et al. (2007), with the prerequisite of the object being alone in the environment. Furthermore for grasp planning, the assumption that the tool is already at the object and ready for grasping is common, and brings with it the fact that the robot kinematics are not taken into account. (Berenson et al., 2007)

Again, we encounter the issue of a perfect grasp generated for an object, might not be reachable by the robot tasked with grasping it. The assumption that the object to be grasped is alone in the environment, can hold true for multiple applications, but in our case with bin-picking we have both obstacles (pedestal) and random behaviour (distribution of cylinders in the bin changing throughout picking procedure) close to the object to be picked; the pedestal and the camera housing make up static and moving potential collisions, and the remaining contents of the bin and the bin itself constitutes constraints on the movement close to the chosen grasp. The goal should be to not only select a valid grasp, but also ensure that this grasp is feasible for the robot to reach.

In the following sections, the main results and approaches of seven particularly relevant papers on the combination of grasp and motion planning will be summarized and discussed. This presentation will be concluded by a statement on which of the works weigh the heaviest on the implementations found in Chapter 7.

6.3.1 The work of Berenson et al. (2007)

Berenson et al. (2007) combined grasp analysis and manipulation planning techniques to perform fast grasp planning in complex scenes, presented in their article titled *Grasp planning in complex scenes*. Their framework uses the force closure property to evaluate grasps, and find good grasps, when the object they are to pick is alone in the environment. The framework relies on sampling to find a set of good grasps for each of the objects in the environment. When having found this set for the objects that are to be grasped, they pose the question; *"which one should be chosen for a given environment?"*. Instead of choosing the first grasp that fulfills the force-closure property, is reachable and has a collision-free path to it, they propose a more organized fashion to handle this choice.

The set of grasps which fulfill the force-closure property is potentially large, and checking each of the grasps randomly can become a time consuming affair. This approach is disregarded, and they propose using a *grasp-scoring function*, that take into account the kinematics of the robot, as well as the local environment around the objects to be picked. The so-called grasp-scoring function is used to rank grasps in the grasp set, obtained from the sampling procedure, and when they are ranked, bidirectional RRTs are used to plan a path. See Chapter 5 for more information on motion planners, including RRTs.

Their method is two-fold, one precomputation step which is done offline and one on-line computation step. In the precomputation step, the result is a grasp set where all grasps fulfill force-closure. Grasps that do not fulfill this property are not of interest. In the online step, the first part is to rank the grasps according to the grasp-scoring function which takes into account the local environment of the object and the robot kinematics, and then test the grasp according to their ranking by looking for an inverse kinematic solution and a collision-free path. Finally a trajectory is planned to the grasp with the highest ranking. Shall this trajectory fail, the grasp ranked second by the grasp-scoring function is tried. The process continues like this until a valid trajectory is found. Since the grasps already have been ranked, one knows that a grasp with a valid trajectory also fulfills force-closure.

The grasp-scoring function objective is to trim away unnecessary computation time spent on useless computations which arise when the environment and the manipulator kinematics are taken into account. The force-closure property is only a sufficient demand on the grasp if the object and the end-effector are floating in free space, which is never the case. Considering that a contact surface for the object always will exist, such as for example a metal cylinder resting in the bin in the SINTEF set-up, grasps through the bottom of the bin need never be considered. Knowing this increases the efficiency of the grasp planning, since a number of theoretically valid grasps on the object from below will never be valid due to the environment design. Moreover, the kinematics of the robot and its reach can trim away more invalid grasps with a quick check into the inverse kinematics of the robot, or grasps found outside of the robot workspace (Berenson et al., 2007). Thus, the goal of the grasp-scoring function is to take into account all necessary information about both the robot kinematics and the environment, based on the probability that a grasp be successful, again based on the surroundings of the grasp.

Let

$$G(O, E, \mathbf{P}), \tag{6.1}$$

be the score of the grasp, defined by the grasp policy parameters $\mathbf{P}$, on object O, in environment E. The object considered is defined by the parameter O, and E changes when the environment does. The function G consists of three parts. $G_q(\mathbf{P})$, which is the force-closure score of $\mathbf{P}$ found during the precomputation step. $G_b(E, \mathbf{P})$, which is the robot-relative position score, which is the parameter that takes into account the position of the robot in the scene. $G_e(O, E, \mathbf{P})$ is the environment clearance score, which allows for consideration of the environment around the object when computing the grasp score. After the previously defined Gs are calculated, they are combined:

$$G(O, E, \mathbf{P}) = e^{c_1 \cdot G_q(\mathbf{P})} \cdot e^{c_2 \cdot G_b(E, \mathbf{P})} \cdot e^{c_3 \cdot G_e(O, E, \mathbf{P})}, \tag{6.2}$$

where $c_i, i = 1, 2, 3$ are weighting parameters determining the relative importance of the characteristics of the grasp.

Berenson et al. (2007) tested their framework on two set-ups in simulation and on one set-up in experiments. The foundation of the experiment was to compare the efficiency of the grasp-scoring function, $G(O, E, \mathbf{P})$ to the same procedure without ranking the grasps according to said function. Fifty grasp were tested. In one experiment these 50 grasps were not sorted by the grasp-scoring function, whilst in another they were. The grasp-scoring function approach outperforms the unsorted approach consistently in terms of how many attempts at grasping was needed until a successful grasp was found on the object.

The procedure described by Berenson et al. (2007) of orderly ranking grasps based on the robot kinematics and the local environment of the grasp is advantageous to consider for the robot set-

up at hand. A map of the workspace, and obtaining information about the availability of the environment around where the bin is placed, could potentially improve the picking process.

6.3.2 The work of Zacharias et al. (2007)

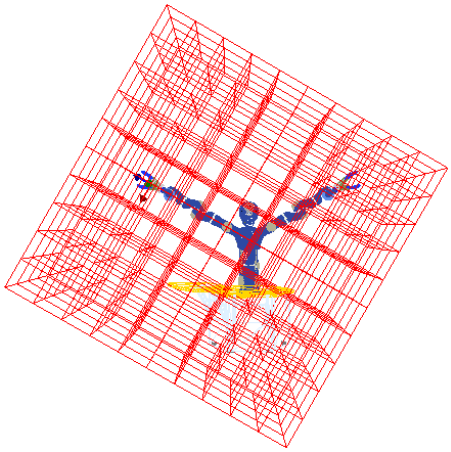


Figure 6.1: "The maximum workspace of the right arm is overestimated by the enveloping cube which is divided into subcubes of 300mm side length" (Zacharias et al., 2007). © 2007 IEEE

In the article titled *Capturing Robot Workspace Structure: Representing Robot Capabilities*, Zacharias et al. (2007) introduce a compact representation of their robots reachability and directional structure information for the whole workspace. They label this representation of the workspace as the *capability map*, and claim it to be a good representation for finding easy to reach poses in three dimensions. By also including directional structure, the robot is able to place its tool in easy to reach regions in the workspace.

A robot arm's workspace is not uniform with respect to reachability (here referred to whether or not an inverse kinematic solution exists for a certain pose), and it is therefore necessary to capture directional information of different workspace areas; there are certain regions that only can be reached from certain directions. In general, Zacharias et al. (2007) needed a representation of manipulator capabilities that could be used to deduce which places are easy to reach for the robot arm. If only a specific direction is off interest, this direction should be applicable in the map, and filtering out all other directions not of interest.

Zacharias et al. (2007) extract their workspace structure through four steps; discretization, randomized sampling, analysis and optimization processes. The discretization step envelopes the robot into a cube with length, width and height twice the arm length of their robot, and this cube is divided into smaller cubes to obtain a discrete representation of the workspace as can be seen in Figure 6.1.

In each of the subcubes, a sphere with diameter equal to the width of the subcube is generated, and N equally distributed points on the sphere are generated. For each of these points, they generate a coordinate frame, which again is rotated in each of the points. Each of these resulting frames are meant to represent a potential tool point frame which will need to be tested for reach by the robot by checking the inverse kinematic solution for this particular frame. If they find one solution for a particular point p on the sphere, this p is marked in an underlying data structure. The randomly sampled configuration is supplied to the inverse kinematic solver as the initial conditions. Since they are working with a kinematically redundant robot (7DOF) which does not have a single unique solution, supplying a starting solution to the solver already near the solution, is computationally beneficial (Zacharias et al., 2007).

When the spheres are all mapped, they visualize the reachability of each of their cubed regions. Zacharias et al. (2007) then introduce the *reachability index* D, which is a measure of the reachability of a region defined as

$$D = \frac{R}{N} \cdot 100 \text{ with } R \leq N, \tag{6.3}$$

where N is the total number of points on a sphere, and R, the number of valid inverse kinematic solutions found on this sphere. The resultant variable D will then give a percentage representation of the reachability of this region. Furthermore, the spheres are coloured according to their reachability index, which range from $D \in [0, 76]$. Looking at Figure 6.2 also taken from their article, as expected the spheres with the lowest reachability index is at the border of the workspace, and a distance away from the base at approximately half the arm length we find a region of good reachability.

The reachability index is a directionless measure, it does not inform where on the spheres the IK solutions were found. Consider that on one sphere, all the solutions were found on one side, this is not explicit in this presentation, this measure only presents an average success percentage for each sphere. The next step of Zacharias et al. (2007) was to look more closely into exactly where of the sampled spheres solutions were found, and have a more comprehensive impression of the results by also including desirable angles of approach in the regions of the workspace based on where on the spheres solutions were found.

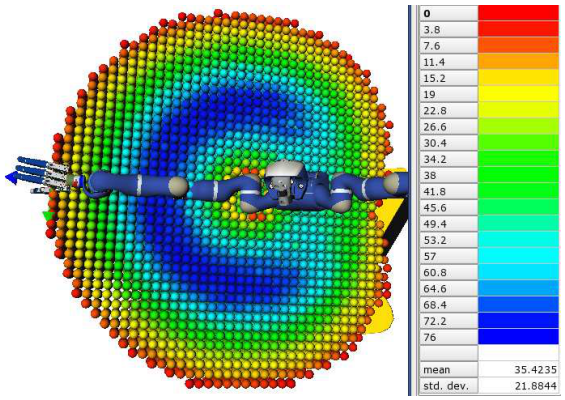


Figure 6.2: "Shows the reachability spheres across the workspace. The workspace representation was cut for better visibility of the structure" (Zacharias et al., 2007). © 2007 IEEE

This type of workspace representation is of interest within this thesis as the initial goal of this work (see Chapter 7) is to find an optimal placement of the bin in the set-up. A representation of the workspace and a corresponding visualization will aid in this process. Furthermore, directions, and where potential grasps need to be approached from is relevant for this work due to the extra freedom that reveals itself when the UR5 is given a point and approach vector from the neural network, and not a complete coordinate system.

Tying this in with the SINTEF set-up, a procedure which can say something about the best angle to come at a grasp from, or have some information about directional approach would be beneficial. For example it has been experienced that grasps which must be approached from the left of the base fail far more often than grasps from north, south and east direction

The objective of Zacharias et al. (2007) was to find a capability map of their manipulator, and due to the design of their robot which has a torso, adjust the placement or incline of this torso as to move the most capable regions of the robot, to the object to be grasped. The objective of the task with the robot at SINTEF is the other side of the coin; a mapping of the workspace was desirable to find, such as to place the bin in the most reachable region of the workspace. The goal with the set-up at hand is to move the bin to the most reachable region, whilst Zacharias et al. (2007) wanted to move the reachable region to the object.

6.3.3 The work of Zacharias et al. (2009)

In this paper, Zacharias et al. (2009), examine two strategies to incorporate the reachability of the robot into the grasp planning. Firstly, they incorporate an IK solver directly into a grasp planner, and secondly they use their previously established model of the reachable workspace, their capability map, presented in Zacharias et al. (2007), see also Figure 6.2.

The aim of their work was to obtain reachable grasps online for a multi-fingered robot. Since finding stable and optimal grasps is computationally expensive, and in addition the grasps are to be reachable, the process would not be possible to execute online. The focus was placed on a second category of planners that generate good grasps, rather than optimal ones. The randomized grasp planner that they utilize is independent of the kinematics of the robot, thus it can generate several good grasps on an object, but will also give grasps not necessarily reachable.

The planner they use have several steps to it which filter out grasps on an object; generate random contact points → check feasibility with respect to hand kinematics → check for collision, both with object and self → check for force-closure. The idea of Zacharias et al. (2009) was to implement an additional kinematic reachability test within the planner to introduce the planner to the robot. Their reachability test consisted of taking the frame attached to the hand given in the base frame and checking for an IK solution. If one could be found, the grasp passed the additional test. To ensure grasps that are not reachable do not take up additional computation time in the planner, the reachability test was placed as the first test after the step where random contact points on the object are generated.

The next module of the work in this paper consists of feeding the planner the capability map for the workspace which was presented in Section 6.3.2 discussing the generation of the map. By using the capability map the planner could be decoupled from the integration of an IK solver. Their capability map describes from what directions regions of the workspace can be approached. When introducing the map into the filtering steps in the grasp planner (generate random contact points → check feasibility → check for collision → check for force-closure) it can be determined if the region the grasp exists in is reachable, and from what direction it must be approached. The capability map is implemented in the grasp planner making it able to predict if a grasp is reachable or not, but does not supply (one of) the arm configuration which are valid at this pose. By using the capability map Zacharias et al. (2009), unreachable grasp were discarded early.

In addition to implementing the reachability concept in the grasp planner, Zacharias et al. (2009) also consider obstacles in the workspace and the need for collision-checking. An obstacle influences the reachability of nearby regions in the workspace. This region of influence also increases when the obstacle is placed close to the robot base, since the robot often operates in this region. By subtracting the obstacle region of influence from the reachability sphere map, a new capability map could be computed which included the influence of an obstacle. By giving the grasp planner this new capability map, collision-avoidance was included in the planner without introducing the robot and object model to the planner. Since the capability map is not a perfect representation, a final collision-check was performed at the end of the process.

By including the capability map in the grasp planner as another module and not changing the planner itself, the only change which needs to be made to utilize this set-up on another manipulator is to compute a new capability map for it. It is worth noting that other sources, such as Fontanals et al. (2014) also discuss and utilize such capability maps.

Seeing this in the context of the SINTEF set-up, this type of representation of the workspace and knowing something beforehand about the different regions in the workspace will be valuable in terms of sorting the grasps coming from the neural network. Since the grasp planner in this context is the output from a neural network completely decoupled from the robot kinematics, being able to quickly supply some information about the region the grasp is in, is highly valuable information to avoid unnecessary attempts at picking. The intention for the SINTEF set-up is to represent a type of capability map as a large look-up table where reachability and path information is the first to be checked.

6.3.4 The work of Vahrenkamp et al. (2010)

Vahrenkamp et al. (2010) state that for grasping an object, several tasks have to solved; finding a feasible grasp, calculating an IK solution and finding a collision-free path. Instead of working with the workspace of the robot, they incorporate the demands on grasping into a probabilistic planning approach using RRTs. Their planner searches for feasible grasps during the motion planning stage such that needing knowledge beforehand regarding grasping poses is not necessary (Vahrenkamp et al., 2010). This is in contrast with for example the work of Zacharias et al. (2009) who incorporate robot kinematics into a grasp planner. Here, Vahrenkamp et al. (2010) incorporate grasp planning in what is originally a motion planner structure. The search for feasible grasps in this implementation focuses on reachable configurations for the robot, and this limits the computation of grasp poses to those that are reachable.

Planning grasping motions based on a pre-defined set of grasps is common. An offline generated set of grasps exist, and IK solutions are searched for in the planning process, for example with the use of a grasp planner like GraspIt by Miller and Allen (2004). The combined grasp and motion planner by Vahrenkamp et al. (2010) is called "Grasp-RRT" and combines the search for a collision-free path with online planning for a feasible grasp. There is no explicit $\mathbf{q}_f$ in the motion planning problem, since multiple grasp poses and approach directions are possible until one reaches the object to be grasped. Recall, that RRTs span a tree structure attempting to cover $\mathcal{C}_{\text{free}}$ densely. The start configuration of the robot $\mathbf{q}_s$ is defined in an RRT, along with the 6D pose of the object to reach, $\mathbf{p}_{\text{obj}}$. Using $\mathbf{q}_s$ as the node of the tree, the RRT builds from this node a tree of collision-free configurations (recall that the nodes are collision-free configurations and vertices are collision-free paths). For every new configuration $\mathbf{q}_i$, the grasp center point (of the grasp), $\mathbf{p}_i$, is calculated and saved. When a new node of the RRT is selected, the tool center point is moved toward a feasible grasping pose. The resultant node in the RRT when planning towards $\mathbf{p}_{\text{obj}}$ defines a potential grasping pose. There is no need for a check of the IK solution at the potential grasping pose, because all the vertices of the tree are collision-free configurations and have paths to them, meaning the IK solutions are implicitly included. This grasping pose at the final node is then checked by a *grasp quality measure*. If this measure is above some defined limit of quality, the grasping motion is readily available since it is found in the RRT already grown.

Considering this work in relation to works utilizing a capability map, this work places a larger focus on path availability in the planning process. With the set-up at hand it was desirable to attempt paths early in the process and having some information on path existence available before attempting a grasp, not necessarily checking for a path once a grasp had been decided upon.

6.3.5 The work of Vahrenkamp et al. (2013)

A slightly different aspect of combined motion planning and grasping can be found in Vahrenkamp et al. (2013). In this work, the robot base is placed favourably to reach a given grasp, instead of favourable grasps being planned in reachable parts of a stationary base workspace. The capabilities of a robot can be presented in a map such as in the work of Zacharias et al. (2007), where IK solutions and joint limits are included. This type of map is beneficial for quickly querying about the reachability of a certain point in the workspace. The type of representation can also be used to support the search for a suitable robot base pose for grasping.

In the paper named "Robot Placement Based on Reachability Inversion", Vahrenkamp et al. (2013) present an approach for inverting reachability data in order to generate a distribution

in $SE(2)$ of potential base poses for grasping. $SE(2)$ is the special Euclidean group of order 2, and consists of the C-space where a rigid body in 2D can rotate and translate in the plane (LaValle, 2006). With their inverse reachability representation they transform the capability representation to an object-oriented view, instead of a robot oriented one (Vahrenkamp et al., 2013).

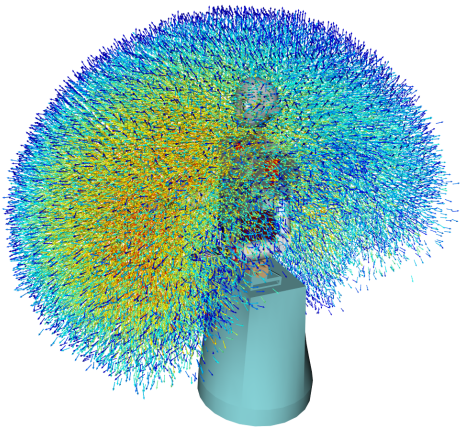


Figure 6.3: "The reachability distribution of the robot's right TCP. The kinematic chain covering hip and right arm with 10 degrees of freedom was used for generation while manipulability, self-distance and joint limits were incorporated for determining the quality" (Vahrenkamp et al., 2013). © 2013 IEEE

In an offline step, the robot's capabilities in terms of reaching and grasping is computed. This is done by discretizing the workspace into small cubes, voxels, before filling these cubes with possible tool point placements. To build the representation, reachability information (IK solutions) is used. Furthermore, information on the distance between links of the robot are used to penalize being in a configuration with limited maneuverability, and the joint limits when considering singularities. This data will be a representation of the tool point given in the base frame (Vahrenkamp et al., 2013).

Once the previously mentioned data has been generated, it can be inverted to supply suitable robot base poses with respect to grasps needed to be reached in the workspace, instead of generating a good grasp given a base pose. This inverted map can also be generated offline such that querying it online will not be too time consuming.

This type of representation is particularly useful for a mobile robot in a human-centered environment (Vahrenkamp et al., 2013). Vahrenkamp et al. present an example of a mobile robot opening a dishwasher. The placement of the robot base is a more intuitive way to solve the problem, rather than having a fixed-base robot attempt to reach the handle from a stationary position, even though a "direct" capability map might aid in finding a proper approach direction for solving the task. Considering the specific grasp needed to open the dishwasher, it is a better solution to allow this pose be the fixed one, and move the base accordingly.

Considering the use of an inverse reachability map in the work of Vahrenkamp et al. (2013) and its ability to guide the base towards a stationary grasp, confirms the idea of using a "direct" path reachability map for finding good grasps in the set-up at hand. In the SINTEF set-up, the robot base is of course fixed, but the use of the workspace representation before it is inverted and used by Vahrenkamp et al. (2013) is advantageous to investigate. Since the "dishwasher-grasp" is stationary, and there exists multiple grasps to choose from in the set-up at hand, the procedure of collecting data offline and using it online with the robot confirms that a workspace representation could be advantageous.

6.3.6 The work of Haustein et al. (2017)

In the paper "Integrating Motion and Hierarchical Fingertip Grasp Planning" by Haustein et al. (2017), an algorithm that simultaneously searches for high fingertip grasping and a collision-free path to said grasp is presented. They show that their planner can achieve reachable high-

quality fingertip grasps in cluttered scenes. Their methods assumes that the geometry, state and kinematics of the robot is known (they use a 13DOF robot), along with the geometry of the environment the robot operates in as well as the geometry and the pose of the object to grasp.

The framework consists of two main components; one motion-planner, and one grasp-planner for the hand to obtain good grasps with the appropriate contact points on the objects. Both the motion- and grasp-planner share current knowledge of $\mathcal{C}_{free}$. Let $\mathcal{C}_{\mathrm{G}}$ denote the goal region of the grasp, containing the goal configurations. The grasp-planner, goal sampler, for the hand provides to the motion planner configurations that obtain high-quality fingertip grasps for the hand in this region. The motion-planner queries the goal sampler for this information in turn, reliant upon their shared knowledge of $\mathcal{C}_{free}$ (Haustein et al., 2017).

The objective is to allow the motion planner to search for goal configurations from $\mathbf{q}_s$ and find a path in $\mathcal{C}_{free}$, to the goal configurations supplied by the goal sampler. As the motion-plan search continues, trees are constructed, the information on $\mathcal{C}_{free}$ increases, and this aids in guiding the search towards the goal region $\mathcal{C}_{\mathrm{G}}$. The planner they use is bidirectional and attempts to grow two trees, one from $\mathbf{q}_s$ and one from goal configurations supplied by the goal sampler, to find a reachable grasp. To provide such configurations, the grasp planner needs knowledge of $\mathcal{C}_{free}$, which it obtains from the motion planner.

Even though this paper is regarding fingertip grasping and the system at hand employs a suction cup, the methods are interesting in terms of the ability to combine grasp and motion planning into one.

6.3.7 The work of Akinola et al. (2018)

In their paper "Workspace Aware Online Grasp Planning" Akinola et al. (2018) provide a framework for workspace aware online grasp planning by incorporating the notion of reachability into the online grasp planning process. In the offline process, a dense reachability space is constructed for the manipulator and its environment, where for a given sampled pose a check of the existence of an IK solution is done. The reachability space is post-processed to a signed distance field, which includes a points distance from the border of the reachable workspace. The distance field is used in the grasp planning to guide the grasping away from unreachable regions and towards regions of higher reachability, increasing the chance of success.

By utilizing the IK solutions to sampled poses in the workspace, the resultant map is binary in nature; either a grasp is reachable, or it is not. The reachable workspace of a robotic manipulator has a boundary between where the robot can and cannot reach. To incorporate a gradient in their map, Akinola et al. (2018) utilize the distance to this manifold, which they call d_{sdf}, where "sdf" stands for signed distance field, to also rate potential grasp poses on how far they are from this manifold separating the reachable and unreachable workspace. By using this representation and this implementation, reachable grasps obtained a positive "sdf"-value and unreachable grasps had negative "sdf"-values. Once the map is ready, it can be queried for a pose in the workspace and its binary reachability value and the distance from the border of the reachable workspace is returned.

Since the notion of a "good grasp" has no intuition with regards to reachability, the reachability map of Akinola et al. (2018) is used to guide the grasp planning into low energy regions of the space, where it is clear the reachability is good. Since the reachability map has a gradient, it can be optimized in terms of this energy consideration. Akinola et al. (2018) use simulated annealing which is an optimization technique for approximating global optima for functions. By

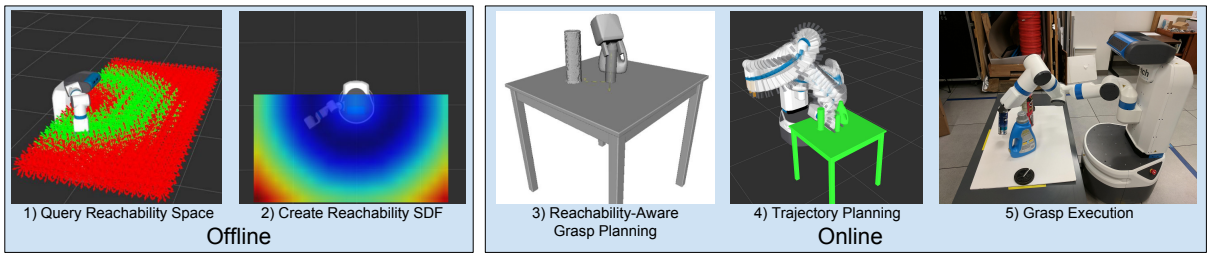


Figure 6.4: Workspace Aware Online Grasping Framework - **Offline**: 1) the robot's reachability space is queried for IK solutions that are free of collisions with the robot itself and static objects such as walls and tables. 2) An SDF is created from the reachability space. **Online**: 3) Grasp planning is quickly accomplished utilizing and reachability space SDF. 4) A motion plan for one of the planned grasps. 5) Trajectory executed by the robot for stable grasp (Akinola et al., 2018). © 2018 IEEE

using this technique they ensure that sampling for grasps is only done in reachable regions of the workspace.

Viewing Figure 6.4, the structure of the framework of Akinola et al. (2018) can be seen. In the online phase of the process, grasp planning is done by using the reachability signed distance field. When grasps are found, planning a motion to the grasp is done and executed by the robot.

Akinola et al. (2018) state that a grasp is reachable if a motion plan can be found to move the arm from its current configuration to a goal configuration that places the end-effector at a desired pose. Yet, only the existence of an IK solution is incorporated in the reachability map and paths are planned at a later stage in the online phase. Since the grasps in the system at hand are planned independently of the reachability, grasp planning will not be incorporated in this work, since we assume the grasps from the network to be valid. The most relevant part of this work is the construction of the reachability space and signed distance field, and how this may be incorporated as an additional step in the system at hand.

6.3.8 Remarks

In conclusion, the works of Zacharias et al. (2007), Vahrenkamp et al. (2013) and Akinola et al. (2018) supply inspiration in terms of using a map to display the robot abilities in the workspace it operates in. A workspace representation containing the robot capabilities in terms of accessibility and reachability would be an implicit way to categorize the workspace enabling the inclusion of the robot in the grasp selection process. With the use of a map, one is also free to populate it with the characteristics important for a particular application. Taking the set-up at hand as an example, the existence of paths to different parts of the configuration space was of foremost importance, whilst for example in the work of Vahrenkamp et al. (2013) configurations some distance away from singularities were deemed important for their application. As will be demonstrated in the following chapter, the use of such capability maps is a decisive part of the work.

Furthermore, Berenson et al. (2007) and Vahrenkamp et al. (2010) use a grasp-scoring function and a grasp-quality measure respectively to rate the grasps in their environment in terms of robot abilities. This aspect of the combination of grasp planning and motion planning also served its purpose as inspiration. As will become apparent in the following chapters, the combination of a type of capability map, and the use of a cost function to evaluate the grasps to be reached has been implemented. Inspired by these works, the implementation of this type of function became

an implicit part of the map data structure, where a query into the map returns grasp quality information in terms of the existing paths to the queried area.

Chapter 7

Method

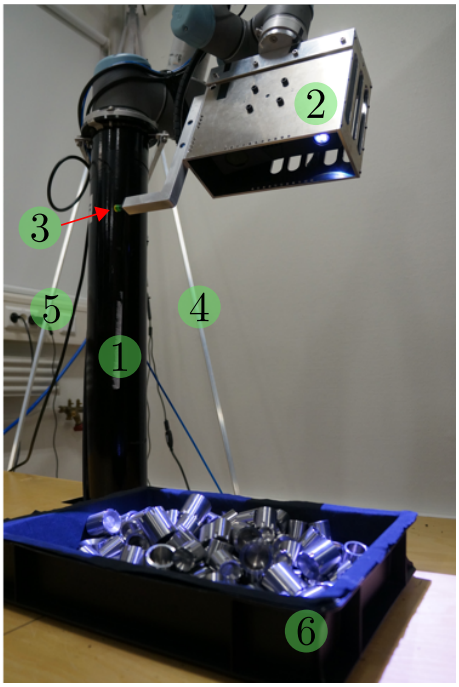


Figure 7.1: Photo of the bin-picking set-up, courtesy of Katrine Seel, MSc SINTEF Digital. Notice the pedestal (1), camera housing (2), suction cup (3), fastening mechanism (4)(5), and bin with reflective steel parts (6).

This chapter will document the methods used and present some of the results obtained. Results are presented in conjunction with the method to highlight choices made and conclusions reached to bring the work forward. This chapter begins with an introduction to the system set-up specifically with grasp selection in mind. The chapter continues with a reiteration of previous work done on the set-up at hand, also with grasp selection in mind. The results obtained during the specialization project will be presented and serves as motivation for mapping the workspace of the robot. The workspace mapping will then be presented. This section includes the methods used for mapping, which traits are mapped, and other issues encountered during the work. The section concludes with deciding upon a new bin placement to replace the current. Following this bin placement, a new mapping is undertaken. The new bin area is presented, discretized and sampled, before several planners and optimization objectives are used to map the area. Several metrics used for comparing planners are also presented. Following this testing, the most appropriate planner is decided upon.

The motivation behind mapping the workspace, replacing the bin, and mapping the new bin area, is to be able to use this map of the robot's path reachability in conjunction with the output from the neural network. The use of this path reachability map will be outlined, and an algorithm demonstrating its use will be presented. This chapter concludes with a case study, where real grasps from the neural network are used to present functionality of the algorithm, and it will be shown that incorporating the robot's abilities when choosing a grasp increases picking success.

7.1 System set-up

The system at hand is a bin-picking loop with all the belonging traits; pose estimation, a 3D sensor, a robotic manipulator arm, a robot scene, a workspace, the need for path planning etc. For this set-up, a dual-resolution convolutional neural network trained on simulated data is used to supply the grasps (Dyrstad et al., 2018). Another popular approach is to utilize grasp planners for the same purpose, such as GraspIt! (Miller and Allen, 2004), OpenRAVE (Diankov and Kuffner, 2008) and OpenGrasp (León et al., 2010).

The bin-picking system set-up used comprises a UR5 robotic manipulator arm, a Zivid 3D camera and a vacuum gripper to pick reflective parts from a bin. The vacuum gripper is what is known as the tool point, tp for short. To supply a grasp, it is important to obtain sufficient depth information from the images of the distribution of parts in the bin, see Figure 7.1. An eye-in-hand configuration provides flexibility in this regard. Information from the camera is used to compute multiple grasps based on how objects are placed in the bin. When the sensor is attached to the robotic arm performing the grasping, additional constraints on how the manipulator can move whilst avoiding self-collisions and collisions with the bin or other parts of the environment are imposed. A characteristic of the grasps supplied by the network (Dyrstad et al., 2018), is that they are decoupled from the robot tasked with reaching them. The network has no knowledge about the existence, and kinematics, of the robot. This raises the issue of reachability, and the need for coupling these two aspects; optimal grasp generation in terms of the object geometry, and prioritizing grasps that are reachable for the robot.

The issue at hand is determining with what amount of ease the robot can reach a specific pose in the workspace, and to find a way to judge which grasps are favourable for reaching with the robotic manipulator arm. The following sections detail the results obtained attempting a solution to this problem. The results were obtained in simulation, using a simulator supplied by Universal Robotics (Universal Robots A/S, 2019), and upon it, working with the Robot Operating System (ROS) workspace structure for the physical system.

The solution flow chart attempted in this work can be seen in Figure 7.2, where each of the yellow boxes will be detailed, culminating in a study of the functionality. The yellow boxes also indicate the main contributions made in this thesis, and is an extension of the flow chart presented in Chapter 1, Figure 1.3.

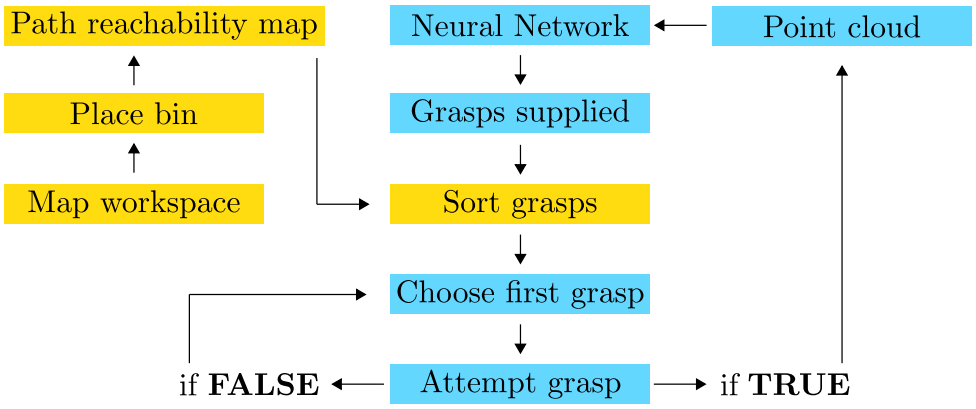


Figure 7.2: Illustration of solution workflow. The flow chart depicts the steps taken to find a solution to the task at hand. First, a mapping of the workspace will be done in order to place the bin optimally in terms of most grasps reached. Once the bin is placed based on the mapping a *path reachability map* will be made, before this map will be used to sort the grasps coming from the neural network, maintaining the modularity of the system as a whole.

7.1.1 Supplying a grasp

The current placement of the bin is based on the optimal range of the Zivid 3D camera, which is 50-60cm from the objects. This 3D camera is placed within a camera housing of substantial size (marked 2 in Figure 7.1) which further limits the arm configuration space. Due to this demand, the UR5 was placed upon a pedestal.

The grasps chosen for picking in this set-up are supplied by a dual-resolution convolutional neural network trained on simulated data (Dyrstad et al., 2018). The input to the network is a point cloud of the current distribution of parts in the bin, and the output is multiple grasp pairs, $\{\mathbf{d}_i, \mathbf{v}_i\}$, where $i \in \{1, \ldots, N\}$, $N \in \mathbb{N}$, $\mathbf{d}_i \in \mathbb{R}^3$ is a point and $\mathbf{v}_i \in \mathbb{R}^3$ is an approach vector. The network supplies four lists of grasp pairs, one for each quadrant in the bin. A grasp is said to be valid if there are no local collisions with other objects in the bin. The output pairs from the network are ordered based on their closeness to the world z-axis pointing up through the pedestal, and in the direction of the camera frame. The motivation behind focusing solely on grasp planning in the neural network, is that it can be robot agnostic.

Since the output of the neural network is a point and an approach vector only, a change in joint configurations whilst keeping the TCP stationary at the point, may lead to multiple viable solutions. As a result of this characteristic, several coordinate frames were sampled with the origin at the same point, but with different orientations, as will be motivated further in this chapter. It is worth noting again, that since additional constraints on the system in terms of the pedestal and the camera housing were present, the need for finding a collision-free path was decisive.

7.1.2 End-effector geometric constraints

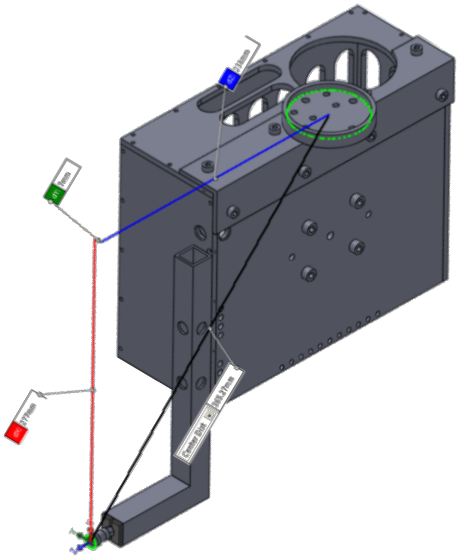


Figure 7.3: Model of the camera housing geometry. The green circle on top of the housing is the point of attachment for the robot end-effector. The L-shaped object has the suction cup attached to the end of it, with the structure also housing the pressurized air supply.

The robot operates under different types of constraints, where the focus of this work is on the end-effector geometric constraints; the camera housing and attached gripper. Differential constraints are constraints placed on the kinematics of the robot, for example limitations on $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$, limiting the movement of the robot due to constraints on its actuators. Geometric constraints however, deal more with the geometry of objects and how they influence the pattern of movement permitted. The constraints can refer to geometric parts in the environment such as static obstacles, or objects attached to the robot.

Let the task of moving from $\mathbf{q}_s$ to $\mathbf{q}_f$ be an optimization problem without constraints other than the physical limitations on the system. When this path is planned, the only need for collision-checking arises when there is a risk of self-collision or saturation in the joints. Let it now be a constrained optimization problem, where the camera housing, gripper and pedestal are included. Collision-checking is now not only needed to check for self-collisions in the joints, but also checking if the geometry of the

camera housing interferes with either the pedestal or the robot links. Both the pedestal and the camera housing with gripper attachment is part of the URDF-file (file describing the robot) used when working with the set-up. The geometric constraints imposed on the system are thus taken into account when planning for a path, which is a decisive part of this work.

7.2 Previous work on grasp selection on the SINTEF set-up

†Using the visualization tool RViz, a 3D visualization tool for ROS, the set-up can be visualized. Figure 7.4 is a screenshot from one such visualization run upon a simulated UR5 robot true to the system at hand

This thesis furthers the work done during the specialization project within the same thematic boundaries; grasp selection in bin-picking tasks for robotic manipulator arm with end-effector geometric constraints. In the following paragraphs, a summary of the results obtained during the specialization project, done during the autoumn of 2018, will be given, along with a detailed description of the experiment executed.

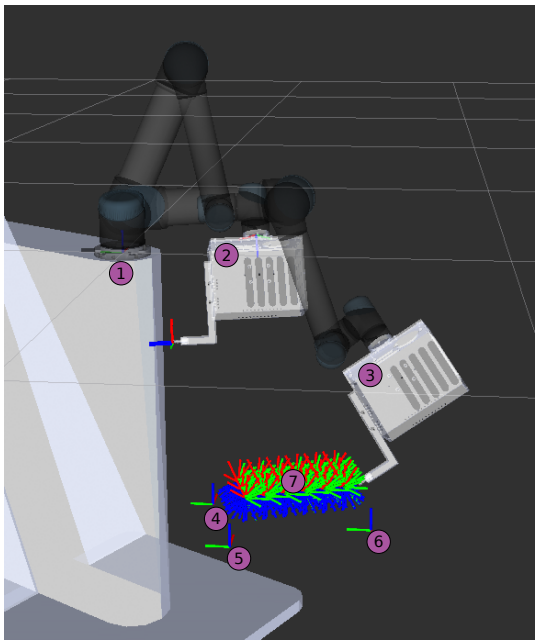


Figure 7.4: Visualization of the specialization project experiment as seen in rviz. 1) pedestal where the robot is placed, 2) robot pose at the scan configuration and Zivid 3D camera, 3) visualization of the path from the scan configuration to an experimental grasp, 4,5,6) actual placement of the bin corners in the physical setup, 7) experimentally generated grasps for investigation during the specialization project

The project was centered around the notion of reachability and path-existence, and how easy or difficult it was, and is, to reach a certain point in the workspace of a robot. This was done to look into how easy it is to reach a grasp deemed optimal by the neural network, since this network does not take into consideration that a robot, with constraints on its links and actuators, must be able to reach it. The work consisted of investigating a "patch" of the workspace, meant to represent the current placement of the bin.

The project is best described by observing Figure 7.4, where everything is labeled. Firstly, the robot is stood upon a pedestal (indicated by the number 1) in a robot cell and consists of a UR5 robot with a specialized camera housing, containing a Zivid 3D camera (2), and a vacuum gripper (coordinate system at the end of L-shaped object attached to camera housing). The robot pose at this instance, as indicated by the same number (2) on the figure, is currently one of four scan configurations. The scan configuration is the pose the robot scans the bin from with the 3D camera, to select an appropriate object to grasp among the many objects in the bin.

Imagining a compass when looking at Figure 7.4, this is the "north" scan configuration. South, west and east configurations also exist. The objective of having multiple permanent scan configuration like this is flexibility in finding valid grasps, and the freedom to capture images from

†Reiteration of the work done in the specialization project (Gravdahl, 2018)

several angles. For example, the robot may scan in a particular pattern; first from the north configuration, then from the east, south and finally the west configuration. This also means that it is from this pose (or these poses) one must plan a path to the given grasp. Since the 3D camera is attached to the end-effector as opposed to having it placed stationary in a different location within the cell, it is necessary for the robot to withdraw from the bin between each picking operation. This is due to the need to reevaluate the new picking opportunities by capturing a new 3D image which in turn is sent to the neural network which gives a new optimal object to pick given the current distribution of objects in the bin. It is highly likely that the distribution has changed compared to the last image captured.

The bin is placed upon a table beneath the robot (this is not visible in the visualization). In connection with a different experiment, the vacuum gripper was placed in the inner corners of the bin and the corresponding pose was saved, such that the robot pose in the four corners given in the base coordinate system was available and known. In Figure 7.4 three of these bin corners, visualized with coordinate systems, are visible and indicated with the numbers (4), (5) and (6). Using these four corners, the "patch" (7) was generated by systematically sampling points and generating different orientations in the points. Each of the coordinate systems shown in the "patch"(7) is meant to represent a grasp from the neural network that the robot must reach, $\mathbf{T}^{\text{base}}_{\text{grip}_i}$. A sampling pattern of 3×6 points equidistantly apart centered in the bin was decided upon for this initial mapping; this gives 18 sampled points in total.

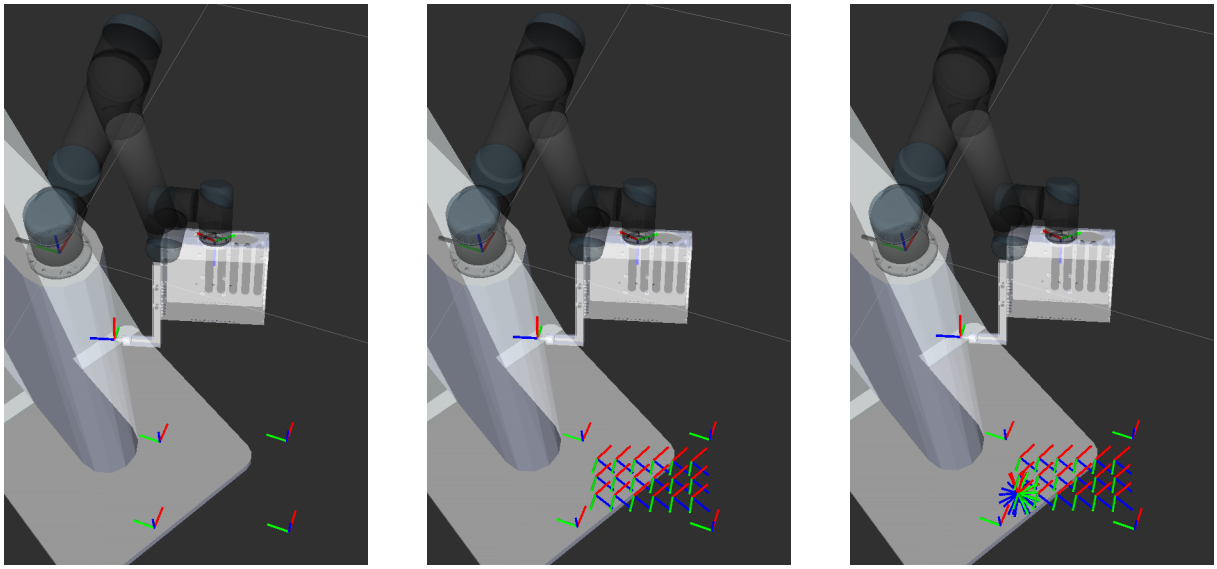


Figure 7.5: Generation of optimal grasps for testing during the specialization project. In the left photo, the bin corner coordinate systems are published to RViz. In the middle photo, the same coordinate system is generated at equally distanced x- and y-coordinates. In the right photo, each of the coordinate systems in the middle photo are present, whilst a full "bundle" of coordinate systems are seen in the the down left corner. In this point there are 27 coordinate systems with coinciding origins.

Viewing Figure 7.5 and Algorithm 1, the procedure for generating test grasps, and results on how easy it is to reach a grasp, is outlined. Firstly in the leftmost image, the bin corners are published to RViz. With each iteration of the inner for-loop in Algorithm 1, 18 coordinate systems are sequentially visualized with the same orientation. After the first 18 are published, the next orientation in generated, before this as well is published to the scene, at every point. When looking at the rightmost image in the figure, the final visual of one of the points is shown. When the outer for-loop in Algorithm 1 concludes, there are 18 points in the area of the bin, with 27 orientations with coinciding origins in each of the points. To conclude, there are 27

Algorithm 1: Algorithm for experiment done in the specialization project

```
for every set combination of x- and y-coordinates do
    for all specified rotations of the generated coordinate system do
        generate T^base_grip_i;
        if there exists an inverse kinematics solution then
            check for a valid path using MoveIt and the joint angles from the IK solution;
        else
            go to next coordinate system;
        end
    end
    save information for post-processing;
end
```

different z-directions where it is possible to approach with the vacuum gripper, to be able to look into both reachability and path reachability of the points.

At each point in the xy-plane (18 of them), and for each orientation (27 in each point), the objective was to align the coordinate system seen at the vacuum gripper, to the coordinate system generated in the "patch". The code first checked for an inverse kinematics solution, and if it found one, checked if there also existed a valid path for the robot to take to this coordinate system by manipulating its joint angles $q_i, i \in [1, 6]$. To summarize, the objective for each of the coordinate systems was to obtain:

$$\mathbf{T}_{tp}^{base} = \mathbf{T}_{grip_i}^{base}, \tag{7.1}$$

where $\mathbf{T}_{tp}^{base}$ is the pose of the tool point given in base coordinates at the scan configuration, and $\mathbf{T}_{grip_i}^{base}$ is the grasp given in the same base coordinate system.

For each coordinate system, the transformation matrix given in the base, $\mathbf{T}_{grip_i}^{base}$ was saved, along with a Boolean value for whether or not there existed an IK solution. If a solution was found, the corresponding joint angles needed to reach his pose were saved and passed along to the built in path planner obtained through the plug-in MoveIt, which looked for a valid path. MoveIt bases its planning on the description of the robot within the simulator, meaning it also takes into account the geometry of the camera house and any possibility of self-collisions or collisions with environmental obstacles.

Lines of the output table which was used for post-processing, had the following structure depending on whether there was a path or not;

$\mathbf{T}_{grip}^{base}$	IK?	q_i
$\mathbf{T}_{grip\ 1}^{base}$;	True ;	$(q_1, q_2, q_3, q_4, q_5, q_6)$
$\mathbf{T}_{grip\ 2}^{base}$;	False ;	None

After the experiments were done, post-processing was necessary to visualize the obtained results. The results were saved in a a large table containing the information for each of the 27 rotations in the 18 points, 486 lines in total. This meant that each 18th line corresponded to information regarding the same point. The data was sorted and heatmaps were used to present the data. To also be able to separate inverse kinematics solution and path success results from each other, separate heatmaps were created. Heatmap presentation was chosen as an intuitive way to visualize the results, also inspired by the presentation design of Zacharias et al. (2007).

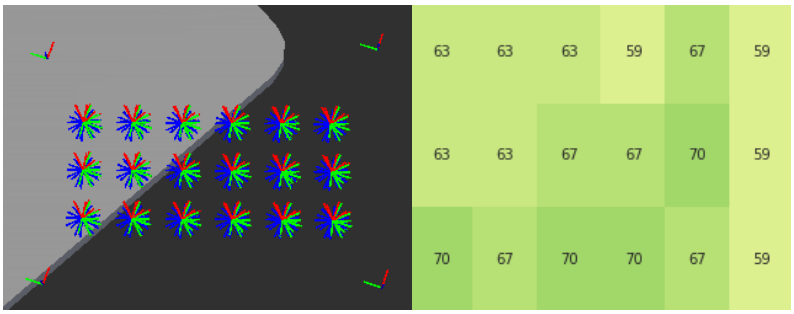


Figure 7.6: On the left: screenshot from RViz taken of the patch under investigation from above. On the right, an example of a resultant heatmap. The numbers within each of the slightly different coloured rectangles represents the percentage of IK solutions found at this point. As such, if all 27 poses in 1 of the 18 points had a valid inverse kinematic solution, the corresponding rectangle would contain the number 100 and be bright green.

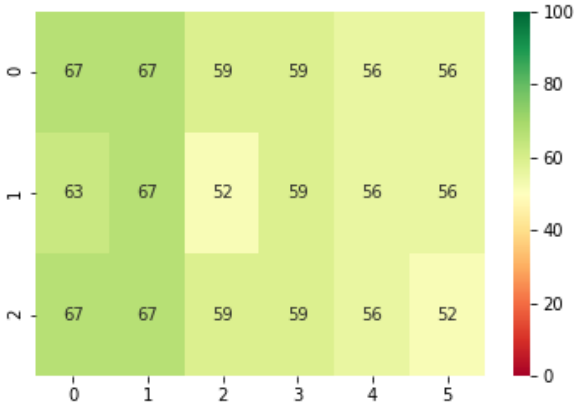
Two types of experiments were executed where the only difference was the seed to the inverse kinematic solver. The two seeds were; the joint angles at the scan configuration, and, the joint angles of the first obtained inverse kinematic solution. The seed of an IK solver is in essence the initial conditions of the search process to find a solution. The closer to the actual solution the seed is, the more likely a solution is to be found (Zacharias et al., 2007). The objective behind looking into two different seeds was to investigate this effect on the results.

The result heatmaps are 3×6 in size, the same dimension as the number of points investigated, as can be seen in Figure 7.6. The results are presented in the form of four heatmaps; one for the coverage of inverse kinematic solutions with the scan configuration joint angles as the seed and one for the path coverage with the same seed, one with the first obtained joint solutions as seed, and one with its corresponding path coverage. Note that these results are twofold; first an IK solution was searched for, and after that, a path to these valid joint angles was attempted.

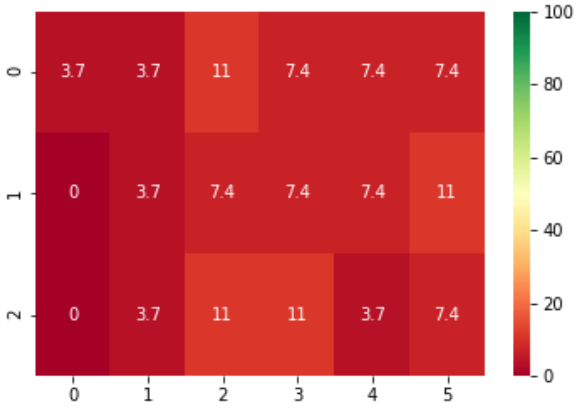
The results are also summarized inTable 7.1. From this table along with the result heatmaps, the best seed to use for the inverse kinematics solver is a set of joint angles close to where we expect the solution to be. Considering the seed is the initial conditions for the search for a solution, this conclusion is in line with expectation. Since multiple inverse kinematic solutions are possible, the fact that there is a difference in path reachability is due to the different joint angles found to reach the same pose, due to the different seeds. A conclusion which can be drawn from this albeit limited data set, is that the further away from the solution the seed is, the worse the path availability becomes. Combining the two traits, IK and path coverage, by a simple AND-operation, gives the final path reachability of the current bin area. This result in total is naturally constrained by the path existence as this is poorer than the inverse kinematic solution coverage.

Table 7.1: Overview of averaged results for both IK and path coverage, for both seeds.

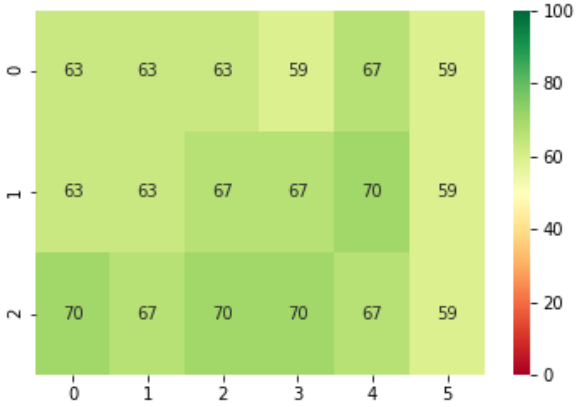
	Scan config. as seed	*First joints as seed*	*Difference*
$\overline{IK}$	59.67%	64.81%	5.14%
$\overline{Path}$	6.38%	26.95%	20.57%
$\overline{IK + Path}$	6.38%	26.95%	20.57%



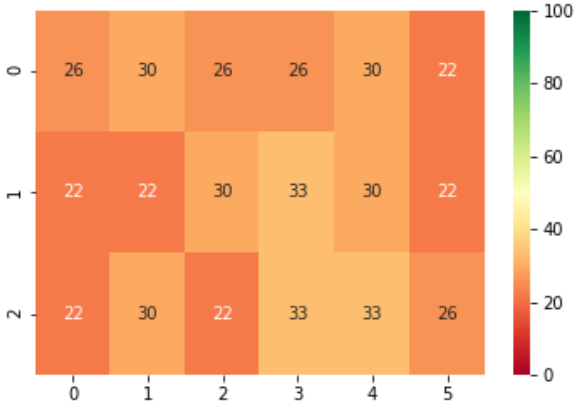
(a) Percentage coverage of inverse kinematic solutions, using the joint angles of the scan configuration as seed to the solver



(b) Percentage coverage of availability of path availability using the resultant joint angles from the inverse kinematic solutions found



(c) Percentage coverage of inverse kinematic solutions in the experimental bin, using the output joint values from the first grasp that gave a solution as seed



(d) Percentage coverage of availability of path availability using the resultant joint angles from the inverse kinematic solutions found

Figure 7.7: Top row: Results of inverse kinematic coverage and availability of paths to these solutions, using the scan configuration joint values as seed to the solver. The numbers in the squares are percentage values. Bottom row: Results of inverse kinematic coverage and availability of paths to these solutions, using the first valid joint angles a path could be planned to as seed to the solver. The numbers in the squares are percentage values.

7.3 Mapping the robot workspace

With the results from the specialization project in mind, see Figure 7.7a, b, c and d, the placement of the bin in the workspace seems sub-optimal. Considering the best final results with regards to path reachability in Figure 7.7d, where the number of valid paths is shown in percentage, there is significant potential for improvement. There was interest in continuing this work of looking into path reachability and with what ease a robot can reach a grasp, and finding a better placement of the bin in the workspace to increase success percentage. Furthermore, after a better, or optimal, placement in terms of maximizing grasp success was found, different planners and planning objectives from OMPL implemented through MoveIt were of interest to look into. Finally, after having found optimal placement, a best possible planner and an objective, it was to be investigated if the procedures found here could be used to filter the

output grasps from the neural network efficiently. If an additional check on the output from the neural network could be done to tie it together with the robot and its kinematics, this would be advantageous as to increase picking success.

The initial work was to find an optimal placement of the bin, to decide on the best part of the workspace to test different planners and metrics in. It was of course possible to trawl the whole workspace and test planners and reachability metrics, but finding a smaller space to do experiments in would assist in increasing the resolution of the results. In addition, offline planning such as when using a simulator is limited in computational time, mostly by the necessary planning time the program demands to look for valid paths to grasps. The first phase of the experimental procedure was to trawl the reachable workspace only looking at valid paths and find a single level in the xy-plane to do specific further work in.

7.3.1 Expanding the volume of interest

Based on the work done during the specialization project, where only a small part of the workspace was looked into, a natural continuation of this work was to investigate a larger part of it. There were two main objectives for expanding the volume. The first reason to do so was to optimize the placement of the bin in the most path reachable workspace of the robot. Previously, its placement had been based the optimal range of the Zivid 3D camera, which has an optimal range of 50-60cm from the objects it is to capture images of. To find a true optimal placement, a thorough testing procedure was necessary. To improve and streamline the picking process with a presumably better success rate when comparing to the current placement of the bin, a larger part of the workspace had to be analyzed. Secondly, by optimizing placement of the bin, specific testing in this area with a higher resolution would be possible. It was expedient to look into optimization metrics after having found an optimal placement of the bin, for an increased degree of accuracy in terms of how easy it is to reach a given grasp.

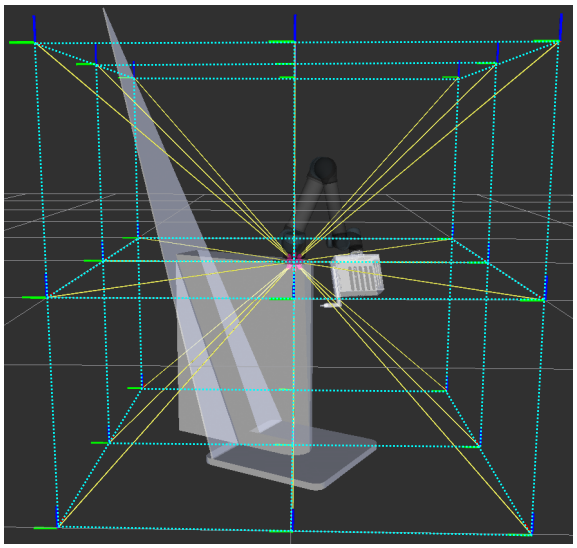


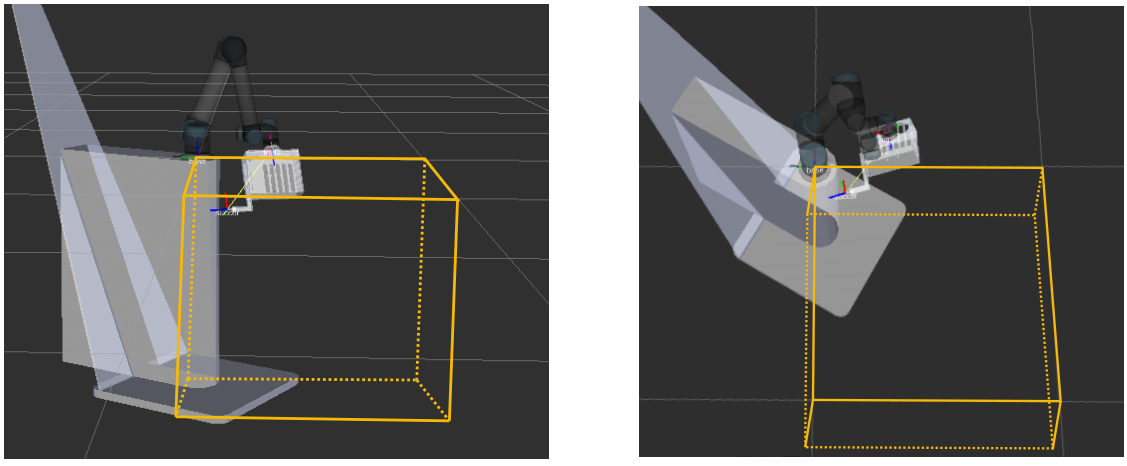
Figure 7.8: Visualization of the eight closest voxels to the robot base constructed by expanding a unit of 1 in each direction. The outline of these are marked in cyan. Also note the coordinate systems defining the corners

Since a visualization of the set-up was available in RViz, it was natural to use this also when looking at a larger volume. A feature of RViz is a voxel, a 3D pixel, grid assisting in viewing the 3D-space in which the robot operates. This grid can be seen in figures taken from the visualization tool RViz, for example in Figure 7.8. It has been defined that the base of the robot is the reference frame for kinematics and planning. With the base as the centre, RViz spans a cubed volume about this point divided into voxels around the robot set-up. Each cube has a width, height and length of 1, which is a non-dimensional unit of the visualization tool where the size is adjustable in the interface. As a starting point, it was advantageous to utilize this grid. The transformation of all corners given in reference to the base in the first eight voxels was known, and can be seen in Figure 7.8.

The initial task of this work was to find the level or height in the workspace at which the reach of the robot was the greatest, constrained by the geometry of the workspace and itself. Consider for a moment, that the UR5 at hand did not have end-effector constraints in terms of the camera housing and attached vacuum gripper, but that the end-effector was coinciding with the tool centre point, $\mathbf{T}_{base}^{ee} = \mathbf{T}_{base}^{tp}$. Then, the task of finding the most reachable part of the workspace could be solved by looking at the already known workspace geometry, as can be seen in Figure 3.1, and choosing the height at which the radius of the sphere was the greatest. If this indeed had been the case, any self-collision would be available through the constraints put on θ_i, where θ denotes a revolute joint in the kinematic chain, and $i \in [1, N]$, N being the number of joints. However, due to the additional imposed limitations on the system supplied by the size and geometry of the camera housing, where the greatest impact is an added chance of self-collision during movement, a different approach than only looking at the workspace geometry needed to be undertaken.

7.3.2 Limiting the workspace

How much of the workspace needed to be analyzed, how much is too little, and how much is too much? As can be seen in the figures from the visualization, as well as pictures of the robot set-up, a large portion of the robot cell, the space surrounding the robot, is obstructed by the pedestal upon which it stands, and the physical fastening mechanism for a Kinect motion sensor. This sensor will not be discussed further as it is outside the scope of this work, only its influence on the workspace will be focused on. The fastening mechanism can be seen as the start of a grey triangle in Figure 7.9a to the left of the robot, and is physical. By this logic, it was assumed that this part of the workspace would be less reachable than the unobstructed parts opposite it. In addition to this, the robot arm and camera housing has leads running along it for power, and doing work in this particular area obstructs movement of the robot further.



(a) Visualization from the side

(b) Visualization from above

Figure 7.9: Screen grab from Rviz, visualization of the voxel enveloping the current placement of the bin, from two different perspectives, marked in yellow.

The current placement of the bin is enveloped by the volume marked in yellow in figures 7.9a and 7.9b. The volume is one of the eight voxels surrounding the robot. The start of the grey triangle to the left of the robot in both figures is the physical fastening mechanism for the Kinect sensor. The fastening mechanism takes up a large part of the cell and obstructs this area significantly. It was therefore deemed unnecessary to map this part of the workspace seeing

as it would certainly impact the reach of the robot and its ability to plan paths around this structure. Furthermore, we also observe the pedestal the robot base is placed upon. By looking at Figure 7.9a, the right bottom volume marked in yellow seems to be the cube where the robot is furthest away from the physical fastening of the Kinect sensor. Furthermore, when looking at this figure, it seems that the robot has the most possibility of movement in this volume, and as such the best reachability. Hence, the voxel foremost of interest became this one, marked in both Figure 7.9a and 7.9b from different views.

7.3.2.1 Inverse kinematic coverage

After having decided that the volume seen in figures 7.9a and 7.9b was to be a starting point, it was of interest to quickly check this volume in terms of inverse kinematic coverage, classic reachability. Since planning a path with MoveIt takes longer than checking if there exists an inverse kinematic solution, this was the first hurdle, and a more efficient way to check if the volume decided upon was sufficient.

The corners of the voxel were defined and points and orientations were generated in a $6 \times 6 \times 11$-grid. This 6×6 resolution in the xy-plane was deemed sufficient enough to observe behaviour. A slightly higher resolution in the z-plane was decided upon to observe behaviour change with the height, where increments of 0.1 were done for $z \in [-1, 0]$. At this point, the goal was to find an optimal value along the z-axis for placement of the bin, and as such this had a higher priority than placement in the xy-plane initially. The idea was to find the best z-placement first, and then in that height optimize placement in the xy-plane.

Algorithm 2: Algorithm for checking inverse kinematics availability in the workspace of interest

```
for all defined test grasps do
    generate T^base_grip_i;
    for all coordinate systems do
        call inverse kinematic solver;
        if there exists a solution then
            save True;
        else
            save Falsr;
        end
    end
end
```

It was necessary to see if the RViz-defined voxel was sufficient to use for the experiments of finding optimal placement of the bin. This was done simply, by checking for an inverse kinematic solution for each of the coordinate systems published and analyzing the results. For instance if there were many IK solutions found at the borders of the voxel, this would suggest that the search area needed to be expanded, so as not to overlook a better placement of the bin.

In Algorithm 2, the procedure of checking for an IK solution is outlined. In essence it only conveys that for each of the published coordinate systems, the inverse kinematic solver is called, and if a solution was found, for the script to save the variable **True** to correspond with the pose of the particular test grasp. If no solution was found, **False** was saved. For post-processing, a large table for each of the different z-levels was available. The Boolean variables for whether there was a solution or not were organized, and a 3D-plots was generated, see Figure 7.10.

The 3D plot is oriented such that the robot base is the column to the left. Note that the colour green indicates a high concentration of IK solutions, and that red indicates a low concentration of solutions. The inverse kinematic solver does not take into account the pedestal in the workspace, which is why one can see that solutions were found inside it in the figure. In addition, this is

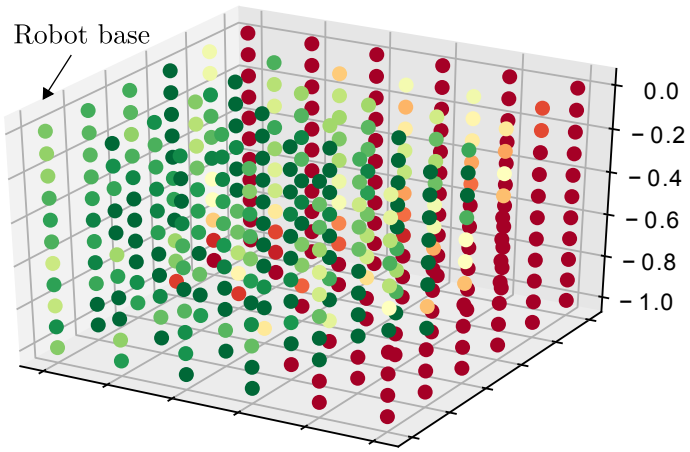


Figure 7.10: Resultant 3D IK map for volume initially of interest

also why investigating if a path exists is of utmost importance to get accurate results for optimal placement.

Again, the existence of IK solutions was checked only to see if the volume of the existing 3D voxel in RViz was large enough to continue the work in. Following the procedure of having used the IK mapping to investigate if the decided upon part of the workspace was large enough, the conclusion became that it was an insufficient volume. This can be seen from the high concentration of solutions on the border to the next voxel. The hypothesis was that the plan coverage would give somewhat similar results to the IK map, but this was not guaranteed. By looking at the 3D plot, there is a higher percentage of solutions inside the cube than at the borders, even though there are solutions found there as well. After having reached the conclusion that this defined volume was not adequate, expansion was necessary.

7.3.3 Expanding into a second voxel

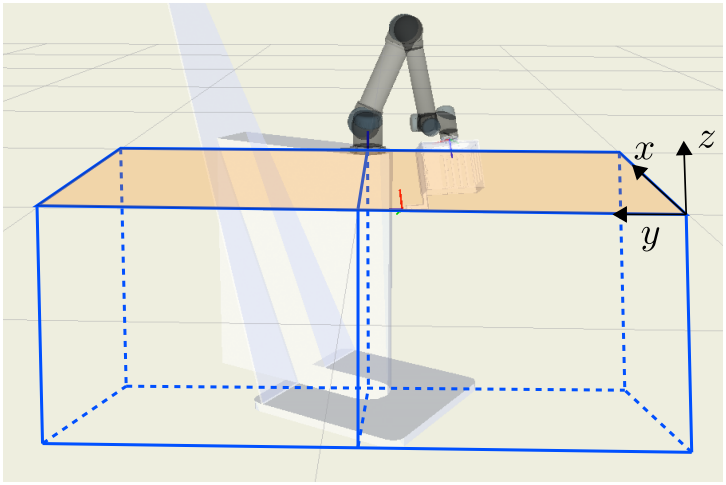


Figure 7.11: The expanded volume of interest. The cube to the right was insufficient due to many IK solution found on the border to the cube to the left. Note that they overlap at the base.

When viewing the results from what was thought to be a sufficient part of the workspace, the voxel previously described, it was indeed not enough to obtain complete results. The IK map was a strong indication of the possibility of finding regions of high path reachability also in the next voxel. The expanded volume can be seen in figure, 7.11 in blue. Here both cubes are outlined. The assumption of limited maneuverability and obstructions around the fastening mechanism for the Kinect sensor still holds, but more of the workspace in front of the robot and to the left had to be investigated. Due to the fastening mechanism, it was hypothesized that the overall path reacha-

bility would be somewhat lower on the left, than on the right. For simplicity, when mapping the space consisting of these two cubes as seen in Figure 7.11, each cube was mapped individually and the total results were combined conclusively.

7.3.4 Non-deterministic planners

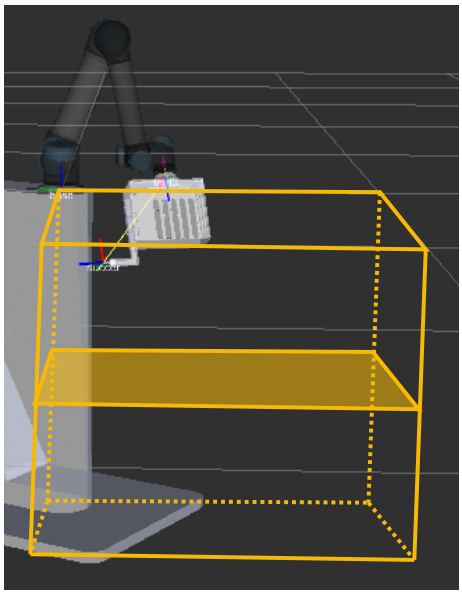


Figure 7.12: Test layer for investigating the non-deterministic behaviour of the planners.

The goal at this stage was to map the path reachability of the a portion of the workspace, the volume outlined in Figure 7.11. Instead of only representing the presence of IK solutions, the existence of collision-free paths were to be included. When starting this work and implementing path planning procedures and testing them, a matter revealed itself which significantly influenced the results. Due to the non-deterministic nature of the planners in OMPL which are sampling-based, see Chapter 5 for background, the planning procedure did not return the same path each time it was run.

Since the planners are random in nature, there is no guarantee that the planning pipeline returns the same result each time even under the exact same conditions. This also includes that we run the risk of the planner saying that there is no path to a specific pose, even if one actually exists. This is due to the random sampling of the workspace when the planner looks for a path. All the planners in OMPL are sampling-based and have this characteristic. Since this presented an interesting behaviour, it was of interest to see how different, and to what order of magnitude, the results could become with a large enough data set.

To look into this non-deterministic behaviour before starting the big-scale data collection on all points, rotations and layers in the volume of interest, the horizontal layer in the middle of the right voxel became the test layer for this phenomenon, see Figure 7.12. The choice of using this layer was made mainly to ensure that a fair amount of feasible points existed, since it was hypothesized that further down in the voxel the coverage would be worse since it is further for the robot to reach.

To look into the random nature, a script checked for a path to all 36 points (6×6-resolution in the xy-plane) for the 27 orientations ($36 \cdot 27 = 972$ test grasps in total). A path was attempted 25 times without any editing between instances. The program had the exact same prerequisites for all tests; the default planner was used, the planner was allowed a 5 second planning time and the optimization objective of the planner was to choose the shortest path lengths. When post-processing the results, the number of found paths was recorded and plotted against its occurrence, the results can be seen in Figure 7.13. With a larger data set it could be speculated that the distribution of the results is Gaussian in nature, but a conclusive results can not be guaranteed here.

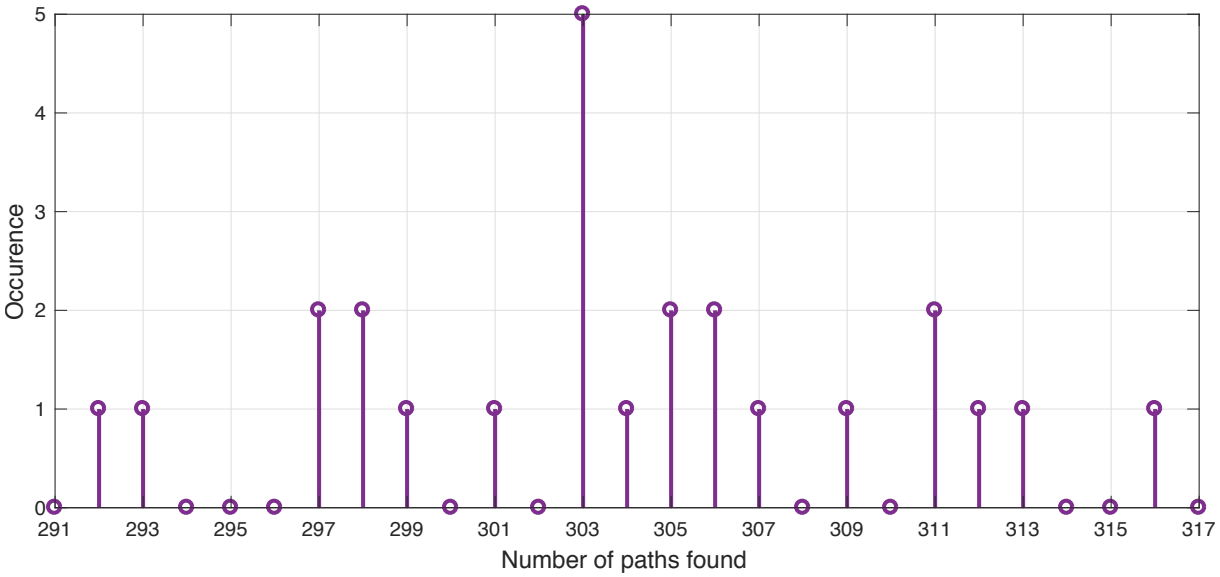


Figure 7.13: Histogram: Number of paths found for an attempt at a path to all 972 coordinate systems, plotted against the occurrence. The planner used was LBKPIECE.

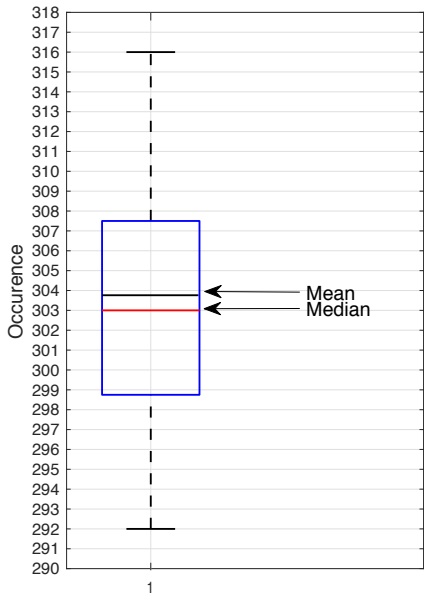


Figure 7.14: Box plot illustrating the behaviour of the behaviour of a non-deterministic planner, LBKPIECE.

A box plot of the data can also be constructed, to look for tendencies of a normal probability distribution. The data is likely normally distributed as the mean almost coincides with the median, and the box plot is roughly symmetric about this value. From Figure 7.14, where both the mean and median is marked, we may deduce that the distribution is close to normal. The mean of the data set is 303.8, and the median is 303. Since the number of paths found needs to be an integer, we may say that the mean and median are "one off". From this plot, we can observe a close to normal distribution, with a slight skew towards lower values. The box itself contains 50% of the information and has a height of about half the span of the data, and this looks to be in line with the observations (Walpole et al., 1998). Even though both Figure 7.13 and 7.14 show tendencies towards a normal distribution, this cannot be said with complete certainty, as extrapolating outside a model is less than ideal, and should be done with care.

As a result of this inquiry, it was decided that when continuing on with mapping the workspace, the procedure of looking for a path was to be executed 5 times to saturate the results. Since, if there at one time is found a path, a path does indeed exist regardless of it not having been found at another instance. Ideally, this number should be much higher, but due to time-constraints, 5 repetitions was deemed sufficient to demonstrate the effect. From the article titled "Performance Study of Single-Query Motion Planning for Grasp Execution Using Various Manipulators" by Meijer et al. (2017) this finding is supported as they state that due to the randomization of sampling-based motion planners, planning problems have to be run multiple times to provide saturated results on the performance (Meijer et al., 2017).

7.3.5 Workspace mapping

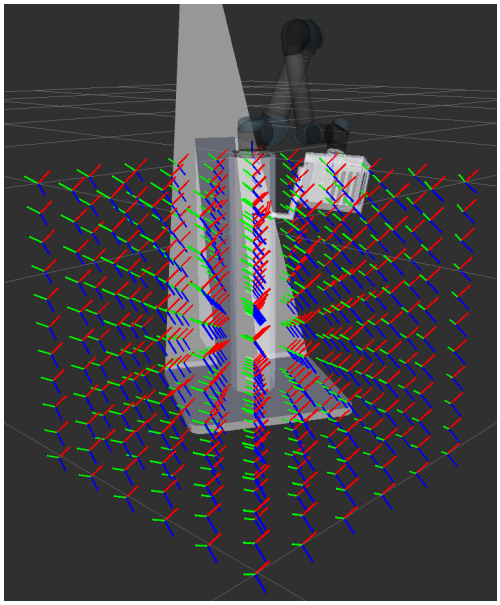


Figure 7.15: Visualization of the right voxel under investigation, with one published coordinate system in each of the defined points

Since optimal placement of the bin was first priority, it was decided that default settings from MoveIt were to be used. The overall goal was to test multiple planners, objectives and metrics in the optimal part of the workspace in terms of path reachability, but for this first iteration on the search for optimality, default would do. The default planner LKB-PIECE1 was used with the shortest path length optimality objective. The planner was allowed 5 seconds to look for a path and the longest valid segment fraction was set to 0.01 which is the number defining the fraction of the configuration space we assume is collision-free when the robot is in a collision-free configuration $\mathbf{q}$, and is used when investigating if a path or configuration is collision-free (MoveIt, 2019). The attempt at a plan was made 5 times.

For finding optimal placement the same 11 layers that were used when checking for an inverse kinematic solution was generated in both cubes. The resolution in the z-direction was again higher than the resolution in the xy-direction, since height was of foremost importance. The algorithm implemented when checking for a valid path can be seen in Algorithm 3.

Looking at Figure 7.15, the points that were checked can be seen, here only seen in the right voxel. The points are here represented with just one coordinate system. In each of these points, 27 different orientation were checked, such that significant exploration of another joint configuration resulting in a solution could reveal itself. This "degree of freedom" is supplied by the fact that the grasps generated from the neural network in the real system is just a point and an approach vector, and not a full coordinate system which can coincide with the gripper coordinate system. As such, a change in robot configuration might supply a successful grip, and had to be tested.

The coordinate systems were published to the scene, and tested for a path systematically. As Algorithm 3 outlines, one coordinate system is created, the transformation matrix of this grasp given in base coordinates is defined as the goal pose for the planner, and if there is a path to this pose which both has an inverse kinematic solution and a collision-free path, the path will be executed. Following the execution of the path, a path back to the scan configuration is found and executed, before the results are saved to file.

7.3.5.1 Mapping procedure: Plan to pose

For the specialization project, outlined in Section 7.2, two metrics were checked consecutively, where both needed to be fulfilled for the grasp to be path reachable. The first metric was that there needed to exist an IK solution. The second demand was that a path could be planned and executed to the joint configuration, the output of the IK solver. The procedure for this is outlined in Algorithm 1. As such, the IK solution was checked first, and if it that was a success, the output joint angles from the solver were passed to the planning pipeline in MoveIt. If also a

Algorithm 3: Algorithm for mapping the workspace in terms of path reachability

```
for variations in height z do
    for every point (x, y) do
        for all defined orientations about (x, y, z) do
            generate T^base_grip_i;
            plan a path to the pose with MoveIt;
            if there exists a path to the specified pose then
                save True;
                execute the path;
                return to scan configuration;
            else
                save False;
                go to next coordinate system;
            end
        end
    end
    save information on path coverage for post-processing;
end
```

path could be found, the grasp was said to be path reachable for the robot. Note again, that the grasp supplied by the neural network, does not take into account that a robot needs to reach it, only that it is the optimal way to pick an object based on the distribution of the parts in the bin at a given time.

In this work however, planning to the grasp pose, and not the joint angles, provided the advantage of checking for both an IK solution and a collision-free path simultaneously, limiting the work done on each coordinate system. This is simply an advantage to MoveIt. The metric used in this first part of the thesis, finding an optimal placement of the bin by workspace mapping, makes use of the same metrics as in the specialization project, but with a slightly different implementation; planning directly to a pose which both checks for an IK solution and a path, instead of looking for one of the IK solutions and planning to the corresponding joint angles.

When planning to a pose it was of great importance to handle the different warnings and error-messages which indicated whether there was a collision-free path to the grasp or not, and ensure that the proper conclusions on the validity was reached by the script. There were three main possible outcomes for a planning instance:

1. An IK solution does not exist: "`Unable to sample any valid states for goal tree`". The default planner which was the planner of choice for the mapping is bi-directional, meaning that it builds a tree from both the start and the goal state and attempts to connect the trees to find a valid traversal of the graph. If there are no valid states found for the goal tree, it is reasonable to assume an IK solution does not exist.
2. The planner did not find a path within the allotted time, but an IK solution existed: "`Fail: ABORTED: No motion plan found. No execution attempted`". This error is the primary reason for the need to saturate the results. An instance where this exception was raised, might be valid the next time it is tried, due to the random sampling of configurations when attempting to find a path by the planner.
3. An IK solution existed, and a path had been found, but it was discovered that the path

contains a collision with either itself or the pedestal: "`Computed path is not valid...`", "`...Found a contact between 'upper_arm_link' (type 'Robot link') and 'camera_house' (type 'Robot link'), which constitutes a collision`"

To briefly summarize, the metric used when mapping the workspace is threefold; an IK solution must exist, a path must be found, and the path must be collision-free. If these demands are all met, the pose is said to be path reachable. The planning procedure, outlined in Algorithm 3, was executed five times to saturate the results (Meijer et al., 2017), without any interference between instances. When all five iterations were done, a table for each iteration was available. This table contains the homogeneous transformation for each grasp given in the base frame along with a Boolean value corresponding to whether or not the potential grasp is path reachable or not. Saturating the results was then as simple as OR-ing the Boolean values for each run. This procedure is outlined in Algorithm 4. If for any of the planning instances a path was found, a path does indeed exist to this pose.

Algorithm 4: Algorithm for saturating the results of the workspace mapping in terms of path reachability.

for *all instances of* $T^{base}_{grip_i}$ **do**
  if *any of* $\mathbf{T}^{base}_{grip_i}$, $i \in 1, \ldots, 5$ *is path reachable* **then**
    $\mathbf{T}^{base}_{grip_i}$ = path reachable
  else
    $\mathbf{T}^{base}_{grip_i}$ = not path reachable
  end
end

The results of the mapping are presented with heatmaps, where one individual square represents a point in the workspace containing the 27 orientations. Each of the squares contains the percentage score of how many of the sampled poses in that particular point were successful. The color scale of the results goes from red to green, where red is low reachability and green is high reachability.

An example of a point result can be seen in Figure 7.16. Out of 27 orientations in this point, 18 of them were reachable by the robot and had a valid path executed to it. As in the work of Zacharias et al. (2007), and their use of reachability indices and reachability spheres, the percentage and colour is calculated by the same formula, seen in Equation (7.2). In the equation, n_{xy} is the total path reachability score in percent for a point represented by a square in the level map, r_{xy} is the number of path reachable grasps in the point, and N_{xy} is the total number of possible grasps in said point.

$$n_{\text{xy}} = \frac{r_{\text{xy}}}{N_{\text{xy}}} \cdot 100, \text{ where } N_{\text{xy}} = 27 \tag{7.2}$$

$$n_{\text{xy}} = \frac{18}{27} \cdot 100 = 66.6667\% \approx 67\%, \tag{7.3}$$

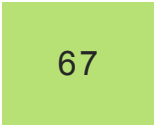


Figure 7.16: Example of path reachability in a point, here represented by a point in the workspace with 67% path reachability

7.3.5.2 Results from right voxel

After having mapped the right voxel, heatmaps like the one in Figure 7.17 were obtained, 11 in total, one for each increment in the z-direction. The heatmap in the figure is the result cross-section at $z = -0.5$ presented for illustration. There are several observations to be made from the heatmaps using this illustration. Firstly, the right edge of the area is relatively unreachable by the robot and gives an overall "red" score on path reachability. This same characteristic was observed with the other 10 heatmaps. This indicates that the volume under investigation was sufficiently large in this direction.

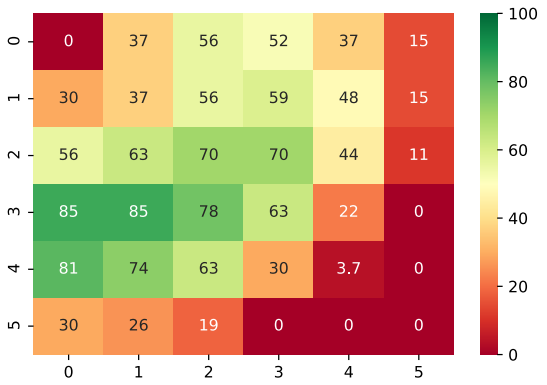


Figure 7.17: Cross section in the xy-plane of the mapped right voxel at $z = -0.5$. The numbers inside the squares shows successful path planning percentage, with the legend to the right of the figure explaining the coloured squares.

Secondly, the right bottom corner of the area creates an unreachable circle segment, indicating that this is the border of the 6DOF spherical manipulator workspace. In the other 10 heatmaps, we observe the same characteristic with varying radii. Note again the illustration of the spherical workspace of the UR5, without a tool, supplied by Universal Robotics, shown in Figure 3.1. This indicates that the border between the reachable and unreachable workspace has been included in the mapping and is sufficient also in this direction. Thirdly, the top left corner, which is the robot base and the pedestal upon which is stands, is completely unreachable by the robot. This is of course because test grasps also were generated inside this pedestal. Note that collision-checking was not part of the procedure when checking for IK solutions alone.

However, the left border of the area in Figure 7.17 shows a high degree of path reachability, a higher degree than initially anticipated. The hypothesis initially was that the centre of this area would be more path reachable then this left border since this area is part of the cube diagonally opposite the fastening mechanism for the Kinect sensor. Fortunately, by the check for IK solutions previously executed, results were also available in the voxel to the left, continuing the map beyond this area of high reachability. The observation of high path reachability on the left border of the heatmap was observed for the remaining 10 heatmaps as well, confirming the need for the expansion of the mapping as dicussed in Section 7.3.3.

7.3.5.3 Results from left voxel

The procedure for obtaining the data describing the left voxel was identical to the one which gave the results for the right, see Algorithm 3. The resultant xy-plane cross section at $z = -0.5$ is shown in Figure 7.18. Note that this is the same level that Figure 7.17 portrays, but for the left side of the volume seen in Figure 7.11. As expected, since this cube is closer to the obstruction that is the Kinect fastening mechanism, the total path reachability is lesser than for the right voxel.

Some of the same observations can be made for the results of this left cube as for the right, and it mirrors it nicely. The left border of the map is substantially less path reachable then the remaining part of the map, and gives also here an overall "red" score. This can be observed for the remaining 10 heatmaps for the left side of the base.

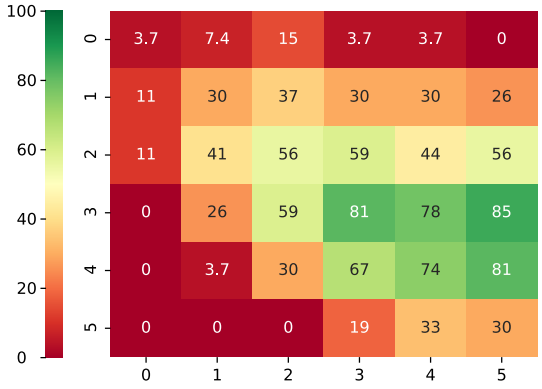


Figure 7.18: Cross section in the xy-plane of the mapped left voxel at $z = -0.5$. The numbers inside the squares shows successful path planning percentage, with the legend to the right of the figure explaining the coloured squares.

Secondly, it is also possible in this cross section to see the boundary of the spherical workspace of the robot in the left bottom corner of the figure. Thirdly, as expected, the highest concentration of reachable poses is found at the right-hand side border. It is also interesting to observe here, the influence from the physical fastening mechanism at the back of the set-up. The greatest difference between figures 7.17 and 7.18 is the top row of points, where the left top row is far less path reachable than the one on the right; these are the test grasps closest to the obstruction.

From the heatmaps in figures 7.17 and 7.18, it is possible to deduce that a sufficient part of the workspace has been covered by the mapping. The remaining work as of this point to obtain a complete visual of the volume mapped is to combine the results for the right and left side, as will be explained in the next section.

7.3.5.4 Resulting plots from combined volume

It was advantageous to implement the mapping procedure separately for each voxel, and then combine the results afterwards. The only change which then had to be made in the implementation was which voxel grid corner was used as the origin for coordinate system generation. Combining the results from both voxels in this manner resulted in an overlap in the results. This is visible in Figure 7.11 where the outlines of the cubes coincide in the middle.

Viewing Figure 7.19, which is a side by side view of the path reachability coverage for the two voxels, cross-sectioned at $z = -0.5$, we observe this overlap. The results for the column at $x = 5$ in Figure 7.19a and the results for the column at $x = 0$ in Figure 7.19b are similar. Viewing Figure 7.19, it is clear that the combined result of these two areas gives a complete image of the path reachability of this part of the workspace. Within this overlap the best results were chosen. Viewing the columns previously mentioned, the only difference between them is the result at $y = 1$.

For the right voxel, it was found that a path to 8 out of the 27 orientations existed, resulting in a total path reachability score of

$$n_{\text{xy}} = \frac{r_{\text{xy}}}{N_{\text{xy}}} \cdot 100, \text{ where } N_{\text{xy}} = 27 \tag{7.4}$$

$$n_{\text{xy}} = \frac{8}{27} \cdot 100 = 29.6 \approx 30\% \tag{7.5}$$

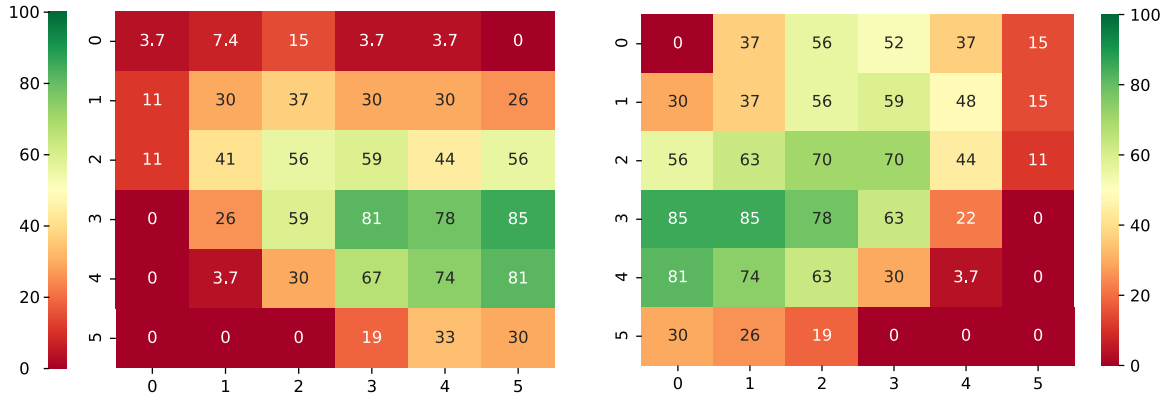
For the left voxel, it was found that a path existed to 7 of the 27 orientations, giving a total path reachability score of

$$n_{\text{xy}} = \frac{r_{\text{xy}}}{N_{\text{xy}}} \cdot 100, \text{ where } N_{\text{xy}} = 27 \tag{7.6}$$

$$n_{\text{xy}} = \frac{7}{27} \cdot 100 = 25.9 \approx 26\% \tag{7.7}$$

However, since the eighth path was found in one of the planning instances, a path to it exists. In the total path reachability map for this level, the value at $y = 1$ becomes 30, and not 26. Recall that if there once was found a path, it does indeed exist.

The results for each xy-level in the z-direction were combined where they overlapped to supply the total path reachability coverage in the workspace of the robot deemed the most accessible. The results for all levels can be seen in Figure 7.20. The visualization of the results in this manner supplied the information needed to place the bin optimally in terms of path reachability in both x, y and z-direction.



(a) Left-hand level at $z = -0.5$

(b) Right-hand level at $z = -0.5$

Figure 7.19: Results from both left and right mapping, illustrating a complete overview of the path reachability at level $z = -0.5$.

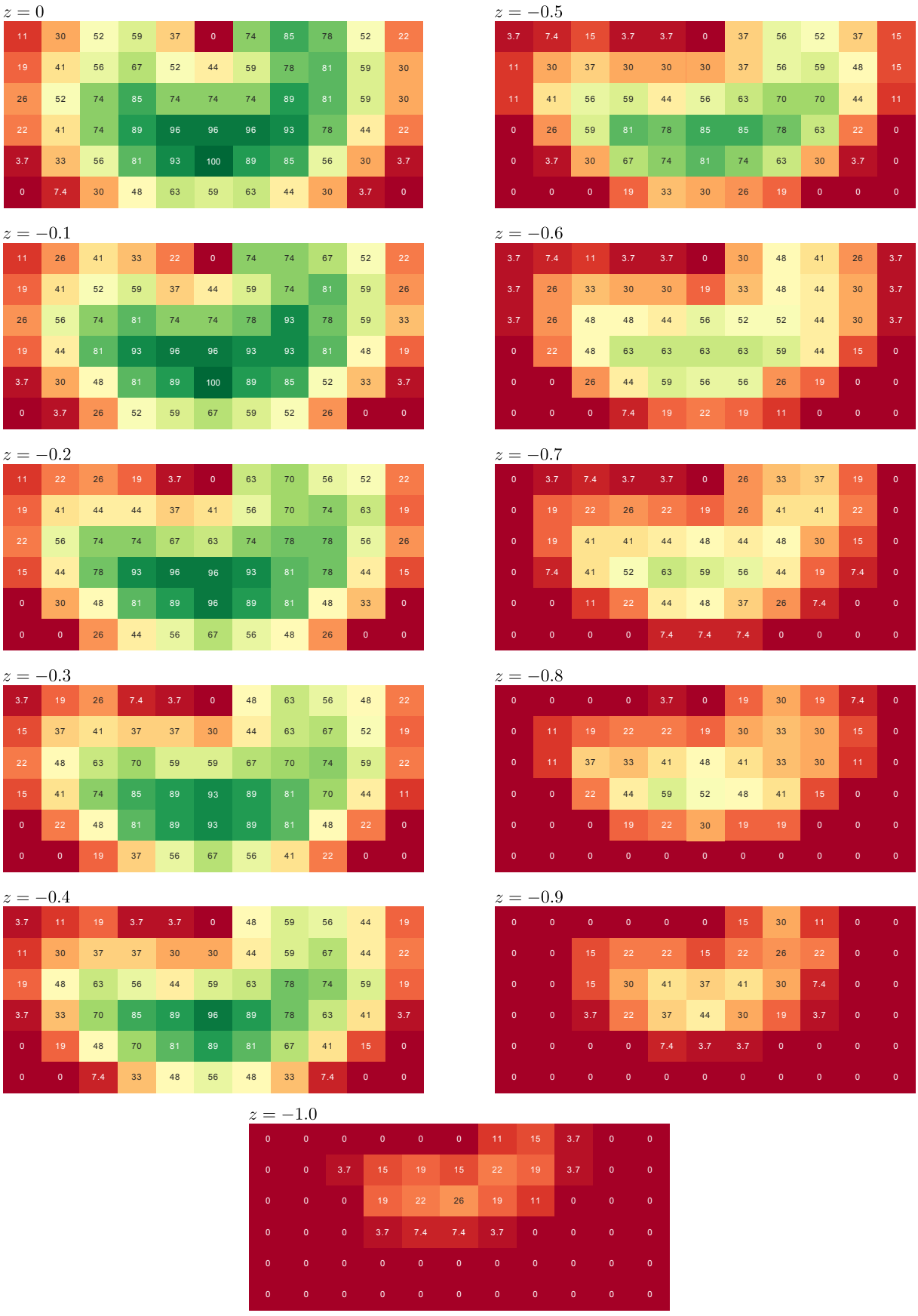


Figure 7.20: Path reachability map: Path reachability of the volume seen in Figure 7.11, split into the cross sections resulting from an increment by 0.1 in the z-direction. The numbers inside the squares are percentage path reachability scores for the particular point, and the color scheme goes from red (low path reachability) to green (high path reachability).

7.4 Bin placement based on mapping

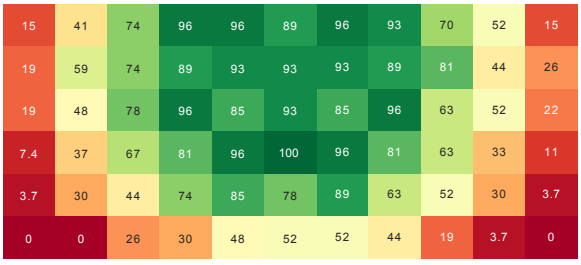
Table 7.2: Average path reachability of both voxels along with the total results. Note that the total is not the average of the left and right voxel due to the overlap.

Level	**Left** [%]	**Right** [%]	**Total** [%]
0	51.23	57.30	53.54
−0.1	48.77	56.79	51.83
−0.2	44.96	52.98	48.06
−0.3	41.15	48.97	44.16
−0.4	37.04	45.78	40.23
−0.5	31.38	39.30	34.34
−0.6	24.69	28.19	25.42
−0.7	18.83	21.30	19.07
−0.8	14.20	16.15	14.47
−0.9	8.74	9.98	8.72
−1.0	3.70	4.32	3.73

With the results shown in Figure 7.20 the information needed to place the bin optimally in terms of path reachability was available. Viewing the results, it is clear that level 0 in Figure 7.20 is the layer with the highest number of successful paths on average. This level is also presented in Figure 7.21a. The heatmaps do not take into account explicit directional preferences of the robot, but is the total results from all approach directions.

This result was not unanticipated as it is the level closest to the centre of the workspace, the base. It is also the region of the theoretical workspace with the largest span. Even though it was likely the best option, due to the imposed constraints of the pedestal and the camera housing, it had to be validated. Table 7.2 summarizes the results for all the mapped levels, and it is the total map at level 0 which gives the best average results. In addition to being the level with the highest average path reachability, the dark green "patch" in the middle was also the region with the highest concentration of successes when comparing this area with the other level maps in the z-direction in Figure 7.20.

(a) Path reachability heatmap seen from above: Optimal level based on average hit rate along with a concentrated area with high reachability. The numbers in the squares represent percentage path reachability for the points and the colors reflect this scale from red (low) to green (high)



(b) Reachability heatmap seen from above: Coverage of IK solutions in the top level of the workspace. The numbers in the squares represent percentage IK solutions for the points and the colors reflect this scale from red (low) to green (high).

Figure 7.21: Comparison of the reachability map and the path reachability map of the robot

The heatmaps produced are representations of the path reachability of the robot in its workspace. The map in Figure 7.21a is the path reachability heatmap, meaning that both an IK solution and a collision-free path must exist to the test poses for the pose to be marked valid. For comparison, the reachability map of the same level is presented in Figure 7.21b, this contains IK solution information exclusively. By comparing the two, one can observe that the path reachability map

is a more conservative estimate of accessible regions in the workspace.

Naturally, obstacles in the workspace changes the reachability characteristics of the workspace and how the workspace looks in terms of kinematic reach and path coverage. According to Zacharias et al. (2009), the region of influence of an obstacle increases, the closer it is to the robot base. By placing the same obstacles in different regions of the workspace, they showed that when the object was placed close to the base, it had more of an impact on the capabilities of the robot. The reason for this behaviour is that the region close to the base is often swept by the robot since it is natural for the robot to move close to the center of its workspace; the base. The bin, which is not visualized in Rviz, can be viewed as an obstacle due to its edges and must be considered when planning a path. Keeping this in mind, it seemed reasonable to place the bin well away from the base whilst still maintaining good coverage.

The metric for placing the bin is as previously mentioned; the highest concentration of total test grasps making up the totals maps that both have an IK solution and a valid collision-free path planned to them, in an area equivalent to that of the bin. The objective of the following sections is to find this area in the level with the highest success rate, $z = 0$, level with the base. In the following sections the bin will be placed on the basis of highest hit average hit rate, and approach directions preferable by the robot will be briefly discussed.

7.4.1 Concerning the sampled points

Seeing the results in level 0, seen in Figure 7.21a, placing the bin such that the point with 100% reachability is covered should be of great interest, and served as the starting point for placement. Since the routine of "drawing" the bin in Rviz starts with defining a corner of the bin and spanning two vectors of the appropriate lengths perpendicular to each other from this point, the starting point became the point to the left of this 100%-point. When spanning the bin from this point in RViz, the optimal point ends up on the edge of the bin, where no grasps will originate. Due to this, the whole bin was moved away from the base until the optimal point with the best hit rate was roughly centered, and in an area where grasps would originate from. The reason the whole bin was moved away from the base was to minimize its impact on possible collisions with the end-effector when grasping. The final placement of the bin can be seen in Figure 7.22, where the heatmap for the level has been layered upon the new bin placement. The reason that the heatmap expands outside the border of the image with the bin, is that the squares in reality are points which were placed on the border of the voxels under investigation.

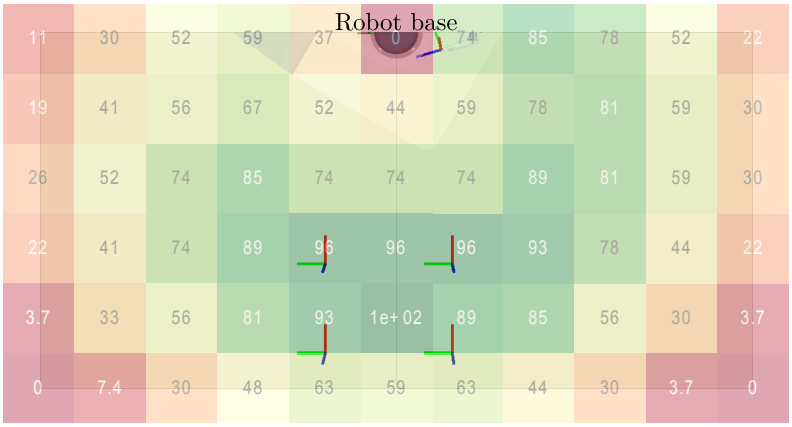
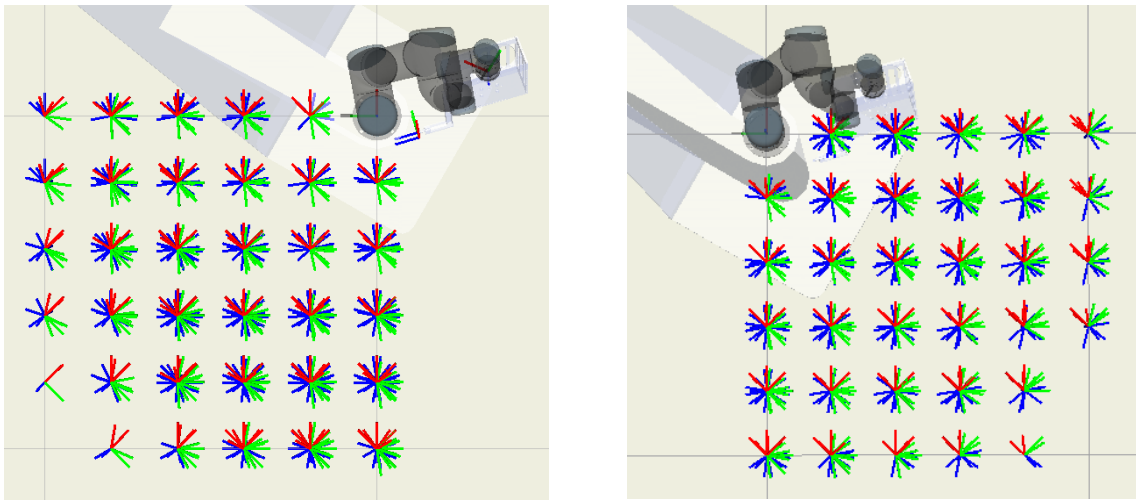


Figure 7.22: Optimal placement for bin. Both placed optimally in z-direction and xy-direction

7.4.2 Concerning the rotations in the points



(a) All valid potential grasp poses in the left voxel (b) All valid potential grasp poses in the left voxel

Figure 7.23: All valid potential grasp poses in both voxels at the optimal level in terms of path reachability $z = 0$. Only the poses with both an IK solution and a valid path to them are visualized

In all 36 points (sampling in a 6×6 grid), 27 different orientations were investigated. Viewing Figure 7.23, the orientations that had a valid path to them can be seen. This figure is the "real" view of the heatmap seen in Figure 7.21a. The heatmaps up until now have conveyed the average path reachability score of the points in the workspace, whilst disregarding which orientations have been more successful than others. Consider for instance a path reachability score of 50%, but all successes being coordinate systems where the z-axis points in the same general direction. Recall, that it is the z-axis of the test grasps that need to coincide with the z_{tp}-axis for a grasp to be successful.

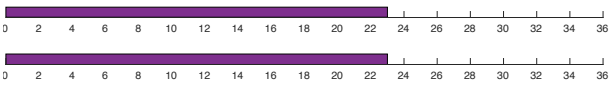
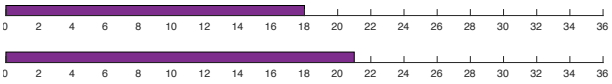
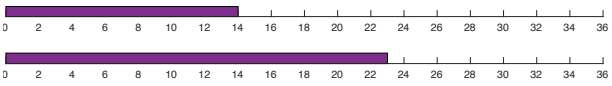
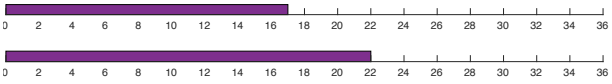
It was therefore of interest to also consider if there were some directions that more often than others led to a valid solution being found. As such, the data concerning each of the orientations, instead of each of the points are considered here. An excerpt of the table containing these rotations and their success with regards to path reachability can be seen in Table 7.3. The complete table is available in the appendix, B.1. In the table, the top bar chart for each entry is the path reachability for the orientation in the right voxel, and the bottom bar chart corresponds to the same in the left voxel.

A summary of the results include, that grasps coming in from above have a tendency to perform well in both voxels. When looking at grasps coming in towards the base from west, these perform well in the right voxel, but have fewer hits in the left. The same goes for grasps coming in from east, towards the base; these perform better in the left voxel when compared to the right. However, there exists some examples of the opposite also holding true, such that this conclusion is somewhat weak, and should be investigated further.

Previously, when the bin was placed as in Figure 1.1, it has been experienced that grasps coming in towards the base and the bin from the left had been problematic due to the potential collision with the pedestal when the robot has to maneuver past this. The hypothesis was that grasps from west are worse than from north, east and south, skewing the optimal placement of the bin into the right voxel. This is partially supported by the findings here, and also supported by the lesser degree of path reachability on the left side of the overlap as opposed to the right

in Figure 7.21a. There is a greater degree of path reachability further away from the base in the left voxel than in the right, leading towards the conclusion that the further away from the pedestal the robot is in the left voxel, the easier it is to maneuver in towards the bin. In the right voxel, characteristics of the opposite can be observed, where there is a larger degree of path reachability closer to the base.

Table 7.3: Overview of approach directions in the optimal level in terms of path reachability. Note that this is only the four first lines of the complete table given in Appendix B.

Grasp	Statistics
	
	
	
	
⋮	⋮

7.5 Optimal level mapping

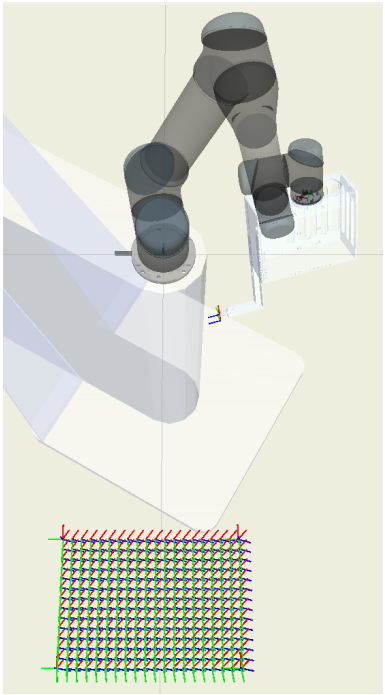


Figure 7.24: New placement of bin in the workspace, with one direction of test grasps shown

Based on the analysis undertaken in the previous section, a new placement of the bin was available, based on path reachability mapping of a large part of the workspace of the robotic manipulator arm with end-effector constraints taken into account. Placement in terms of the highest concentration of path reachability in both x-, y-, and z-direction had been achieved. The new placement of the bin can be seen in Figure 7.24, in relation to the robot. It can also be seen that the optimal placement in terms of path reachability reaches across the two cubes mapped.

With this new placement, several tasks were of interest to complete. First, sampling this area to be able to map it with an appropriate resolution was to be done. The sampling was to be reasonable in terms of the size of the bin and the objects to be picked. Secondly, after an appropriate sampling rate was decided upon, several different planners from OMPL, implemented using MoveIt, were to be tested in this new bin placement to be able to determine which planner and planning objective was the most effective for the set-up at hand. The planners were all to be tested and compared in terms of path reachability as with the workspace mapping, as well as the metrics path length, planning time and execution time of the obtained paths.

Following the data collection on the different planners, the appropriate planner and optimization objective was to be used along with the data collected as a compliment to the grasps supplied by the neural network to incorporate the robot kinematics and its abilities in the grasp selection process. Since the bin and its immediate surroundings is the main region of the workspace the robot will operate in, this can be viewed as a task-specific workspace mapping. The objective was to use known, offline gathered data, to constitute an additional step in the grasp selection process by being able to guide the grasps from the network, $\{\mathbf{d}_i, \mathbf{v}_i\}$, towards easy to reach regions of the bin, whilst maintaining the modular structure of the system. The following paragraphs will deal with the sampling, the planners and the data collection. Chapter 8 will outline the use of the data together with real grasps supplied by the neural network, and present the algorithm used to include the additional path reachability test.

The possible tool point placements used to map the new bin area will be from here on referred to as *test grasps*. A test grasp is a pose with the homogeneous transformation $\mathbf{T}_j^{\text{test}}$, consisting of the rotation matrix of the test grasps given in the base frame, $\mathbf{R}_j^{\text{test}}$, and the position vector of the test grasps in the base frame, $\mathbf{d}_j^{\text{test}}$.

7.5.1 Sampling the bin

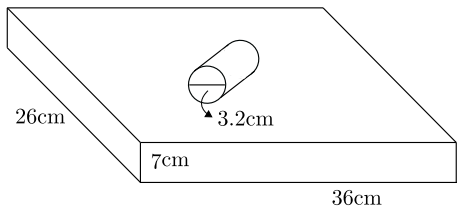


Figure 7.25: Dimensions of the bin and the diameter of the cylindrical objects to be picked

As can be seen in the results from the total workspace mapping in Figure 7.20, the sampling is sparse. This sparse sampling of possible configurations that the robot needed to reach, was done to be able to determine with some degree of accuracy the best level to test different planners and planner objectives in, and where to sample more densely. Now that the optimal level in terms of best path reachability degree had been determined, a different sampling scheme had to be employed.

Considering the bin dimensions and the objects to be picked, illustrated in Figure 7.25, an appropriate sampling rate had to be decided upon. In a worst case situation in terms of surface area available for grasping, all parts would stand upright in the bin. This worst case situation is used as the basis for deciding on sampling rate. The diameter of the cylindrical objects is 3.2cm, and the dimensions of the bin is 26×36cm. Considering all objects standing upright and allowing for some space in between them, it was estimated that 7 objects would fit along the shorter side of the bin, and 10 objects along the longer side. Allowing for two samples per part, with the size of the suction cup in mind, 14×20 points were to be sampled within the new bin area. Considering again the final degree of freedom at the tool point brought on by the output from the neural network being only a point and a vector, and not a complete coordinate system, the need for exploring several orientations in each point was still present. Employing the same scheme as in the workspace mapping, 27 different orientations were made in all points, to ensure sufficient exploration. Recall, that a change in robot joint configuration might lead to a solution. This brings the number of total test grasps to $14 \cdot 20 \cdot 27 = 7560$.

As with the workspace mapping, saturation of the results as accounted for in Section 7.3.4 proved equally important in this case. Viewing Figure 7.26, the difference between one attempt at a path can be seen compared to the coverage with five attempts at a path. In the figure, the image to the far left is the new bin placement with one orientation of test grasps shown. The middle image is the result of planning a path once to all orientations in all points, and the result

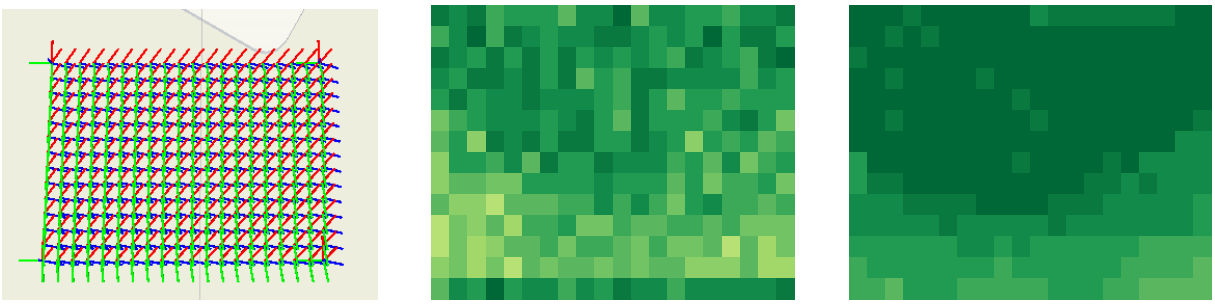


Figure 7.26: Presentation of the importance of saturating the results with non-deterministic planners. To the left: new bin placement with one orientation of possible grasps shown. In the middle: path reachability in the new bin obtained by planning once for a path, the average path reachability is 86.93%. To the right: path reachability in the new bin area allowing for five planning attempts, the average path reachability is increased to 95.70%. Both results were obtained using the planner LKBPIECE and the optimization objective minimizing path length.

is an average path reachability of 86.93%. By planning a path five times instead of once, the average path reachability is raised to 95.70%. The results in the figure were obtained with the planner LKBPIECE (Şucan and Kavraki, 2010) with the optimization objective "minimizing path length" from the Open Motion Planning Library (OMPL) (Şucan et al., 2012).

7.5.2 Optimization objectives

A sampling-based motion planner attempts to find a feasible path, any collision-free path, between the start and final configurations. OMPL supports setting an optimization objective which biases the planners towards optimizing a path according to this objective. By setting an optimization objective, the planner will attempt to answer the query $(\mathbf{q}_s, \mathbf{q}_f)$ whilst also satisfying the path quality metric that is the optimization objective. A common path quality metric is for example the shortest path quality metric which attempts to find the shortest path when traversing the graph on search for the solution (Şucan et al., 2012).

There is also an important distinction between optimizing planners and non-optimizing planners. Optimizing planners, in the context of OMPL at least, will use the entire allotted planning time attempting to find a better path than the initial feasible plan it might have found. For example it may find a feasible path, which is the shortest up until that moment, after 1 seconds, but it will use the rest of the planning time searching for a shorter one. A non-optimizing planner will return the first feasible plan it finds (Şucan et al., 2012).

In OMPL there are several optimization objective which will be explained in the following paragraphs. These objectives are minimizing path length, maximizing the minimum path clearance, a mechanical work objective attempting to minimize the mechanical work done by the robot, and a state cost integral objective which minimizes a cost-function defined over the configuration space. These are all different ways of looking at the cost of a path, and it is assumed that costs of smaller motions can be super-positioned to obtain the total cost of the paths (Şucan et al., 2012).

7.5.2.1 Path length and maximizing minimum path clearance

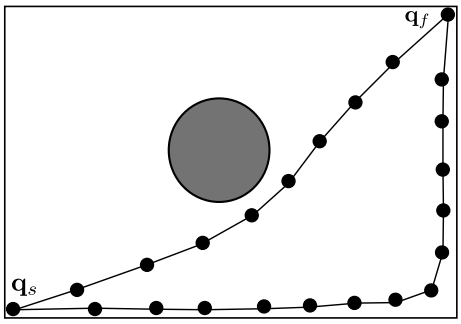


Figure 7.27: Example paths of different objectives. The grey circle in the middle is an obstruction which constitutes a collision. The path closest to it is a "shortest path", whilst the one along the edge is a "maximize minimum clearance" path. Inspired by visualizations found on the OMPL documentation website (Şucan et al., 2012)

The *minimizing path length* objective attempts to find a feasible path from the start state $\mathbf{q}_s$ to the final state $\mathbf{q}_f$ which also is the shortest one. This objective attempts to minimize the sum of accumulated edges when traversing the graph. Recall that the nodes of a graph are collision-free configurations and edges are the paths. The weight of an edge can be viewed as the path length so the goal is to minimize this sum. (Şucan et al., 2012)

The *maximizing minimum path clearance* objective attempts to maximize the clearance to objects in the environment by choosing paths far away from obstructions in the configuration space. The paths chosen for this objective seem to balance going in the direction of the goal as well as steering clear of any obstructions. View for example Figure 7.27 where two example paths from the same start to the same final configuration are illustrated. The one which stays along the border of the workspace is attempting to maximize its clearance from the obstruction in the middle (large circle). This path is longer than the other one, which illustrates a shortest path solution, as described in the section above.(Şucan et al., 2012)

7.5.2.2 Mechanical work and general state cost integral

The notion of mechanical work and the general state cost integral are closely related. To illustrate the functionality of these two objectives, the definitions utilized by Jaillet et al. (2010) will be presented.

Let $\mathcal{P}$ be a path of length l, and let $c : \mathcal{C} \to \mathbb{R}_{\geq 0}$ be a cost function mapping the space $\mathcal{C}$ where a query is to be answered, to a positive number. The cost function is defined such that at each configuration $\mathbf{q}$, there exists an associated cost $c(\mathbf{q}) > 0$. The path $\mathcal{P}$ is represented by $\gamma : [0, l] \to \mathcal{C}$, where $\gamma(s) = \mathbf{q}(s) \in \mathcal{P}, s \in [0, l]$. Moreover, the parametric cost function $v : [0, l] \to \mathbb{R}_{\geq 0}$ of a path is defined as $v(s) = c \circ \gamma(s) = c(\gamma(s)) = c(\mathbf{q}(s))$ (Jaillet et al., 2010).

The following expression produces a notion of the cost of the path and is a more reliable criterion than for example average cost or maximum cost according to Jaillet et al. (2010). $v(s)$ is the parametric cost of a path and the integral of the cost long the path is defined as:

$$S(\mathcal{P}) = \int_0^l v(s)ds \tag{7.8}$$

The function $v(s)$ varies along along the path, and this variation can be viewed as mechanical work. Positive variations of the function can be viewed as "forces acting against motion, and producing mechanical work" (Jaillet et al., 2010). Take for instance a path going over a hill in the configuration space defined by a cost function; going up this hill will cause loss of "energy" as more needs to be spent on going up. When the function has a negative variation no "energy"

will be lost. When looking for these "low mechanical work paths", a small penalty proportional to the distance of the path segment is added, to favor paths that are short, when comparing path segments of equal mechanical energy. Jaillet et al. (2010) define the work of a path as:

$$W(\mathcal{P}) = \int_{\mathcal{P}^+} \frac{\partial v}{\partial s} ds + \epsilon \int_{\mathcal{P}} ds, \tag{7.9}$$

where $\mathcal{P}^+$ is the part of the path where the slope of the path is positive, and the constant ϵ is the small penalty guiding the process to favor short paths when a choice between two segments of equal energy is encountered (Jaillet et al., 2010).

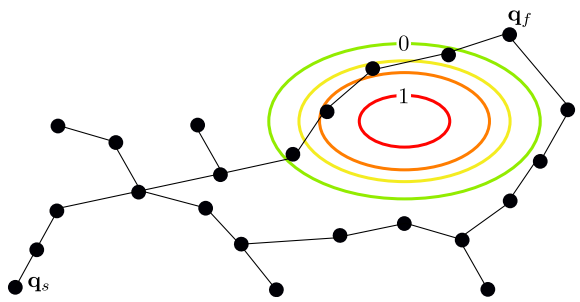


Figure 7.28: Example of functionality of the T-RRT algorithm, inspired by Jaillet et al. (2010). Consider the contours an elevation in 2D terrain with height 1. If the objective was to find a feasible path, the path going over this "hill" is not unthinkable. With the T-RRT which works with cost functions and mechanical work, the path around this hill might be preferred.

Consider as a real-world example a robot in rocky terrain that is to perform outdoor navigation, an example presented by Jaillet et al. (2010). The objective is to find a path to a goal from an initial configuration, where the start is on one side of a hill and the goal state is directly opposite, as depicted in Figure 7.28. Think of the coloured contours as part of a cost-function defined on the configuration space representing this hill. The T-RRT algorithm, which is the one depicted in the figure, uses a cost-function as an additional input to the path planning problem. The optimization objectives "state cost integral" and "mechanical work", are particularly applicable in these types of scenarios. If the only objective was to find a feasible short path, the path going over this hill might be preferred. However, for the T-RRT at least, which utilizes the cost framework, the path going around the hill is likely to be preferred; it could be deduced cost-optimal. Even though the path might be longer, less mechanical work, less cost, is probable (Jaillet et al., 2010).

7.5.3 Metrics for classifying planners

When mapping the workspace to find optimal placement, the default settings for planner and optimization objective was used; LBKPIECE which sought to find the shortest path from the start configuration $\mathbf{q}_s$ to the final configurations $\mathbf{q}_f$. The mapping consisted of looking into whether or not there existed a path to a grasp, and recording the results for uniformly distributed test grasps in part of the workspace. Only one metric was used as criterion, albeit composed of several steps; IK solution, path found, and path found to be collision-free in the post-processing of said path, for this mapping.

The objective of this part of the thesis was to compare different planners from OMPL with a selection of path quality metrics/optimization objectives and compare performance to be able to determine if one is better than the other and find the one best suited for the bin picking design, as it is today, at SINTEF Digital Trondheim. To compare performance, a simple check of whether or not a path exists was hypothesized to not be sufficient. Partly due to this being only one metric providing only one marker of comparison, but also due to the randomized nature of the planners themselves. When mapping the workspace, the data was saturated by running the data collection five times, but in this instance for many more grasps, limitation on time

became a factor which needed to be considered. Tests were also here repeated five times, as this seemed a sufficient number to get a representative average, while at the same time not being too time-consuming.

The fact that a grasp is path reachable, as done with the workspace mapping, is only one criterion which gives very limited information on planner and objective performance. Of course, if one planner and one objective were to stand out with a significantly better result regarding path reachability, this could be theorized to suffice. However, when comparing the planners, more metrics/criteria had to be considered to make the results as robust and informative as possible.

When deciding upon metrics to rank results on the basis of, inspiration was drawn from Meijer et al. (2017) and their benchmarking work regarding use of OMPL with a UR5 robot. They looked at several benchmarks and compared the following metrics; solved runs, computing time and path length. Solved runs refer to how many of their paths were valid in the environment; whether an IK solution existed, a path could be found within the allotted planning time and whether or not this path was found to be collision-free. Computing time refers to the time needed by the planner to calculate a path, and the path length is just that, the length of the path.

Solved runs, or the degree of path reachability, was a metric already available in the implementation of this work. If there existed an IK solution and a collision-free path, the grasp was marked in the underlying data structure that it indeed was valid. This metric is also the most important one, considering that any tests subsequently done on this test grasp would be unnecessary. Since this is a corner stone of the work, this metric was of course utilized when looking at planners and objectives.

Information on computing time needed by the planners is a metric which could be detrimental in terms of concluding on the quality of a grasp when connected with whether a grasp is path reachable. Consider for example two grasps, which are both reachable, and thus ranked equal in terms of path reachability since this is a Boolean value. If one of the grasps was found to be path reachable by the planner after 2s and the path for the other grasp was found in 0.1s, this latter grasp would then rank above the former when considering the two criteria in unison.

Algorithm 5: Excerpt of algorithm used for finding planning time

```
print("Start planning")
start = time.time()
plan = move_group.plan()
end = time.time()
print("Done planning")
planning_time = end - start
```

The time it takes the planner to find a solution is presented to the user in the terminal (echoed). However, it proved difficult to obtain for usage in the script since it does not exist as a variable in the Python application program interface (API) for ROS. One way of solving this is with the help of the internal clock of the computer. In the API, the action of planning a path is done by calling the `plan`-command for the `move_group`-class. When finding the planning time, the time before planning was recorded, the `plan`-command was called, the time was recorded again, and the difference between the two was calculated and labeled "planning time". Since the actual planning time from OMPL was visible in the terminal, it was possible to compare the two parameters, and hopefully verify the procedure. For one complete test of all test grasps in the bin, 7560 different grasps, the first and last ten planning times were copied from the terminal and compared to the timed variable from the script. For both the data sets, a consistent deviation of 0.10 seconds between the actual value and the timed value was found. Down to two decimal places this is a valid approach, and the variable

was kept as a metric usable for comparison and ranking of grasps. It was important to note however, that the third decimal and onward in this deviation is not consistent and must be considered measurement noise. The planning time can only be guaranteed up until two decimal places.

When calling the `plan`-command, the returned variable is an array of configurations in the workspace the robot will pass through en route to the goal pose. These configurations present information on joint position, velocity and acceleration for the path, as well as the variable `time_from_start`. The length of the path became the measure of taking the length of the array the plan consists of. If the path is short, fewer robot configurations would be necessary to reach it. If a path is longer, more configurations are needed. The metric is relative to other paths, and does not necessarily have a unit, but is just the number of configurations needed by the planner to construct a complete path. To record the execution time of each path, the last robot configuration in each plan was identified, and the variable `time_from_start` was extracted from this entry in the plan.

Two rows in the resultant table for each planner could for example look like this, dependent on the test grasp:

$\mathbf{T}_j^{\text{test}}$	Path	Path length	Planning time	Execution time
$\mathbf{T}_1^{\text{test}}$;	True ;	38 ;	0.65218186378479 ;	4.80159859
$\mathbf{T}_2^{\text{test}}$;	True ;	30 ;	0.25548410415649414 ;	3.648819854
$\mathbf{T}_3^{\text{test}}$;	False ;	;	;	

7.5.4 Testing

Based on the work of Meijer et al. (2017), who also utilized a UR5, a selection of planners to investigate for the set-up at hand were chosen on the background of their results. The functionality of each of these planners are outlined in Chapter 5. The choice was based upon comparing their benchmark tests and viewing their results. Meijer et al. (2017) test each manipulator on their ability to place a grasp, pick a grasp and place a grasp with motion constraints. The first benchmark is designed to reveal which planner produces the best results when moving out of a constrained space, and the second is designed to identify which of the planners perform well when moving into a constrained space (Meijer et al., 2017). Tying this together with the set-up, both these tasks are to be undertaken if one views moving in and out of the bin as moving in and out of a constrained space. The third benchmark includes motion constraints, which they compare to holding level a glass of water, and was the benchmark with the least impact on the task at hand. The performance metrics they utilize are solved runs, computing time and path length. Solved runs refer here to the number of feasible, or optimal paths with the use of an optimal planner, found during a planning session. This is equivalent to the degree of path reachability.

Viewing the results for the first benchmark, there were several planners which performed well for the UR5 in terms of solved runs, but fewer that performed as well in terms of computing time and path length. The planners ProjEST, KPIECE, LBKPIECE, RRTConnect and BiTRRT were the best performers for this benchmark. SBL and EST also performed satisfactory. Furthermore, looking at the second benchmark, only the planners SBL, BKPIECE, LBKPIECE, RRTConnect and BiTRRT produced results for the UR5. LBKPIECE and RRTConnect were the best performers in terms of all three metrics, solved runs, computing time and path length. When comparing SBL and BiTRRT, SBL has a significantly higher computing time, but no

outlier data when it comes to path length. BiTRRT however, has some outliers. Since short path lengths were sougth to ensure fast picking in the set-up at hand, the choice fell on SBL. On the basis of these results, the three planners that were to be tested became: LBKPIECE, RRTConnect and SBL.

OMPL supports 23 different planners and several optimization objectives. The three planners chosen were discussed in Chapter 5, and their results are presented here. The user has some freedom in setting the configuration parameters for each planner. Default settings for each planner were used and the planning time was set to 5 seconds for all results presented here. There are two important configuration parameters which are worth noting. The first is the "projection evaluator" which takes a list of joints as an approximation of the configuration space. This parameter is set to `joints(shoulder_pan_joint,shoulder_lift_joint)`. This parameter is in use in the planner LBKPIECE.

The second parameter is the "longest valid segment fraction" which determines the resolution of the collision-checking when traversing a path on search for one that is feasible. This parameter is set to `longest_valid_segment_fraction: 0.01`. The collision-checker implemented in OMPL discretizes the edge between two nodes to a set number of states and checks each configuration for collisions. This parameter is the fraction of the configuration space assumed to be collision-free when the robot is not in collision in a certain configuration. In this case, where it is set to 0.01, we assume the robot can move in 1% of the configuration space and still be collision-free (MoveIt, 2019). The size of this variable affects the behaviour of the planners, and declaring it a smaller value led to some time out issues when the allowed planning time was as strict as it was.

The test set-up is similar to the algorithm used for planning to potential grasp poses in the workspace outlined in Algorithm 3, with a few changes. The data collection algorithm for testing the different planners by making a path reachability map of the new bin area as well as collecting information on the metrics accounted for in Section 7.5.3 for each of the planner is outlined in Algorithm 6:

Algorithm 6: Algorithm for creating path reachability maps and data collection for planners under investigation

for *every point* (x, y) *in the new bin* **do**
  for *all defined orientations about* $(x, y, 0)$ **do**
    generate the test grasp $\mathbf{T}_j^{\text{test}}$;
    define goal pose of the planner as the test grasp frame;
    plan a path to the pose with MoveIt;
    if *the pose is path reachable* **then**
      save $\mathbf{T}_j^{\text{test}}$ and **True**;
      record computation time;
      extract planning time;
      extract path length;
    else
      save $\mathbf{T}_j^{\text{test}}$ and **False**;
    end
  end
end

Chapter 8

Results

8.1 Optimal level mapping

To compare planners and metrics, it was advantageous to plot the different information to visualize it. Since each of the test sets were run five times, the results first had to be combined. Whether or not a test grasp was path reachable or not became a matter of employing Algorithm 4; if there was found a path at one instance, a path exists. For the path length, an average size was chosen, if there were found for example three paths out of five tries, these three numbers were averaged, and the final result was set as the parameter for that test grasp. The same procedure was carried out for planning time and execution time as well. Furthermore, to be able to compare the different planners and objectives, plots were made, covering their performance in accordance with the metrics.

When running Algorithm 6 on the the three different planners with all four optimization objectives, 12 saturated sets of data are available. To determine which planner and objective which performed the best, the data was plotted in a box plot, where path length, planning time and execution time is presented, see Figure 8.2. The statistics on path reachability degree is presented using a bar graph, shown in Figure 8.1.

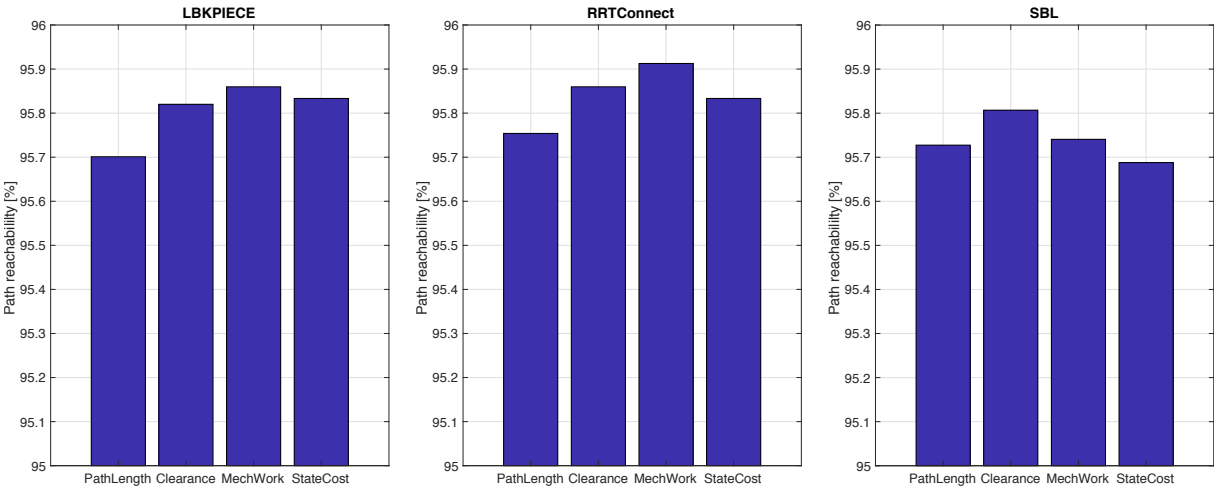


Figure 8.1: Total percentage path reachability for the planners LBKPIECE, RRTConnect and SBL with the optimization objectives in order: minimizing path length, maximizing minimum path clearance, minimizing mechanical work and minimizing the general state cost integral. Note that the y-axis goes from 95% to 96%.

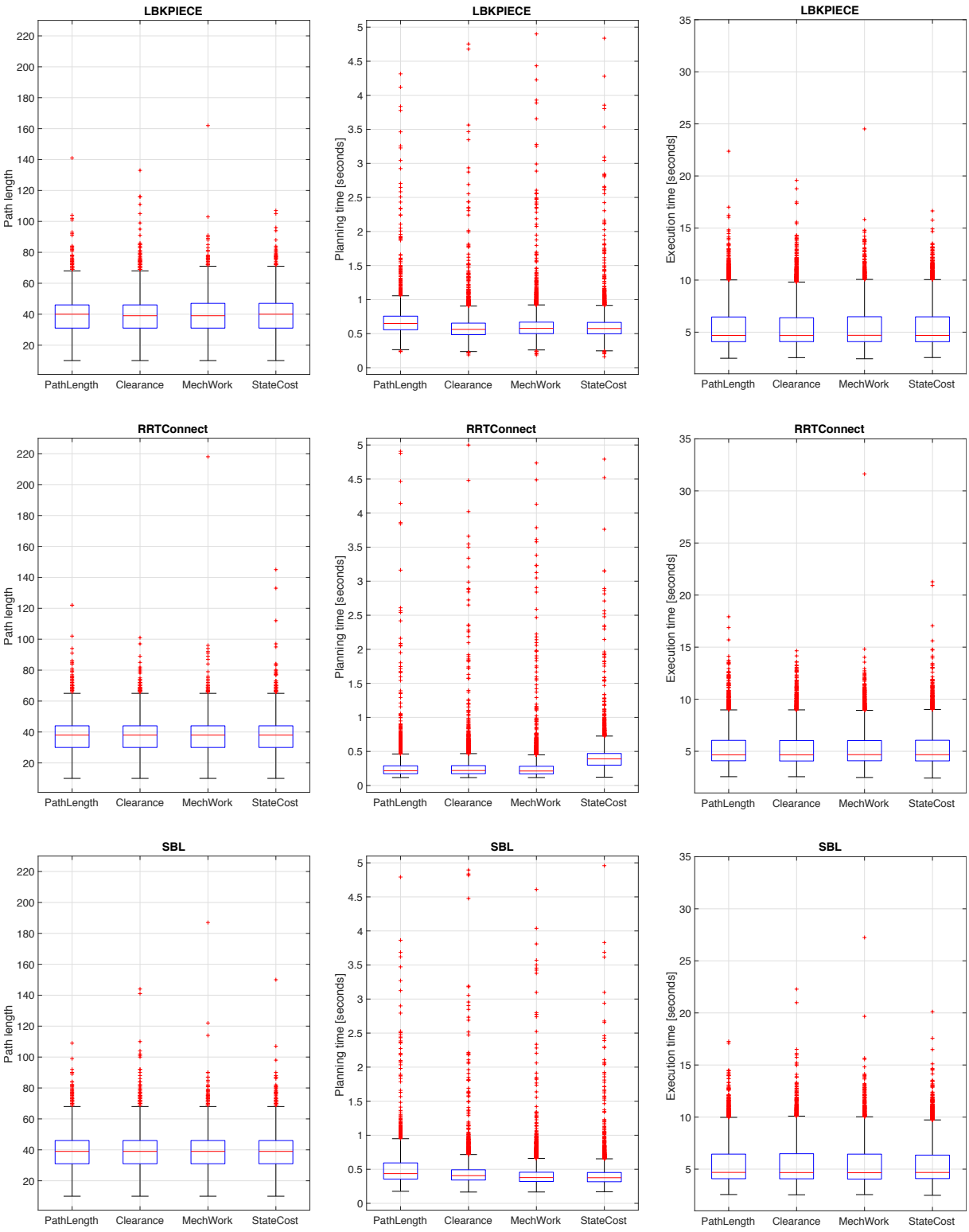


Figure 8.2: Box plot of all planner and optimization objectives tested. Top row: LBKPIECE. Middle row: RRTConnect, Bottom row: SBL. The results for the optimization objectives are in the same order in all plots; minimizing path length, maximizing minimum path clearance, minimizing mechanical work and minimizing the general state cost integral. All axes with respect to metrics of interest (columns) are equal. In each box, the red line within the blue rectangle represents the median value of the data set. The blue box in each plot contains the 2nd and 3rd quartile of the data. This indicates that results which lie between 25% and 75% of the distribution is contained by the box. The red crosses are data points considered outliers, and the "whiskers" extends to data not considered outliers (MathWorks, 2019).

Viewing first Figure 8.1 where each bar is the percentage degree of total path reachability for a planner and objective. Note that the y-axis goes from 95% to 96%, meaning all planners performed well in terms of this metric. Still, there are slight differences, and RRTConnect with the optimization objective mechanical work performs the best with a total path reachability of just above 95.9%.

Secondly, view Figure 8.2 and the first column to the left. These three plots show the distribution of path lengths of all of the paths for the three planners and objectives. These results are quite similar, but RRTConnect performs slightly better than the other two planners. It has a smaller median than the two other planners. LBKPIECE has the least amount of outliers considering this metric, and RRTConnect has the worst outlier with the mechanical work optimization objective.

Moving on to the next column of results, the middle column portraying the planning time distribution of the different planners and objectives. It is immediately apparent that RRTConnect outperform both SBL and LBKPIECE with regards to this metric, except with the state cost objective. All planners have a significant stretch towards the upper limit for planning time ($5s$), but on average RRTConnect is the best performer here. When it comes to the final metric, execution time, this looks to have the same structure as path length. This is not unanticipated, since the length of the path and the time needed to traverse it will be closely related. Take for example the outlier of RRTConnect with the mechanical work objective in path length; the corresponding execution time can be seen for the same objective in this final column. The same conclusions are reached for this metric, RRTConnect slightly outperforms the other two planners, also in terms of the amount of outliers. The optimization objective maximizing minimum clearance objective is the best performer in execution time.

Based on viewing the plots and the previous discussion, the planner RRTConnect is the best performer, and due to the consistent distribution of data using the maximizing minimum clearance objective, this is the best choice of planner and objective for the set-up at hand among the tested planners. Also looking at this planner and objective with regards to path reachability, it is the second best result considering RRTConnect alone, and performs just as well as the best results obtained with LBKPIECE. This planner and objective performs the best in path length and execution time, performs well in planning time, and is the second best option in path reachability.

In Figure 8.3, the path reachability map made with RRTConnect and optimization objective maximizing minimum path clearance can be seen. Recall that in each of these small squares, there are 27 different orientations where the colour is indicative of the percentage of how many of these were path reachable. For each of these path reachable orientations, a path with a corresponding path length, planning time and execution also exists. These final three metrics are not represented in the map, but is stored in another data structure with connections to the map.

Seeing that the metrics up for testing were path length, planning time and execution time, one can argue that path length and execution time are closely related as a shorter path would take less time to traverse. To investigate this assumption, the path length and execution time of several of the planners and objectives were made, and one of these plots can be seen in Figure 8.4. This plot is of the planner LBKPIECE with optimization objective minimizing path length. The same type of behaviour was observed with other planers and objectives.

In the figure, all the points represent a path's length and the corresponding execution time. The blue line is the linear fit to the data set. The red line indicates another trait of the data; when the path is relatively short, it can be seen that the execution time is also relatively low. This

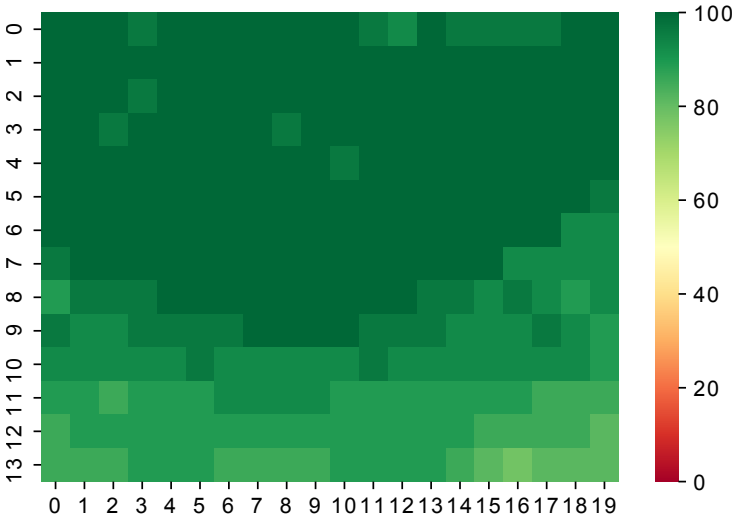


Figure 8.3: Path reachability map made with RRTConnect and optimization objective maximizing minimum path clearance. There is one square for each sample in the bin, and the colour indicates percentage coverage.

red line seems to be the lower bounds on how fast a path can possibly be. It is indicative of the fastest paths possible in terms of the constraints on the robot actuators, the servos cannot move faster. Since this property is observed consistently, there is not a pure linear relationship between path length and execution time, meaning that they are not linearly dependent. It is thus advantageous to use both metrics in a total cost function, as will be illustrated in the following case study.

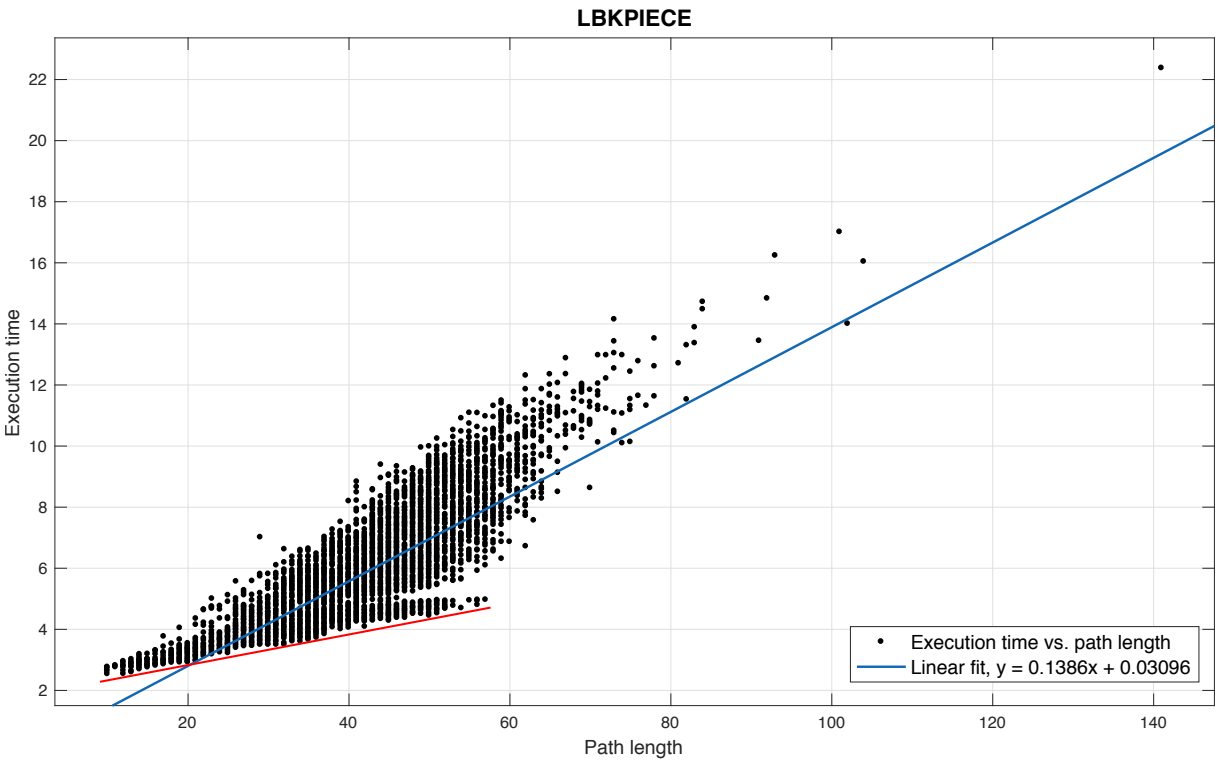


Figure 8.4: Plot of the path length as function of the execution time. The data set is from a planning session using LBKPIECE with the optimization objective minimizing path length.

8.2 Case study

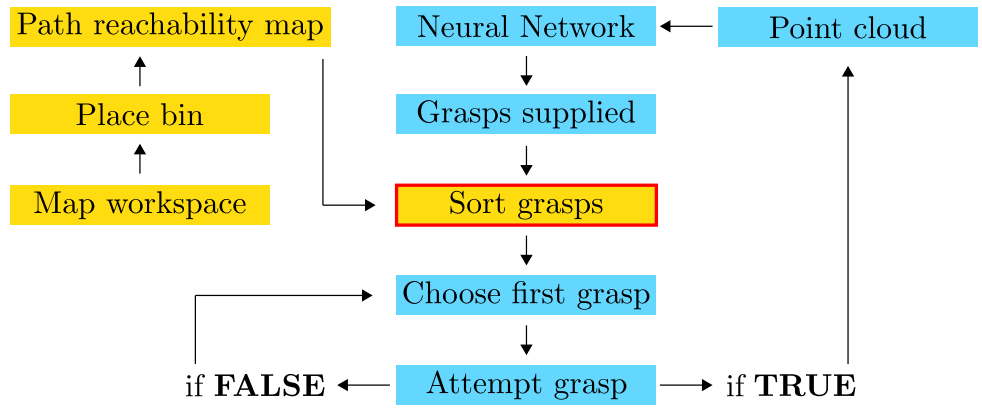


Figure 8.5: System flow with the implementation of the path reachability map as a method to sort the grasps.

After the best path reachability map had been decided upon, the map using the planner RRT-Connect and the optimization objective maximizing minimum path clearance, it was of interest to use the map as an additional step in the grasp selection process. Now we have available a generous amount of information on path reachability, path length and time consumption for this planner and objective, called the path reachability map. The goal now was to use this information to bias the grasp selection in the system towards grasps that are path reachable and "good" for the robot. Considering the robot's reach has not been included previously in the system, a method to do this was of interest. Viewing again Figure 8.5 (same as Figure 7.2), this is the work flow implemented and tested in the following sections.

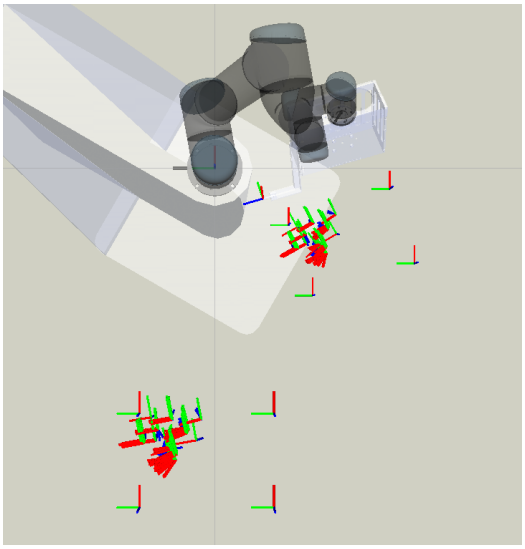
Provided a grasp from the neural network, we can find the test grasp that most resembles the true grasp, and then be able to return information on path length and time. Take for instance the network outputting 100 grasps in order ranked on the quality of the grasp. We then wish to rearrange this ranked list based on similar test grasps and their attributes.

8.2.1 Prerequisites

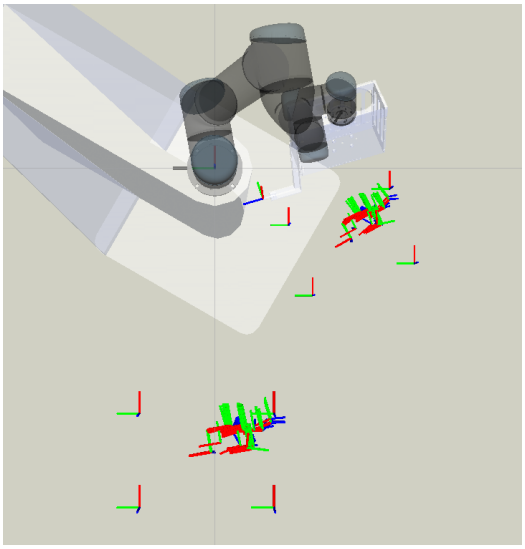
The neural network outputs sets of grasp pairs $(\mathbf{d}_i, \mathbf{v}_i)$, where $\mathbf{d}_i$ is a point and $\mathbf{v}_i$ is the approach vector for the grasp. The most common reasons a grasp fails are collisions with the bin edges or with other objects in the bin. If and when a grasp fails, the system automatically moves on the next grasp from the network, without capturing a new point cloud and obtaining a new list of grasps. The probability that objects move if a grasp fails due to collision with the objects in the bin has been experienced to be high. It is due to these events, that the bin area has been split in four; to increase the chance of successfully grasping an object. For example, if a grasp fails in the south eastern quadrant of the bin due to collision, it is less likely that this disturbs the distribution in the north western quadrant such that the grasp pairs given here still might be valid.

For each quadrant in the bin, the neural network was called to output 100 grasps in order. Next, the grasp set was pruned automatically to remove grasps given outside the bin, or on the edges of the bin. In Figure 8.6, the grasps from the neural network can be seen as the clusters of coordinate frames. In the figure, the same grasps are shown both in the current placement, and the new placement of the bin. Furthermore, to make a complete frame from the pairs $(\mathbf{d}_i, \mathbf{v}_i)$ to

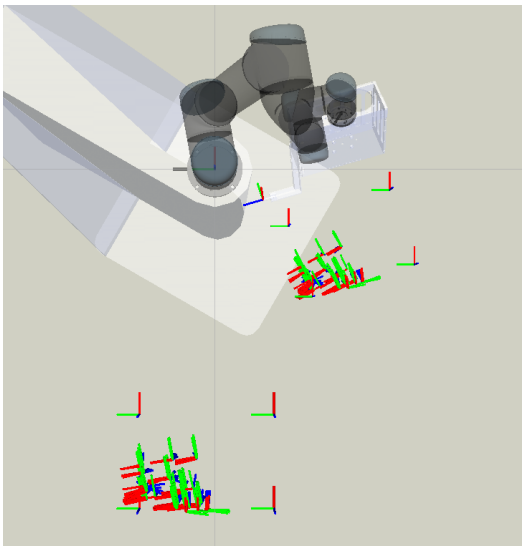
be able to test the algorithm soon to be presented, the x-axis of the neural network grasps were set to coincide with the x-axis of the tool centre point at the point $\mathbf{d}_i$, see the frame attached to the tool in the figure. Recall, RGB$\rightarrow (x, y, z)$. Prerequisite number one for testing is thus to complete the grasp pairs such that a complete frame is available.



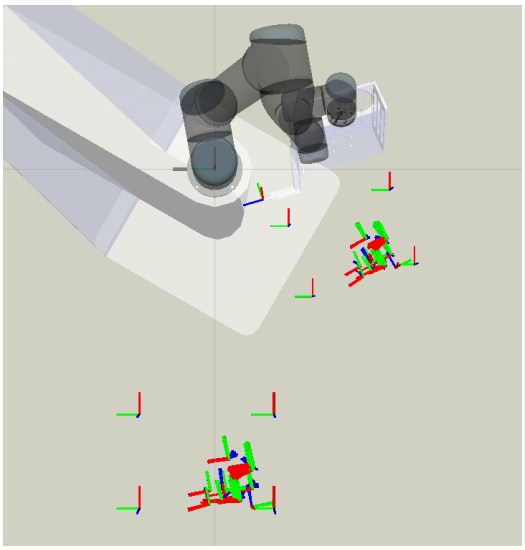
(a) Grasps from the neural network in the north western corner given in both the current placement of the bin, and the new.



(b) Grasps from the neural network in the north eastern corner given in both the current placement of the bin, and the new.



(c) Grasps from the neural network in the south western corner given in both the current placement of the bin, and the new.



(d) Grasps from the neural network in the south eastern corner given in both the current placement of the bin, and the new.

Figure 8.6: Grasps from the neural network in both the current and the new bin placement. The grasps from the neural network are the clusters of coordinate systems. Recall the following: x, y, z.

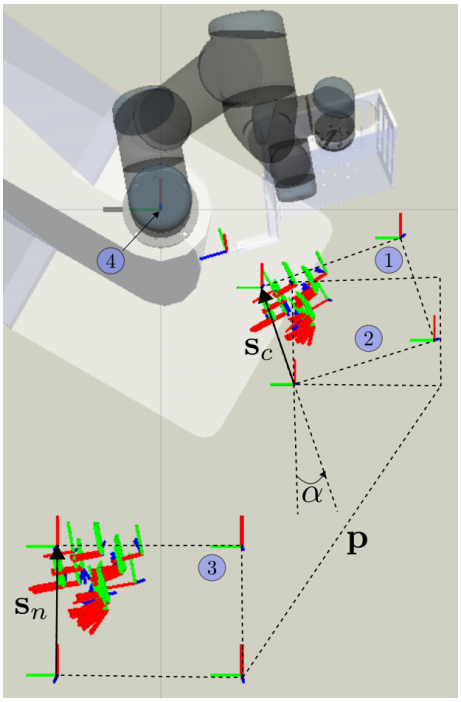


Figure 8.7: Transforming the neural network grasps from the current placement to the new placement. 1) marks the current placement, 2) marks the rotated current placement, 3) marks the new placement, and 4) marks the base coordinate system. α is the rotation angle between the current and new placement, and $\mathbf{p}$ is the required translation to align the two bins.

The grasps from the neural network are given relative to the current placement of the bin, the placement seen in Figure 7.4. Transforming the grasps to the new bin, as shown in Figure 7.24, enables testing of the algorithm. Since the current placement, and the new placement had been done separately, it was first necessary to find the transformation between the two, to be able to represent the grasps from the neural network in the new bin area. The second prerequisite is to transform the grasps from the neural network to the new bin area.

To find the rotation angle between the current and new placement, the dot product between the vectors of the short sides of the bin was taken:

$$\cos\alpha = \frac{\mathbf{s_c} \cdot \mathbf{s_n}}{|\mathbf{s_c}| \cdot |\mathbf{s_n}|}, \tag{8.1}$$

where α is the angle between the bins, $\mathbf{s_c} \in \mathbb{R}^3$ is the vector describing the short side of the bin in the current placement and $\mathbf{s_n} \in \mathbb{R}^3$ describes the short side of the bin in the new placement, see Figure 8.7. After having identified this angle and rotated the bin, the translation could be found.

The vector needed to translate the bin was found as the Euclidean distance from one corner of the rotated bin in the current placement to the same corner in the new placement. The grasps from the neural network needed to be transformed by the same sizes as the bin. The complete transformation needed to obtain the grasps in the equivalent placement in the new bin, provided the current bin already was rotated is:

$$\mathbf{T}_{\text{grasp, new bin}} = \mathbf{T}_1\mathbf{T}_2\mathbf{T}_{\text{grasp, current bin}}\mathbf{T}_4 \tag{8.2}$$

$$\mathbf{T}_{\text{grasp, new bin}} = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{T}_{\text{grasp, current bin}}\ \mathbf{T}^{\text{ee}}_{\text{tp}}, \tag{8.3}$$

where $\mathbf{T}_1$ is the translation from the current placement to the new placement, by finding the vector $\mathbf{p}$ connecting the two. Recall that the current placement of the bin has already been rotated by α, see Equation (8.1), such that the distance from each corner in the (rotated) current placement to each corner in the new placement were all vectors of equal size. $\mathbf{T}_2$ is a rotation about the base z-axis to rotate the grasps from the neural network, $\mathbf{T}_{\text{grasp, current bin}}$, to the new placement. Note, that this is not a rotation about the z-axis of the grasps themselves, but by the base z-axis pointing up from the pedestal, also marked in Figure 8.7. Furthermore, the grasps from the neural network are given in the robot base with respect to the end-effector, $\mathbf{T}^{\text{base}}_{\text{ee}}$, and must be transformed to the tool center point by the static transform $\mathbf{T}^{\text{ee}}_{\text{tp}}$. Note again the use of pre-multiplication for operations about the fixed base frame and post-multiplication for operations on the grasps themselves.

The third and final prerequisite required to use the path reachability map, is the appropriate saturation and normalization of the information in the map. For each test grasp in the new bin area, recall that 5 attempts at a path were executed. This meant that there were five reachability maps with associated path lengths, planning times and execution times. To saturate the results with respect to path reachability, a Boolean value, this was simply the operation of OR-ing this value. Furthermore, due to the sampling-based nature of the planners, the same path was rarely returned. So for each test grasp, the average of the found path lengths, planning times and execution times were used in a final *saturated path reachability map*. Viewing Figure 8.8, the top row shows the histograms for the average values of path length, planning time and execution time respectively.

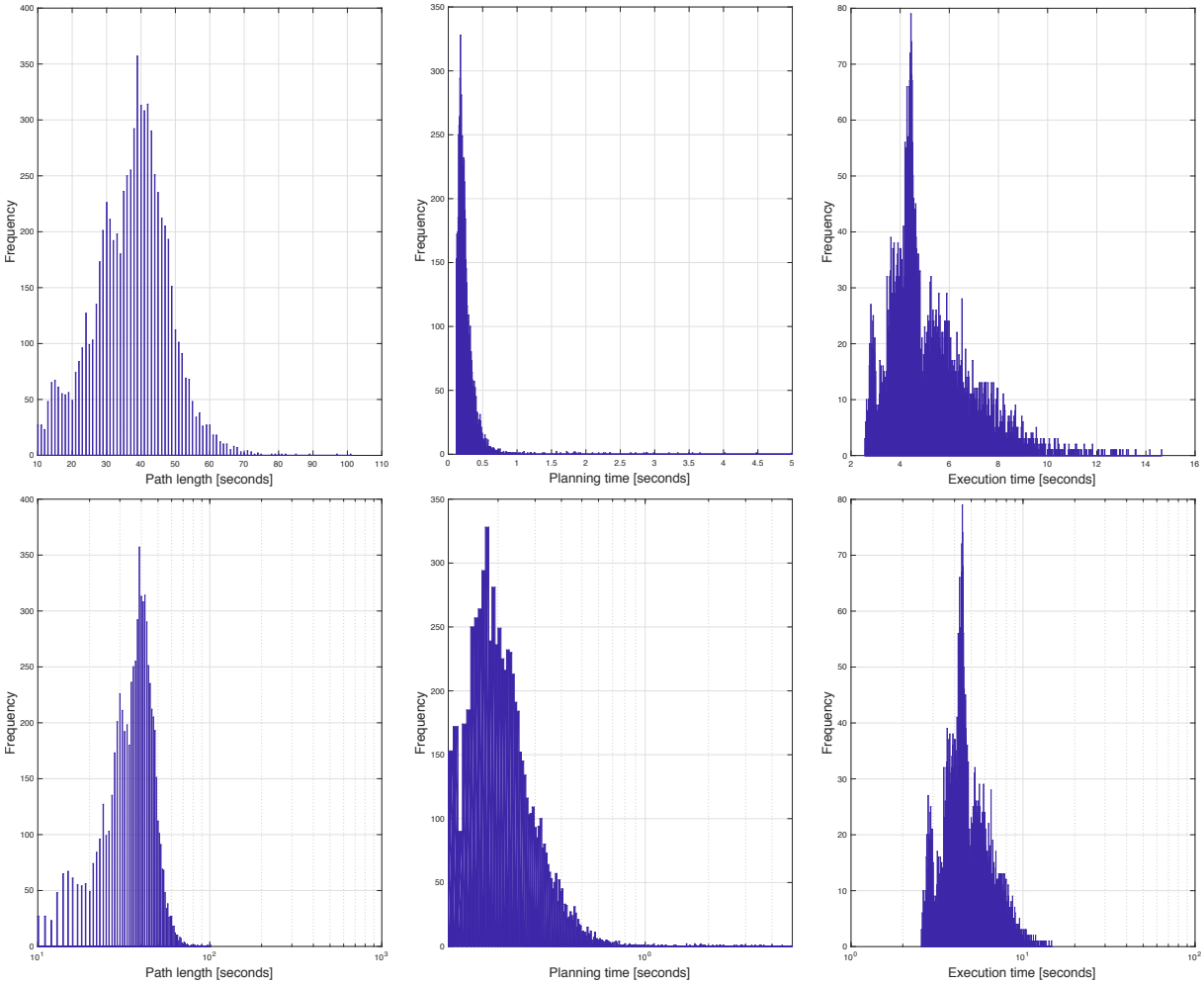


Figure 8.8: Histograms of averaged values of path length, planning time and computation time. Top row: plotted on a linear scale, notice the substantial right skew. Bottom row: same data plotted on a logarithmic scale, notice the data more resembling a normal distribution.

When viewing the top row of plots in Figure 8.8, it is clear that the data has a significant right skew, with a substantial amount of outliers in the upper regions. Since the objective here is to compare and weight the metrics (path length, planning time and execution time), a proper normalization scheme is decisive. As we want to compare sizes different in nature, they have to be normalized equally.

If one was to normalize this data with a max-min normalization by the following formula:

$$\text{normalized value} = \frac{\text{current value} - \text{minimum value}}{\text{maximum value} - \text{minimum value}}, \tag{8.4}$$

which attempts to focus the data around the mean, outliers disturb this process. Due to the right-skewness and the amount of outliers, a different approach had to be undertaken.

View again, Figure 8.8, but the bottom row. These are the same values plotted on a logarithmic scale, where it can be observed that the outliers have a lesser impact on the distribution, the "tail" of the graphs has been shortened. If the logarithm of the data set produces a normal distribution, one has a log-normal distribution (Walpole et al., 1998). It is clear that the data does not have a perfect normal distribution in these bottom row plots, but it is significantly more normal than the plots in the top row.

Since a logarithmic presentation of the data provided a distribution more closely resembling a normal distribution, albeit far from perfect, the choice fell on using a logarithmic normalization of the data set. An alternative to using a log normalization, is to use a square root or reciprocal normalization (Kenny, 1987). All three metrics must be normalized using the same scheme to be compared correctly, and the log normal distribution was the one where they were most similar in behaviour. Furthermore, the logarithmic normalization suppresses the impact outliers have on the data set. Since the data is more concentrated around lower values, we want the outliers to interfere less with the normalization, and damp their impact on the data.

To normalize the data according to a logarithmic distribution, the terms of the normalization seen in Equation (8.4) were "log-ed" and implemented:

$$\text{normalized value} = \frac{\log(\text{current value}) - \log(\text{minimum value})}{\log(\text{maximum value}) - \log(\text{minimum value})} \tag{8.5}$$

8.2.2 Implementation

After the grasps from the neural network were provided, the path reachability map had been saturated, and the metrics had been normalized, an algorithm which composes the path reachability test had to be implemented:

- Given the grasps from the neural network, the test grasp resembling it the most is to be identified.
- The path length, planning time and execution time for this corresponding test grasp is then looked up in the map (a large table).
- A cost function is evaluated based on the attributes of the test grasp for each grasp from the network.
- The grasps from the neural network are re-arranged based on this cost and returned to the picking loop.

$\mathbf{T}_{\text{grasp, new bin}}$ is now the exact same grasp from the neural network given in the current bin placement, simply transformed to be given in the new bin area. From here on $\mathbf{T}_{\text{grasp, new bin}}$ will be referred to as $\mathbf{T}_i^{\text{nn}}$ to mean the i^{th} grasp pose from the neural network, abbreviated nn.

Recall the following relations for the pose of the test grasps generated, and the grasps provided by the neural network:

$$\mathbf{T}_i^{\text{nn}} = \begin{bmatrix} \mathbf{R}_i^{\text{nn}} & \mathbf{d}_i^{\text{nn}} \\ \mathbf{0}^\top & 1 \end{bmatrix} \text{ and } \mathbf{T}_j^{\text{test}} = \begin{bmatrix} \mathbf{R}_j^{\text{test}} & \mathbf{d}_j^{\text{test}} \\ \mathbf{0}^\top & 1 \end{bmatrix}. \tag{8.6}$$

Furthermore, path length, planning time, and execution time are normalized average values such that they can be compared with regards to how much each of these sizes influence the cost of a grasp. The path length will be referred to as ϕ_3, planning time ϕ_4 and execution time ϕ_5, as the metrics ϕ_1 and ϕ_2 are related to identifying the test grasp which resembles a true neural network grasp the most. This will be accounted for in the following section.

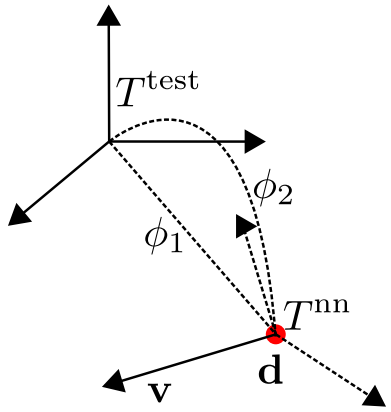


Figure 8.9: Illustration of ϕ_1 and ϕ_2. ϕ_1 is the Euclidean distance between d^{nn} and d^{test} and ϕ_2 is the geodesic distance between R^{test} and R^{nn}

The first step in identifying which test grasp resembles a grasp from the neural network the most is a check of the distance between the network grasp point and all test grasp points in the data set, and choosing the closest test grasp. So, for each grasp from the network and all the test grasps, we calculate the Euclidean distance $\phi_1 : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}_{\geq 0}$:

$$\phi_1(\mathbf{d}_i^{\text{nn}}, \mathbf{d}_j^{\text{test}}) = |\mathbf{d}_i^{\text{nn}} - \mathbf{d}_j^{\text{test}}|, \tag{8.7}$$

where $\mathbf{d}_i^{\text{nn}}$ is the position vector of the grasp from the neural network and $\mathbf{d}_j^{\text{test}}$ is the j^{th} test grasp position vector. After the point closest resembling the neural network grasp point is found, there are 27 options for rotation, where a comparison of the 3D rotations must be done to identify the approach angle closest in magnitude. Based upon the work of Huynh (2009), the metric in Equation (8.8), $\phi_2 : SO(3) \times SO(3) \to \mathbb{R}_{\geq 0}$, is applied to all test grasps and compared to the neural network grasps. The matrix logarithm is defined as follows: Let $A \in \mathbb{C}^{n \times n}$, then $B \in \mathbb{C}^{n \times n}$ is a logarithm of A if $e^B = A$ (Bernstein, 2018).

$$\phi_2(\mathbf{R}_i^{\text{nn}}, \mathbf{R}_j^{\text{test}}) = \left\| \log(\mathbf{R}_i^{\text{nn}} (\mathbf{R}_j^{\text{test}})^\top) \right\|, \tag{8.8}$$

where $SO(3)$ is the special orthogonal group of order 3, $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers, $\log(\cdot)$ is the matrix logarithm, $\mathbf{R}_j^{\text{test}}$ is the rotation part of the transformation matrix describing the test grasp j, and $\mathbf{R}_i^{\text{nn}}$ is the same for the neural network grasps.

The $\|.\|$ (2-norm), gives the magnitude of the rotation angle Huynh (2009). The metric returns values in the interval $[0, \pi)$, where the objective is to find the smallest geodesic distance between the neural network grasp and test grasps. The pseudo code is outlined in Algorithm 7. The first two for-loops return the test grasp closest in ϕ_1 and ϕ_2, and the next loop evaluates the cost function J for each grasp, and enables sorting.

After the closest test grasp has been returned, the attributes of this test grasp in terms of path length, planning time and execution time, ϕ_i, $i \in [3, 4, 5]$, is to be evaluated by the help of a cost function;

$$J_i = \alpha\phi_3 + \beta\phi_4 + \delta\phi_5, \tag{8.9}$$

where α, β and δ are weighting parameters enabling prioritization of characteristics that are more important than others in an application. Now that ϕ_i, $i \in [3, 4, 5]$ are normalized, Algorithm 7 ensures that path reachable grasps with low computation times and short paths are prioritized over less ideal grasps, and allows the highest rated grasp to be the one best for the robot as well. Information on IK, path-existence, collision-checking and time consumption is implicitly included in the cost function J as it is calculated offline, and is accessible faster than the time it would require to do the operations sequentially for each grasp from the network.

Algorithm 7: Path reachability test: Algorithm using the path reachability map for rearranging grasps from the neural network taking into account the robot capabilities.

Input: Robot capabilities in terms of path reachability map and corresponding information on path length, planning time and execution time, and list of grasps from neural network
Output: Re-arranged list of grasps in terms of path reachability
for *all grasp from neural network* $\mathbf{T}_i^{nn}$ **do**
  for *all test grasps* $\mathbf{T}_j^{test}$ **do**
    calculate $\phi_1(\mathbf{d}_i^{\text{nn}}, \mathbf{d}_j^{\text{test}})$ and $\phi_2(\mathbf{R}_i^{\text{nn}}, \mathbf{R}_j^{\text{test}})$;
  end
  return the closest test grasp in ϕ_1 and ϕ_2;
end
for *each test grasp corresp. to a network grasp* **do**
  if *the grasp is path reachable* **then**
    calculate cost of each grasp;
    $J_i = \alpha\phi_3 + \beta\phi_4 + \delta\phi_5$;
  else
    $J_i = \infty$;
  end
end
sort grasps with respect to J;

8.2.3 Evaluation

To evaluate the functionality of the algorithm, all weights, α, β and δ were set to 1 such that path length, planning time and execution time of the paths were considered equally important, since these three aspect influence how fast a full picking cycle is.

A comparison between the list of grasps from the neural network and the re-arranged list of grasps when considering path reachability, had to be investigated. To do so, the first 10 grasps from the network, unsorted on path reachability, and the first 10 grasps from the re-arranged list were recorded. The average cost $J = \alpha\phi_3 + \beta\phi_4 + \delta\phi_5$ of these groups was calculated and plotted in a bar chart, seen in Figure 8.10. There is one plot for each quadrant in the bin.

Viewing the figure, the columns to the right in each group represent the average cost of the first 10 grasps from the network, not considering path reachability. The columns to the left in each group represent the average cost of the first 10 grasps of the sorted list. The value of 10 was chosen to obtain a representative average when comparing the cost J. When comparing the two, it can be seen that when including path reachability, grasps with a short path length, planning time and execution time, are prioritized. Note that the y-axis goes to infinity since some of the grasps preferred by the network were unreachable by the robot in the south west quadrant of the bin, this is illustrated in Figure 8.10 by the grey column. When comparing the results for the south-east and south-west regions in the bin in terms of a higher cost, this is consistent with the slightly lower level of path reachability in these regions as can be seen in Fig 7.26.

The time the algorithm requires to process and sort the grasps from the neural network and the map in one quadrant is under a quarter of a second, and in theory it should be able to process all four quadrants in under a second to account for path reachability. The results indicate that faster picking can be achieved when including path reachability as a separate module in the bin-picking system, whilst still keeping the neural network focused on grasp planning alone.

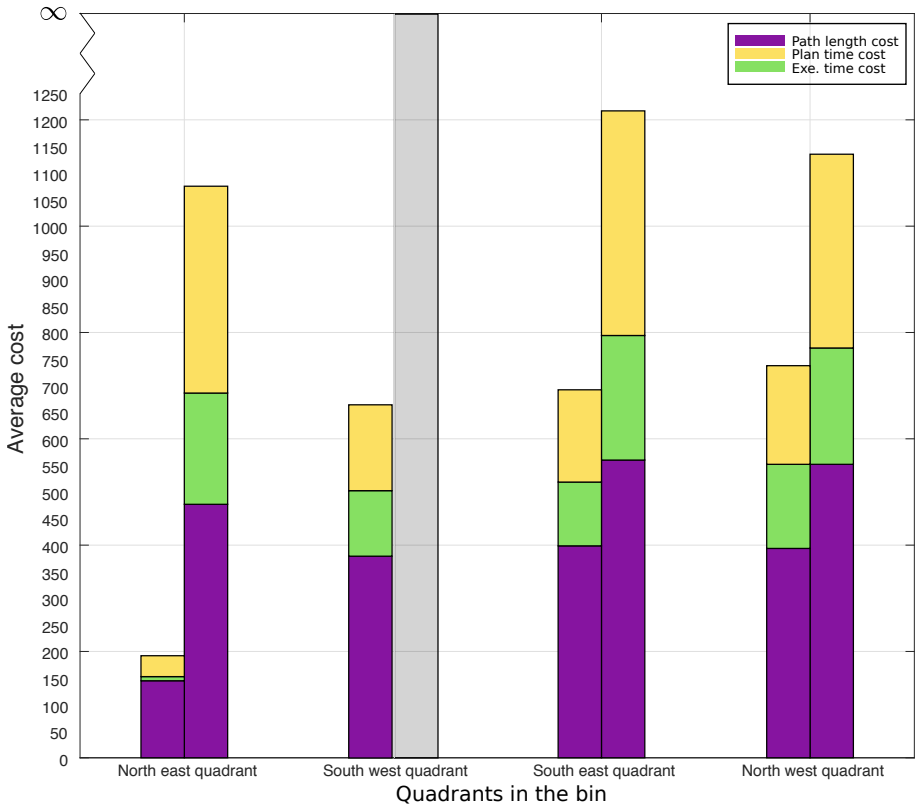


Figure 8.10: Comparison of the average cost of the first 10 grasps from the neural network (column to the right) and the first 10 grasps from the re-arranged list based on path reachability (column to the left) The average cost is split into the average sub costs of path length, planning time and execution time for the grasps.

Chapter 9

Conclusion

9.1 Concluding remarks

In this work, a mapping of the workspace of the system shown in figure 1.1 has been undertaken. Based on this mapping, the region of the workspace with the highest *path reachability* was chosen as the best location for placing the bin, when considering that a robot needs to reach the objects to be picked and the grasps provided by the neural network. Due to the constraints on the end-effector, and the constraints in the robot cell such as the pedestal and the bin, collision-free paths had to be found and considered. After placing the bin, several planners and optimization objectives from the Open Motion Planning Library were tested on their ability to plan and execute possible grasps in this new bin area. The planner RRTConnect was the best performer when trying to find paths that maximized the minimum path clearance. A path reachability map based on this planner and objective was constructed and used in an algorithm for predicting whether or not a path could be found to the different grasp poses. This procedure is kept separate from the neural network which supplies the grasps, and is based on the path reachability map. The algorithm evaluates a cost function weighting the metrics path length, planning time and execution time found by the planner, which is used to skew the grasps chosen for picking towards grasps reachable by the robot, and where a path exists, is efficiently planned and quickly executed. The algorithm was tested on real grasps from the neural network in simulation and the results indicate that faster "robot friendly" picking can be achieved when taking path reachability into consideration.

9.2 Future work

In the future, the algorithm should be tested on the physical system in depth to investigate if the picking time is influenced considerably by the extra check on the path reachability. The time it takes to run the algorithm and outputting the grasps in a new order in all four quadrants is theorized to be under one second, but this would need to be investigated on the physical system.

In addition, a new robot configuration to capture depth images from should be found and considered when creating the path reachability map, if re-positioning of the bin is undertaken. Considering still, that the 3D camera needs to capture point clouds 50-60cm from the bin to supply a grasp, it is preferred the robot operates in the most path reachable region of the workspace and that a weight-off between the camera distance and path reachability should be

undertaken. With the current placement of the bin, the point clouds are captured from a region of high path reachability where the robot rarely will operate, and the bin is placed sub-optimally when considering the same condition. If re-positioning of the bin is to be undertaken on the physical system, ensuring a quality pose to capture point clouds from will need to be considered also. There might then be some difficulties in reaching this type of height above the bin, since this hypothesized pose will have a lower path reachability than it has today. If the camera pose is to be changed, a new offline generation of the robot capabilities, path reachability map, must also be done, since path planning is included in the map and this pose is the start state.

Furthermore, it would be of interest to also include in the cost function, how well a grasp's likeness is to the test grasp it is coupled with in the algorithm. In this implementation, the grasp most similar in ϕ_1 and ϕ_2 was chosen without considering how much alike they were. As an example, view Figure 9.1, where two pairs of coordinate systems are shown. On the left, there are two coordinate systems where ϕ_1 are ϕ_2 are relatively small in size. The z-axes are almost coinciding, so in this instance it seems that the test grasp matches quite well the grasp from the neural network, and it is probable that the attributes of this test grasp represents the neural network grasp. Viewing the two coordinate systems on the right however, these differ quite a bit in ϕ_1 are ϕ_2, and it is not equally probable that this test grasp represents the attributes of the neural network grasp here. A suggestion for future work is thus to change the cost function from

$$J = \alpha\phi_3 + \beta\phi_4 + \delta\phi_5 \tag{9.1}$$

to something like

$$J = \chi\phi_1 + \psi\phi_2 + \alpha\phi_3 + \beta\phi_4 + \delta\phi_5, \tag{9.2}$$

where χ and ψ are weighting parameters for the Euclidean and the geodesic distance between the grasps respectively. The metrics should also be normalized by for example dividing ϕ_1 by π, since the interval of this size is $[0, \pi]$. ϕ_2, the Euclidean distance could for example be divided by the difference $d_{max} - d_{min}$.

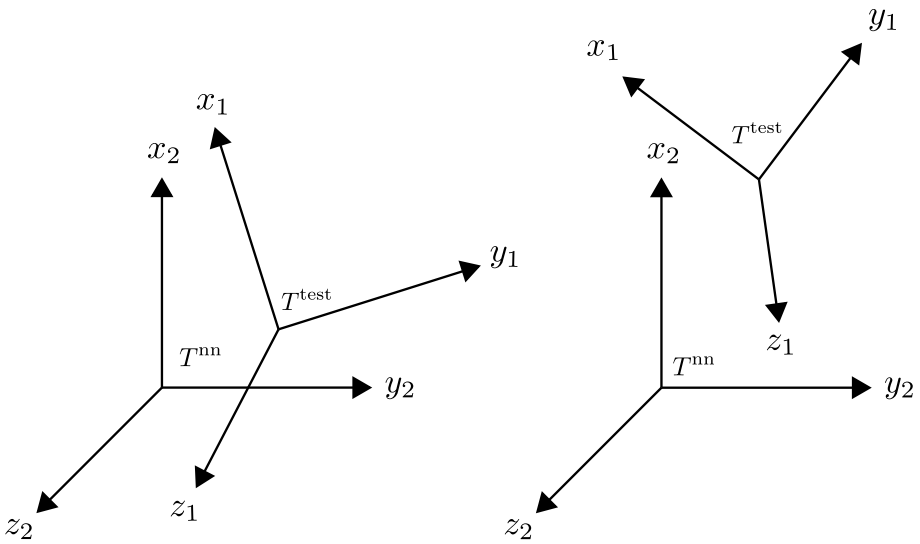


Figure 9.1: Two examples of differences in ϕ_1 and ϕ_2, motivating including these metrics in the cost function in the future to also weigh how much a test grasp resembles the neural network grasp provided.

Moreover, the distribution of objects in the bin is not uniform in height z. The bin is $7mm$ deep and the objects are $3.2cm$ in diameter. This means that the objects can overlap in both x, y and

z-direction. The reachability map currently only exists in the xy-plane at the optimal z-height in terms of path reachability. This height is at $z = 0$. Developing several path reachability maps at varying heights, covering the bin also in the z-direction, at for example mm-intervals throughout the bin, would greatly increase the resolution of the map. This could potentially improve the prediction of the path reachability of the grasps significantly.

When planning for a path, the bin should be physically introduced in the planning scene. This would allow the planner to view the sides of the bin as possible collision, and make sure that these types of collisions can be avoided. After the bin has been added, new maps, both workspace and path reachability, should be generated. The inclusion of an obstruction in the path reachability map would be a way to implicitly include IK coverage and path existence, as done in this work. In addition, the testing of more planners from OMPL or other available libraries may yield a planner outperforming the choice made in this work.

Appendix A

Article submitted to ICCMA 2019

Robotic Bin-Picking under Geometric End-Effector Constraints: Bin Placement and Grasp Selection

Irja Gravdahl[1], Katrine Seel[2] and Esten Ingar Grøtli[2]

***Abstract*— In this paper we demonstrate how *path reachability* can be taken into account when selecting among predetermined grasps in a bin-picking application, where grasps are supplied independently of the robot at hand. We do this by creating a map of the workspace to optimally place the bin with regards to the existence of an IK solution and a collision-free path, a necessary condition for systems with obstructions in the workspace. Furthermore, we densely re-map this region and based on this map predict whether a grasp is reachable by the robot. Moreover, an algorithm is implemented to weight the grasps in terms of path existence, length and time consumption. The algorithm was tested on real grasps from the neural network in simulation and the results indicate that faster picking can be achieved when taking path reachability into consideration.**

I. Introduction

Bin-picking is a concoction of technologies, and branches within those technologies. Attempting a solution of the bin-picking problem by solving it part by part seems a good strategy due to the complexity of the system as a whole. Combining solutions to subsystems is reasonably assumed to lead to the solution of the system as a whole. Following this reasoning, much research has been done on one of two things; finding high quality grasp candidates based on the geometry and pose of the objects to be picked, and finding good trajectories to reach a pose associated with the grasp. However, this dividing strategy can be problematic: it may not be feasible to reach the pose for the robot due to constraints in the workspace [1].

There exists extensive previous work on the notion of grasping an object given that the end-effector is already at the appropriate contact point to initiate the actual grasping, e.g.: [2]. A grasp may be deemed good through the appropriate metrics, for example the force closure property [3], but must be rejected if the robot cannot reach it [1].

For a given grasp, the existence of an Inverse Kinematic (IK) solution is sufficient, if the workspace is otherwise clear of obstructions [4]. However, for most practical implementations, both arm kinematics and reachability considerations are necessary.

As an example, consider the system seen in Fig. 1, which has additional constraints in terms of an eye-in-hand vision system, is placed upon a pedestal, and the bin it is to pick from constitutes an additional obstruction in the workspace. These constraints influence the robot's ability to move freely in its workspace. An inverse kinematic solution is hence only necessary, and the existence of a collision-free path to the appropriate pose must be ensured, for the grasp pose to be reachable for the robot.

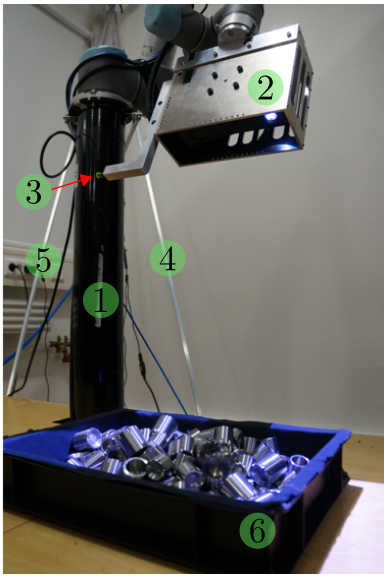


Fig. 1: Photo of the bin-picking set-up at SINTEF Digital, Trondheim. Notice the pedestal (1), camera housing (2), picker (3), fastening mechanism (4)(5), and bin with reflective steel parts (6).

The use of a grasp planner (e.g.: [5], [6] and [7]) is a popular choice to generate grasps for picking. The grasps chosen for picking in this set-up are supplied by a dual-resolution convolutional neural network trained on simulated data [8]. The input to the network is a point cloud of the current distribution of parts in the bin, and the output is multiple grasp pairs, $\{\mathbf{d}_i, \mathbf{v}_i\}$, where $i \in \{1, \ldots, N\}$, $N \in \mathbb{N}$, $\mathbf{d}_i \in \mathbb{R}^3$ is a point and $\mathbf{v}_i \in \mathbb{R}^3$ is an approach vector. The network supplies four lists of grasp pairs, one for each quadrant in the bin. A grasp is said to be valid if there are no local collisions with other objects in the bin. The output pairs from the network are ordered based on their closeness to the world z-axis and in the direction of the camera frame. The motivation behind focusing solely on grasp planning in the neural network, is that it can be robot agnostic. The implementation to include reachability considerations to re-arrange the output pairs in this paper is designed as a separate module to sustain the modular nature of the bin-picking system.

In this paper, we show that by mapping the workspace of

*The work reported in this paper was supported by the centre for research based innovation, SFI Manufacturing in Norway, Grant No. 262900. The work is partially funded by the Research Council of Norway under contract number 237900.

[1]Corresponding author: irja.gravdahl@ntnu.no, Department of Engineering Cybernetics, NTNU, Norway.

[2]SINTEF Digital Trondheim, Norway

the robot manipulator arm, positioning the bin based on this workspace mapping, and thoroughly re-mapping this space with potential grasps offline can increase picking success by placing the bin in a more accessible region in the workspace, as well as prioritizing grasps corresponding to short path lengths and time consumption. Using this mapping data in conjunction with the output from the neural network to connect these subsystems, we can re-arrange the preferred output from the neural network in terms of the robots ability to pick the objects, prioritizing grasps reachable by the robot. Mapping in this context refers to creating a map of the robot abilities. Due to the additional constraints on the system imposed by the pedestal, the camera housing, and potential collisions with the bin, in addition to an IK solution existing, a collision-free motion plan must exist for a pose to be reachable.

The contributions of this work include utilizing the combined result of IK solutions and motion-plan existence in the workspace to place the bin optimally. Optimally in this context refers to the region of the workspace with the highest concentration of IK solutions and motion-plans. Furthermore, in this optimal region, the planner LKBPIECE1 [9] (Lazy Bi-directional KPIECE with one level of discretization) with optimization objective *path length* from the Open Motion Planning Library (OMPL) [10] was investigated in terms of several metrics; path existence, path length, planning time and execution time. Moreover, a method for introducing the aforementioned metrics of the robot into the grasp selection process, without the need for explicitly querying an IK solver, path-planner or collision-checker is described.

Building on the work of [1], [11] and [12] where the existence of an IK solution is used as a criterion when selecting a grasp, we propose in this paper also to include existence of a collision-free path in the grasp selection process. This collision checking is particularly useful when considering geometric constraints, exemplified by the large volume of the 3D sensor in the system at hand.

The rest of this paper is organized in the following way; in section II, previous research on combined grasp- and path-planning will be discussed. Section III will formulate the problem to be solved, and section IV will deal with the methods used, and a discussion of the results. Lastly, the paper is concluded and future work is discussed.

II. Previous research

To the extent of the authors knowledge, there exists some ambiguity on the use of the term reachability. In [13], they define the reachability of a robot as *"its ability to move its joints and links in free space in order for its hand to reach the given target"*, indicating the term involving some movement from one state to another. However, for example in [11], [12] and [14] it refers to the existence of an inverse kinematic solution only. In the rest of this paper, reachability will refer to the existence of an IK solution, and the term *path reachability* will refer to instances where an IK solution and a collision-free path exists.

When combining motion-planning and grasp planning, there exists a substantial amount of research that either implements grasp planning in motion planners or robot capabilities in grasp planners, as a way to include reachability.

In [14], information on the robot kinematics, the local environment of the object to be grasped, and the force-closure property of the grasp is encoded in a *grasp-scoring function*, which is used to rank a precomputed set of grasps.

The use of offline generated "capability maps" for manipulators, a term used by [11] and applied to improve grasp planning in [12], is useful when incorporating robot kinematics with grasp planning. The capability map contains information about the reachability of the robot and aids in predicting if a grasp is reachable. In addition, [11] include directional preferences in the map, so that information on appropriate approach directions can be incorporated. When capturing the workspace structure in this map, the whole workspace of the robot was discretized, and sampled to obtain a uniform distribution of possible Tool Centre Point (TCP) configurations. For each of these TCP configurations, IK calculations were done, and if a solution existed, the point was marked reachable. This procedure results in a representation containing the probability of a grasp being reachable by the robot. By feeding a grasp planner this capability map, the grasp planner is decoupled from explicit implementation of the robot kinematics, but information on the success probability will be available such that unreachable grasps can be discarded early.

An integrated planner, Grasp-RRT, is presented in [16], combining the search for a valid trajectory, a feasible grasp and an inverse kinematic solution, the central elements of grasping. The method does not rely on precomputed grasps like in [11] and [12], but finds feasible grasps whilst planning a path for the robot.

In [1], a framework for workspace aware online grasp planning is provided. By using a precomputed representation of the reachable workspace called the *reachability space* where potential TCP poses are queried, they use this to bias the robot towards more reachable regions of the workspace. Planning for a grasp is only done in these more accessible regions, limiting the time spent searching for grasps in less reachable regions.

III. Problem formulation

The bin-picking system set-up used in this paper comprises a UR5 robotic manipualtor arm, a Zivid 3D camera and a vacuum gripper to pick reflective parts from a bin. To supply a grasp, it is important to obtain sufficient depth information from the images of the distribution of parts in the bin. An eye-in-hand configuration provides flexibility in this regard. Information from the camera is used to compute multiple grasps based on how objects are placed in the bin. When the sensor is attached to the robotic arm performing the grasping, additional constraints on how the manipulator can move whilst avoiding self-collisions and collisions with the bin or other parts of the environment is imposed. A characteristic of the grasps supplied by the network [8], is that they

are decoupled from the robot tasked with reaching them. The network has no knowledge about the existence, and kinematics, of the robot. This raises the issue of reachability, and the need for coupling these two aspects; optimal grasp generation in terms of the object geometry, and prioritizing grasps that are reachable for the robot.

The issue at hand is determining with what amount of ease the robot can reach a specific pose in the workspace, and to find a way to judge which grasps are favourable for reaching with the robotic manipulator arm. The following sections detail the results obtained attempting a solution to this problem. The following results were obtained in simulation, using a simulator supplied by Universal Robotics [17], and upon it, working with the Robot Operating System (ROS) workspace structure for the physical system.

IV. Methods and Results

A. Current placement of bin in workspace

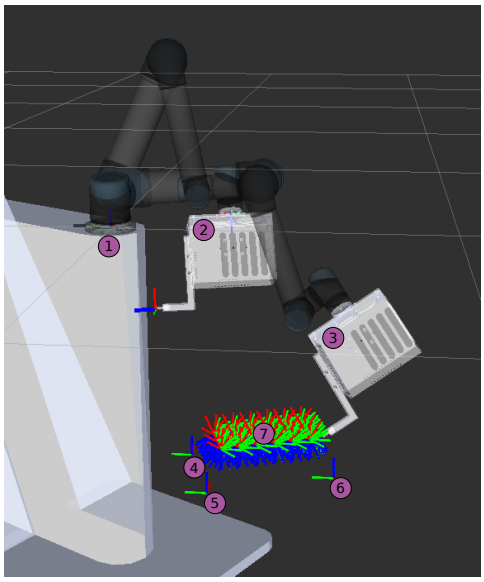


Fig. 2: Visualization of current set-up as seen in RViz. 1) pedestal where the robot is placed, 2) robot pose at the scan configuration and Zivid 3D camera, 3) visualization of the path from the scan configuration to an experimental grasp, 4,5,6) actual placement of the bin corners in the physical setup, 7) experimentally generated grasps for investigation

The current placement of the bin is based on the optimal range of the Zivid 3D camera, which is 50-60cm from the objects. This 3D camera is placed within a camera housing of substantial size (marked 2 and 3 in Fig. 2) which further limits the arm configuration space. Due to this demand, the UR5 was placed upon a pedestal, see Fig. 2. Since the output of the neural network is a point and an approach vector only, a change in joint configurations whilst keeping the TCP stationary at the point, may lead to multiple viable solutions. As a result of this characteristic, several coordinate frames were sampled with the origin at the same point, but with different orientations. It is worth noting, that since additional constraints on the system in terms of the pedestal and the camera housing were present, the need for finding a collision-free path was detrimental.

B. Expanding possible regions for bin placement

First, the level in the workspace at which the reach of the robot was the greatest, constrained by the geometry of the workspace and itself had to be identified. Utilizing RViz as a visualization tool, a larger portion of the workspace was sampled and tested. It was vital that the sampling was broad enough in all directions to capture also the limits of the workspace. Viewing Fig. 3, the portion of the workspace that was investigated can be seen. The rest of the workspace was deemed less accessible due to the layout of the robot cell, and therefore unnecessary to map. The triangular structure at the back of the robot for example, is physical.

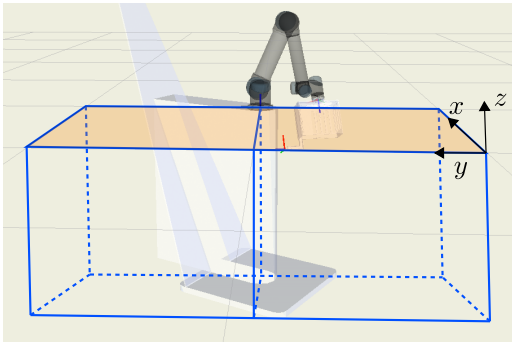


Fig. 3: Outline of the regions of the workspace being mapped. For implementation reasons the mapped region was divided in two.

The two cubes seen in Fig. 3 were discretized, and a 11×6 grid in the xy-direction, with an increment of 0.1 in the z-direction, was created. MoveIt! and OMPL [10] were used to plan a path directly to the sampled pose, succeeding only if there existed an IK solution and a collision-free path.

In each xy-plane, a 2D heatmap was produced to visualize the results. The path reachability coverage from both cubes in Fig. 3 were combined, and the optimal level in terms of best average path reachability was revealed to be the level flush with the base at $z = 0$, seen in Fig. 3 as the opaque orange level. The level map at $z = 0$ can be seen in Fig. 4. This result was not unanticipated as it is the level closest to the centre of the workspace, the base. It is also the region of the theoretical workspace with the largest span. Even though it was likely the best option, due to the imposed constraints of the pedestal and the camera housing, it had to be validated. This level had an average path reachability of 53.53%, but as can be seen from Fig. 4, there is a distinct area where the path reachability is very high. In addition to being the level with the highest average path reachability, the dark green ”patch” in the middle was also the region with the highest concentration of successes when comparing this area with the other level maps in the z-direction. For comparison, the heatmap portraying only the IK solutions at this same level is shown in Fig. 5. It is clear that the path reachability heatmap is a more conservative estimate of accessible regions in the workspace.

Fig. 4: Path reachability heatmap seen from above: Optimal level based on average hit rate along with a concentrated area with high reachability. The numbers in the squares represent percentage path reachability for the points and the colors reflect this scale from red (low) to green (high)

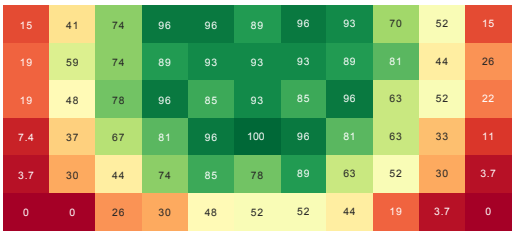


Fig. 5: Reachability heatmap seen from above: Coverage of IK solutions in the top level of the workspace. The numbers in the squares represent percentage IK solutions for the points and the colors reflect this scale from red (low) to green (high).

Note, that the representation of the optimal level as can be seen in Fig. 4 is directionless and does not take into account which approach directions are more favourable for the robot. The number in the squares are path reachability indices calculated by the following formula [11];

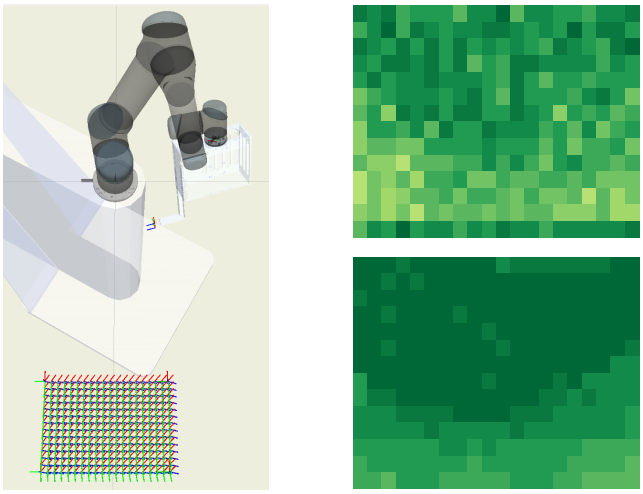
$$n_{ij} = \frac{r_{ij}}{N_{ij}} \cdot 100 \quad i,j = 1,2,\ldots,11 \tag{1}$$

where n_{ij} is the total path reachability score in percent for a point represented by a square in the level map, r_{ij} is the number of path reachable TCP frames in the point, and N_{ij} is the total number of possible TCP test frames in said point.

C. New bin placement and mapping

When placing the bin based on the workspace mapping on path reachability, the bin was placed such that the point with 100% path reachability in the middle of Fig. 4 was roughly centered in the bin. As per this point, the bin was optimally placed in both x-, y-, and z-direction, in terms of maximizing path reachability.

After positioning the bin, it was desirable to thoroughly map this area in the same manner as the workspace was mapped, and use information on potential TCP placements as a compliment to the grasps supplied by the neural network to incorporate the robot kinematics and its abilities in the grasp selection process. Since the bin and its immediate surroundings is the main region of the workspace the robot will operate in, this can be viewed as a task-specific workspace mapping. The objective was to use known, offline gathered data, to constitute an additional step in the grasp selection process by being able to guide the grasps from the network, $\{\mathbf{d}_i, \mathbf{v}_i\}$, towards easy to reach regions of the bin.



(a) New placement of bin in the workspace, with one direction of test grasps shown

(b) Comparison between one planning instance and five instances, saturating the results.

Fig. 6: New bin placement and path reachability in the area

When mapping this space, see Fig. 6a, several metrics were of interest to collect; the homogeneous transformation of the test grasp, $\mathbf{T}_j^{\text{test}}$, consisting of the rotation matrix of the test grasps given in the base frame, $\mathbf{R}_j^{\text{test}}$ and the position vector of the test grasps in the base frame $\mathbf{d}_j^{\text{test}}$, the length of the calculated path to the test grasp, the planning time needed by the planner and the execution time of the path. Provided a grasp from the neural network, we could find the test grasp that most resembles the true grasp, and then be able to return information on path length and time. Take for instance the network outputting 50 grasps in order ranked on the quality of the grasp. We then wish to rearrange this ranked list based on similar test grasps and their attributes. Note, that for both the workspace mapping (volume shown in Fig. 3) and for the mapping of the new bin placement, it was necessary to attempt a plan several times to saturate the results [18]. This is due to the randomized nature of sampling-based motion-planners, which if given enough time will find a path if one exists, but is unable to return information on the existence of a path [19]. Tests were repeated 5 times, as this seemed a sufficient number to get a representative average, while at the same time not being too time-consuming. The effect of testing once for a path, and testing five time for a path can be seen in Fig. 6b, where the average path reachability goes from 86.93% to 95.70%.

The sampling in the bin is based upon the size of the objects to be picked and their size compared to the bin. The bottom area of the bin is 26×36cm and the radius of the cylinder objects is 3.2cm. In a worst case situation with regards to surface area available for grasping, the objects will stand upright, and approximately 7×10 objects would fit on the borders. Allowing for two samples per part, there are 14×20 points sampled evenly in the bin. In each of these points, there are 27 different orientations to ensure sufficient exploration of the possibility of another joint configuration

providing a solution. This brings the number of total test grasps to 7560, for which it is possible to compare neural network grasps with, and conclude on the robots ability. In Fig. 6a the new placement of the bin is shown, along with one orientation of test grasps.

After a map of the bin space containing information on metrics of interest for each test grasp was saved in a look-up table, an algorithm which composes the path reachability test had to be implemented:

- Given the grasps from the neural network, the test grasp resembling it the most is to be identified.
- The path length, planning time and execution time for this corresponding test grasp is then looked up in the table.
- A cost function is evaluated based on the attributes of the test grasp for each grasp from the network.
- The grasps from the network are re-arranged based on this cost and returned to the picking loop.

The first step to identifying this test grasp is a check of the distance between the network grasp point and all test grasp points in the data set, and choosing the test grasp closest. So, for each grasp from the network and all the test grasps, we calculate the Euclidean distance $\phi_1 : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}_{\geq 0}$:

$$\phi_1(\mathbf{d}_i^{\text{nn}}, \mathbf{d}_j^{\text{test}}) = |\mathbf{d}_i^{\text{nn}} - \mathbf{d}_j^{\text{test}}|, \tag{2}$$

where $\mathbf{d}_i^{\text{nn}}$ is the position vector of the grasp from the neural network and $\mathbf{d}_j^{\text{test}}$ is the j^{th} test grasp position vector. After the point closest resembling the neural network grasp point is found, there are 27 options for rotation, where a comparison of the 3D rotations must be done to identify the approach angle closest in magnitude. Based upon the work of [20], the following metric, $\phi_2 : SO(3) \times SO(3) \rightarrow \mathbb{R}_{\geq 0}$, is applied to all test grasps and compared to the neural network grasps,

$$\phi_2(\mathbf{R}_i^{\text{nn}}, \mathbf{R}_j^{\text{test}}) = \left\| \log(\mathbf{R}_i^{\text{nn}} (\mathbf{R}_j^{\text{test}})^\top) \right\|, \tag{3}$$

where $SO(3)$ is the special orthogonal group of order 3, $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers, $\log(\cdot)$ is the matrix logarithm, $\mathbf{R}_j^{\text{test}}$ is the rotation part of the transformation matrix describing the test grasp j, and $\mathbf{R}_i^{\text{nn}}$ is the same for the neural network grasps. The $\|.\|$ (2-norm), gives the magnitude of the rotation angle [20]. The metric returns values in the interval $[0, \pi)$, where the objective is to find the smallest geodesic distance between the neural network grasp and test grasps. The pseudo code for the implemented algorithm is outlined in Algorithm 1.

The algorithm ensures that path reachable grasps with low computation times and short paths are prioritized over less ideal grasps, and allows the highest rated grasp to be the one best for the robot as well. Information on IK, path-existence, collision-checking and time consumption is implicitly included in the cost function J as it is calculated offline, and is accessible faster than the time it would require to do the operations sequentially for each grasp from the network.

Viewing the algorithm, ϕ_i, $i \in [3,4,5]$ refers to the normalized average values of path length, planning time and execution time of the path to the test grasps. α, β and δ are weighting parameters enabling prioritization of characteristics that are more important than others.

Algorithm 1: Algorithm for re-arranging grasps to account for robot abilities

Input: Robot capabilities in terms of test grasps and list of grasps from neural network
Output: Re-arranged list of grasps in terms of path reachability

```
for all grasp from neural network T_i^nn do
    for all test grasps T_j^test do
        calculate φ1(d_i^nn, d_j^test) and φ2(R_i^nn, R_j^test);
    end
    return the closest test grasp in φ1 and φ2;
end
for each test grasp corresp. to a network grasp do
    if the grasp is path reachable then
        calculate cost of each grasp;
        J_i = αφ3 + βφ4 + δφ5;
    else
        J_i = ∞;
    end
end
sort grasps with respect to J;
```

D. Testing

The grasps from the neural network are given relative to the current placement of the bin, the placement seen in Fig. 2. Transforming the grasps to the new bin, as shown in Fig. 6a, enables testing of the algorithm.

To evaluate the functionality of the algorithm, all weights, α, β and δ were set to 1 such that path length, planning time and execution time of the paths were considered equally important, since these three aspect influence how fast a full picking cycle is.

A comparison between the list of grasps from the neural network and the re-arranged list of grasps when considering path reachability, had to be investigated. To do so, the first 10 grasps from the network, unsorted on path reachability, and the first 10 grasps from the re-arranged list were recorded. The average cost $J = \alpha\phi_3 + \beta\phi_4 + \delta\phi_5$ of these groups was calculated and plotted in a bar chart, seen in Fig. 7. There is one plot for each quadrant in the bin.

Viewing the figure, the columns to the right in each group represent the average cost of the first 10 grasps from the network, not considering path reachability. The columns to the left in each group represent the average cost of the first 10 grasps of the sorted list. The value of 10 was chosen to obtain a representative average when comparing the cost J. When comparing the two, it can be seen that when including path reachability, grasps with a short path length, planning time and execution time, are prioritized. Note that the y-axis goes to infinity since some of the grasps preferred by the network were unreachable by the robot in the south west

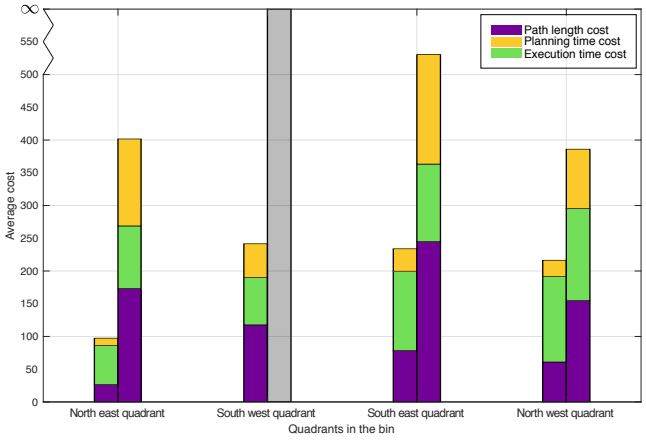


Fig. 7: Comparison of the average cost of the first 10 grasps from the neural network (column to the right) and the first 10 grasps from the re-arranged list based on path reachability (column to the left) The average cost is split into the average sub costs of path length, planning time and execution time for the grasps.

quadrant of the bin, this is illustrated in Fig 7 by the grey column. When comparing the results for the south-east and south-west regions in the bin in terms of a higher cost, this is consistent with the slightly lower level of path reachability in these regions as can be seen in Fig 6b.

The results indicate that faster picking can be achieved when including path reachability as a separate module in the bin-picking system, whilst still keeping the neural network focused on grasp planning alone.

V. Conclusions

In this work, a mapping of the workspace of the system shown in Fig. 1 has been undertaken. Based on this mapping, the region of the workspace with the highest *path reachability* was chosen as the best location for placing the bin, when considering that a robot needs to reach the objects to be picked. Due to the constraints on the end-effector, and the constraints in the robot cell such as the pedestal and the bin, collision-free paths had to be found and considered. After placing the bin, an algorithm for predicting whether or not a path could be found to the grasp pose was implemented on the system, separate from the neural network which supplies the grasps. The algorithm evaluates a cost function including metrics such as path length, which is used to skew the grasps chosen for picking towards grasps reachable by the robot, and where a path exists. The algorithm was tested on real grasps from the neural network in simulation and the results indicate that faster picking can be achieved when taking path reachability into consideration.

In the future, the algorithm should be tested on the physical system in depth to investigate if the picking time is influenced considerably by the extra check on the path reachability. In addition, a new robot configuration to capture depth images from should be found and considered when creating the path reachability map, if re-positioning of the bin is undertaken. Furthermore, it would be of interest to also include in the cost function, how well a grasp's likeness is to the test grasp it is coupled with in the algorithm.

References

[1] I. Akinola, J. Varley, B. Chen, and P. K. Allen, "Workspace aware online grasp planning," In IROS, 2018, pp. 2917-2924.

[2] D. Kraft, L.-P. Ellekilde, and J. A. Jørgensen, "Automatic Grasp Generation and Improvement for Industrial Bin-Picking," In *Gearing up and accelerating cross-fertilization between academic and industrial robotics research in Europe*:, Cham, 2014, pp. 155-176: Springer International Publishing.

[3] A. Bicchi, "On the Closure-Properties of Robotic Grasping," *International Journal of Robotics Research*, vol. 14, no. 4, Aug., pp. 319-334, 1995.

[4] J. P. Saut and D. Sidobre, "Efficient models for grasp planning with a multi-fingered hand," *Robotics and Autonomous Systems*, vol. 60, no. 3, Mar., pp. 347-357, 2012.

[5] A. T. Miller and P. K. Allen, "Graspit! A versatile simulator for robotic grasping," IEEE Robotics & Automation Magazine, vol. 11, no. 4, pp. 110-122, 2004.

[6] R. Diankov and J. Kuffner, "OpenRAVE: A Planing Architecture for Autonomous Robotics," Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 152132008, vol. 79.

[7] B. León et al., "OpenGRASP: A Toolkit for Robot Grasping Simulation," In Simulation, Modeling, and Programming for Autonomous Robots, 2010, pp. 109-120.

[8] J. S. Dyrstad, M. Bakken, E. I. Grøtli, H. Schulerud, and J. R. Mathiassen, "Bin Picking of Reflective Steel Parts Using a Dual-Resolution Convolutional Neural Network Trained in a Simulated Environment," In IEEE International Conference on Robotics and Biomimetics (ROBIO), 2018, pp. 530-537.

[9] I. A. Şucan and L. E. Kavraki, "Kinodynamic Motion Planning by Interior-Exterior Cell Exploration," In *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*, G. S. Chirikjian, H. Choset, M. Morales, and T. Murphey, Eds. 2010, pp. 449-464.

[10] Ioan A. Şucan, Mark Moll, Lydia E. Kavraki, The Open Motion Planning Library, IEEE Robotics & Automation Magazine, 19(4):72–82, December 2012. http://ompl.kavrakilab.org

[11] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: representing robot capabilities," In IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007, pp. 3229-3236.

[12] F. Zacharias, C. Borst, and G. Hirzinger, "Online generation of reachable grasps for dexterous manipulation using a representation of the reachable workspace," In International Conference on Advanced Robotics, 2009, pp. 1-8.

[13] Z. Y. Ying and S. S. Iyengar, "Robot Reachability Problem - a Nonlinear Optimization Approach," *Journal of Intelligent & Robotic Systems*, vol. 12, no. 1, pp. 87-100, 1995.

[14] D. Berenson, R. Diankov, N. Koichi, K. Satoshi, and J. Kuffner, "Grasp planning in complex scenes," In 7th IEEE-RAS International Conference on Humanoid Robots, 2007, pp. 42-48.

[15] J. Fontanals, B. Dang-Vu, O. Porges, J. Rosell, and M. A. Roa, "Integrated grasp and motion planning using independent contact regions," In IEEE-RAS International Conference on Humanoid Robots, 2014, pp. 887-893.

[16] N. Vahrenkamp, M. Do, T. Asfour, and R. Dillmann, "Integrated Grasp and motion planning," In IEEE International Conference on Robotics and Automation, 2010, pp. 2883-2888.

[17] https://www.universal-robots.com/. [Accessed: 20.05.19]

[18] J. Meijer, Q. Lei, and M. Wisse, "Performance study of single-query motion planning for grasp execution using various manipulators," In 18th International Conference on Advanced Robotics (ICAR), 2017, pp. 450-457.

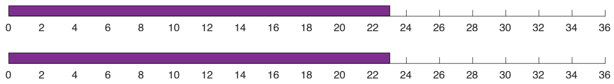
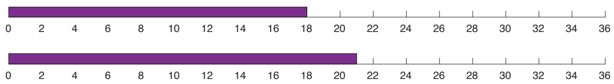
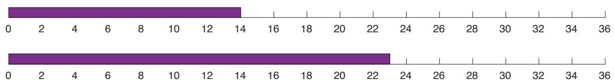
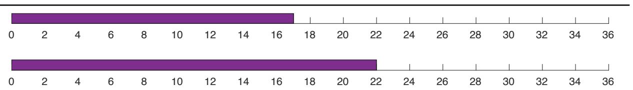
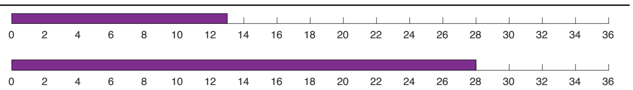
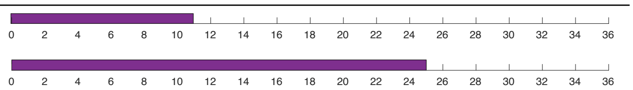
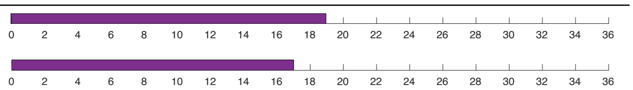
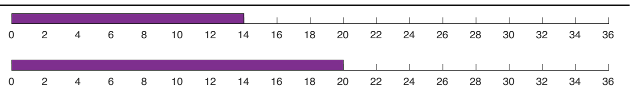
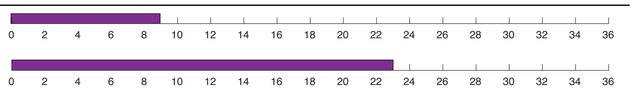
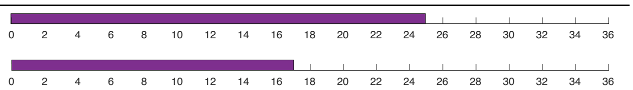
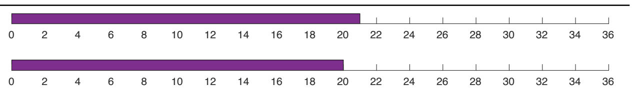
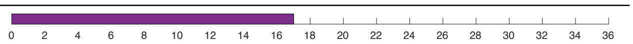
[19] L. E. Kavraki and S. M. LaValle, "Motion Planning," in Springer Handbook of Robotics, B. Siciliano and O. Khatib, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 109-131.

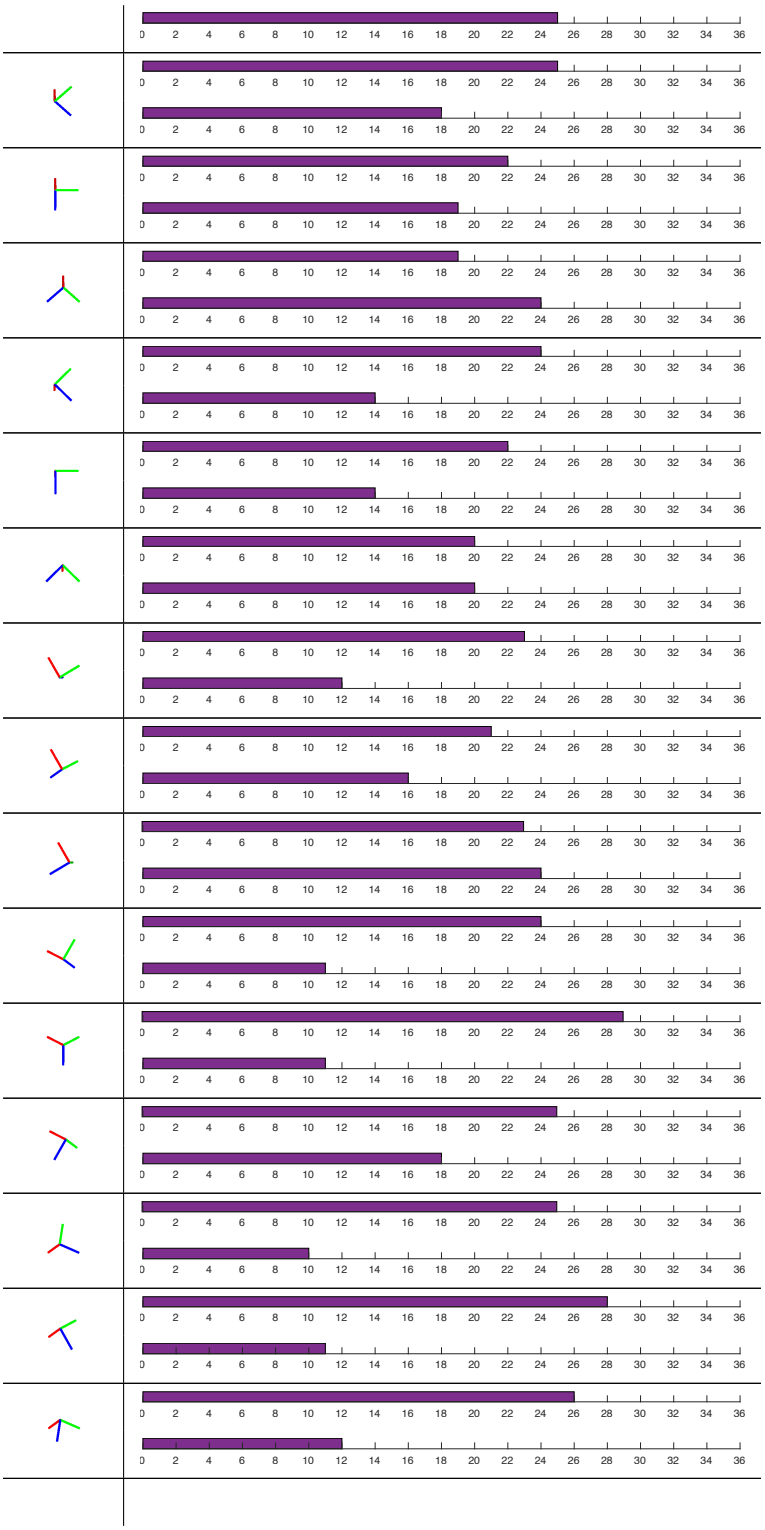
[20] D. Q. Huynh, "Metrics for 3D Rotations: Comparison and Analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155-164, 2009.

Appendix B

Complete table concerning rotations

Table B.1: Overview of approach directions in the optimal level in terms of path reachability (complete)

Grasp	Statistics
	
	
	
	
	
	
	
	
	
	
	
	


0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36

Bibliography

Akinola, I., Varley, J., Chen, B. and Allen, P. K. (2018). Workspace aware online grasp planning, *International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 1.-5. Oct., pp. 2917–2924.

Bajd, T., Mihelj, M., Lenarčič, J., Stanovnik, A. and Munih, M. (2010). *Robotics*, Vol. 43 of *Intelligent Systems, Control, and Automation: Science and Engineering*, Springer.

Berenson, D., Diankov, R., Koichi, N., Satoshi, K. and Kuffner, J. (2007). Grasp planning in complex scenes, *2007 7th IEEE-RAS International Conference on Humanoid Robots*, Pittsburgh, PA, USA, 29. Nov - 1. Dec, pp. 42–48.

Bernstein, D. (2018). *Scalar, Vector, and Matrix Mathematics: Theory, Facts, and Formulas - Revised and Expanded Edition*, Princeton University Press, New Jersey.

Bicchi, A. and Kumar, V. (2000). Robotic grasping and contact: a review, *IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, USA, 24.-27. Apr., pp. 348–353.

Bogue, R. (2014). Random bin picking: has its time finally come?, *Assembly Automation* **34**(3): 217–221.

Bohlin, R. and Kavraki, L. E. (2000). Path planning using lazy prm, *IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, USA, 24.-27. Apr., pp. 521–528.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2009). *Introduction to Algorithms*, 3rd edn, The MIT Press, Cambridge, MA, USA.

Diankov, R. and Kuffner, J. (2008). Openrave: A planing architecture for autonomous robotics, *Technical report*, Robotics Institute, Carnegie Mellon University. Pittsburgh, Pennsylvania 15213, CMU-RI-TR-08-34.

Dyrstad, J. S., Bakken, M., Grøtli, E. I., Schulerud, H. and Mathiassen, J. R. (2018). Bin picking of reflective steel parts using a dual-resolution convolutional neural network trained in a simulated environment, *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Kuala Lumpur, Malaysia, 12.-15. Dec., pp. 530–537.

Egeland, O. and Gravdahl, J. T. (2002). *Modeling and Simulation for Automatic Control*, Marine Cybernetics, Trondheim, Norway.

Ferrari, C. and Canny, J. (1992). Planning optimal grasps, *IEEE International Conference on Robotics and Automation (ICRA)*, Nice, France, 12.-14. May, pp. 2290–2295.

Fontanals, J., Dang-Vu, B., Porges, O., Rosell, J. and Roa, M. A. (2014). Integrated grasp and

motion planning using independent contact regions, *International Conference on Humanoid Robots*, Madrid, Spain, 18.-20. Nov., pp. 887–893.

Foote, T. (2013). tf: The transform library, *Conference on Technologies for Practical Robot Applications (TePRA)*, Open-Source Software workshop, Woburn, MA, USA, 22.-23. Apr., pp. 1–6.

Gravdahl, I. (2018). Grasp selection in bin picking tasks for robotic manipulator arm with end-effector geometric constraints, *Report*, Norwegian University of Science and Technology (NTNU). Specialization project.

Gravdahl, I., Grøtli, E. I. and Seel, K. (2019). Robotic bin-picking under geometric end-effector constraints: Bin placement and grasp selection. Submitted.

Haustein, J. A., Hang, K. and Kargic, D. (2017). Integrating motion and hierarchical fingertip grasp planning, *IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, 29. May - 3. Jun.

Huynh, D. Q. (2009). Metrics for 3d rotations: Comparison and analysis, *Journal of Mathematical Imaging and Vision* **35**(2): 155–164.

Jaillet, L., Cortés, J. and Siméon, T. (2010). Sampling-based path planning on configuration-space costmaps, *IEEE Transactions on Robotics* **26**(4): 635–646.

Kavraki, L. E. and LaValle, S. M. (2008). Motion planning, *in* B. Siciliano and O. Khatib (eds), *Handbook of Robotics*, Springer, Berlin, Heidelberg, pp. 109–131.

Kavraki, L. E., Svestka, P., Latombe, J. and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *IEEE Transactions on Robotics and Automation* **12**(4): 566–580.

Kenny, D. (1987). *Statistics for the Social and Behavioral Sciences*, Little, Brown and Company, Canada.

Kessens, C. C. and Desai, J. P. (2011). A self-sealing suction cup array for grasping, *Journal of Mechanisms and Robotics* **3**(4): 045001–045001–8.

Kraft, D., Ellekilde, L.-P. and Jørgensen, J. A. (2014). Automatic grasp generation and improvement for industrial bin-picking, *in* F. Rörbhein, G. Veiga and C. Natale (eds), *Gearing up and accelerating cross-fertilization between academic and industrial robotics research in Europe*, Tracts in Advanced Robotics, Springer, pp. 155–176.

Kuffner, J. J. and LaValle, S. M. (2000). RRT-connect: An efficient approach to single-query path planning, *IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, USA, 24.-28. Apr., pp. 995–1001.

LaValle, S. M. (2006). *Planning Algorithms*, Cambridge University Press, Cambridge, UK.

León, B., Ulbrich, S., Diankov, R., Puche, G., Przybylski, M., Morales, A., Asfour, T., Moisio, S., Bohg, J., Kuffner, J. and Dillmann, R. (2010). Opengrasp: A toolkit for robot grasping simulation, *in* N. Ando, S. Balakirsky, T. Hemker, M. Reggiani and O. von Stryk (eds), *Simulation, Modeling, and Programming for Autonomous Robots*, Springer Berlin Heidelberg, pp. 109–120.

Marvel, J. A., Saidi, K., Eastman, R., Hong, T., Cheok, G. and Messina, E. (2012). Technology readiness levels for randomized bin picking, *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, ACM, College Park, MD, USA, 20.-22. Mar., pp. 109–113.

MathWorks (2019). MATLAB. [Accsesed: 03.07.19].
URL: *https://se.mathworks.com/help/matlab/*

Meijer, J., Lei, Q. and Wisse, M. (2017). Performance study of single-query motion planning for grasp execution using various manipulators, *18th International Conference on Advanced Robotics (ICAR)*, Hong Kong, 10.-12. Jul., pp. 450–457.

Miller, A. T. and Allen, P. K. (2004). Graspit! a versatile simulator for robotic grasping, *IEEE Robotics & Automation Magazine* **11**(4): 110–122.

MoveIt (2019). Moveit planning framework. Online available documentation on MoveIt concepts and tutorials, open-source. [Accessed: 2018/2019].
URL: *https://moveit.ros.org/*

ROS (2019). Doumentation. Online available documentation on ROS, open-source. [Accessed: 2018/2019].
URL: *http://wiki.ros.org/*

Russell, S. and Norvig, P. (2016). *Artificial intelligence, A modern approach*, Prentice hall series in artificial intelligence, 3rd edn, Pearson, Essex, England.

Saut, J.-P. and Sidobre, D. (2012). Efficient models for grasp planning with a multi-fingered hand, *Robotics and Autonomous Systems* **60**(3): 347–357.

Spong, M. W., Hutchinson, S. and Vidyasagar, M. (2006). *Robot modeling and control*, John Wiley & Sons Inc, New Jersey, USA.

Şucan, I. A. and Kavraki, L. E. (2010). Kinodynamic motion planning by interior-exterior cell exploration, *in* G. S. Chirikjian, H. Choset, M. Morales and T. Murphey (eds), *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*, Springer, Berlin, Heidelberg, pp. 449–464.

Şucan, I. A., Moll, M. and Kavraki, L. E. (2012). The Open Motion Planning Library, *IEEE Robotics & Automation Magazine* **19**(4): 72–82.
URL: *http://ompl.kavrakilab.org*

Sánchez, G. and Latombe, J.-C. (2003). A single-query bi-directional probabilistic roadmap planner with lazy collision checking, *in* R. A. Jarvis and A. Zelinsky (eds), *Robotics Research*, Springer Berlin Heidelberg, pp. 403–417.

Universal Robots A/S (2016). *Universal Robots User Manual UR5/CB3*, 3.3.3 edn, Universal Robots A/S.

Universal Robots A/S (2019). Universal Robots support, Downloads. Download of UR5 simulator [Accessed: 20.01.19].
URL: *https://www.universal-robots.com/download/*

Vahrenkamp, N., Asfour, T. and Dillmann, R. (2013). Robot placement based on reachability inversion, *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhle, Germany, 6.-10.- May., pp. 1970–1975.

Vahrenkamp, N., Do, M., Asfour, T. and Dillmann, R. (2010). Integrated grasp and motion planning, *IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, 3.-7. May, pp. 2883–2888.

Vasilash, G. S. (2017). Beyond dirty, dull and dangerous, *Automotive Design & Production* **129**(5): 6.

Walpole, R. E., Myers, R. H. and Myers, S. L. (1998). *Probability and statistics for engineers and scientists*, 6th edn, Prentice Hall International Inc., New Jersey.

Ying, Z. and Iyengar, S. S. (1995). Robot reachability problem: A nonlinear optimization approach, *Journal of Intelligent & Robotic Systems* **12**(1): 87–100.

Zacharias, F., Borst, C. and Hirzinger, G. (2007). Capturing robot workspace structure: representing robot capabilities, *International Conference on Intelligent Robots and Systems*, San Diego, CA, USA 29. Oct. - 2.Nov., pp. 3229–3236.

Zacharias, F., Borst, C. and Hirzinger, G. (2009). Online generation of reachable grasps for dexterous manipulation using a representation of the reachable workspace, *International Conference on Advanced Robotics*, Munich, Germany, 22. - 26. Jun., pp. 1–8.

Gravdahl

Master thesis 2019

NTNU
Norwegian University of
Science and Technology